Universitat Autònoma de Barcelona

Efficient Neural Network Inference for Resource Constrained Devices

Juan BORREGO-CARAZO

ADVERTIMENT. L'accés als continguts d'aquesta tesi queda condicionat a l'acceptació de les condicions d'ús establertes per la següent llicència Creative Commons: http://cat.creativecommons.org/?page_id=184

ADVERTENCIA. El acceso a los contenidos de esta tesis queda condicionado a la aceptación de las condiciones de uso establecidas por la siguiente licencia Creative Commons: http://es.creativecommons.org/blog/licencias/

WARNING. The access to the contents of this doctoral thesis it is limited to the acceptance of the use conditions set by the following Creative Commons license: https://creativecommons.org/licenses/?lang=en

UNIVERSITAT AUTÒNOMA DE BARCELONA

PHD PROGRAMME IN ELECTRONIC AND
TELECOMMUNICATION ENGINEERING

Efficient Neural Network Inference for Resource Constrained Devices

---

Author:
Juan BORREGO-CARAZO

Director:
Dr. Jordi CARRABINA,
Dr. David
CASTELLS-RUFAS

Tutor:
Dr. Jordi CARRABINA

A thesis submitted in fulfillment of the requirements
for the degree of Doctor of Philosophy

in the

Center for Hardware-Software Prototypes and Solutions
Microelectronics and Electronic Systems Department

Declaration of Authorship

I, Juan BORREGO-CARAZO, declare that this thesis titled, “Efficient Neural Network Inference for Resource Constrained Devices” and the work presented in it are my own. I confirm that:

- This work was done wholly or mainly while in candidature for a research degree at this University.
- Where any part of this thesis has previously been submitted for a degree or any other qualification at this University or any other institution, this has been clearly stated.
- Where I have consulted the published work of others, this is always clearly attributed.
- Where I have quoted from the work of others, the source is always given. With the exception of such quotations, this thesis is entirely my own work.
- I have acknowledged all main sources of help.
- Where the thesis is based on work done by myself jointly with others, I have made clear exactly what was done by others and what I have contributed myself.

Signed:

Date:

JUAN
BORREGO
CARAZO -
DNI
47236869K

Digitally signed
by JUAN
BORREGO
CARAZO - DNI
47236869K
Date: 2022.09.14
19:55:17 +02'00'

"Quien sabe lo que busca encuentra lo que quiere"

Yónatan Calderón

"Who are you, who are so wise in the ways of science?"

Sir Bedivere

"Do you know how frustrating it is to have to translate everything in my head before I say it?
Do you even know how smart I am in Spanish?"

Gloria

"Oír, ver y callar, recías cosas son de obrar"

Refrán español

UNIVERSITAT AUTÒNOMA DE BARCELONA

Abstract

Engineering School

Microelectronics and Electronic Systems Department

Doctor of Philosophy

Efficient Neural Network Inference for Resource Constrained Devices

by Juan BORREGO-CARAZO

Last decade advances in deep learning have supposed a great leap in state-of-the-art results with regard to tasks such as image classification, language translation, and many others. However, with such success, there has been a related increase in model complexity and size, which has incremented the hardware requirements both for training and inference (both generally and initially limited to GPUs). Moreover, the hardware capabilities (OPS performance, memory, throughput, latency, energy) have supposed an initial limitation to deploying applications in resource-constrained platforms and applications, such as mobile or embedded platforms.

There have been many initiatives to reduce training time, and energy costs, and improve data efficiency during the development phase. Equally, there has also been profound research to optimize deep learning models with a focus on inference and deployment: decreasing model complexity, size, latency, and memory consumption. In such direction, there are five optimization methods that have stood out: pruning, quantization, neural architecture search, efficient operations, and distillation.

In parallel, in order to enable inference deployment in specialized hardware platforms, new frameworks have appeared (such as CMSIS-NN or uTensor for MCUS, and TF Lite for mobile platforms). Those frameworks include several features for the deployment of models, but most importantly, the crucial point is if they support the specific model operations and optimizations, ensuring the final application deployment.

All in all, from optimization procedures to conversion and deployment frameworks, the procedure of developing efficient NN-based models and deploying them to constrained hardware has certainly improved, albeit with still some limitations. In such a sense, this thesis is framed by such improvements and limitations: first, with the development and improvement of NN optimization techniques, and second, with the use and development of software for porting the optimized models. All with a special focus on three industrial and practical cases which are the main drivers of the developments: automotive human-machine interaction, ITM in mobile devices, and bronchoscopy guidance.

In the first case, we show the deployment and optimization of RNNs in MCUs, as well as the usage and improvement of Bayesian optimization and NAS methods to deliver minimal but well-performing networks. Altogether we deliver a framework for automatically converting and deploying networks in Cortex-M-based MCUs. In the second environment, we employ quantization and efficient operations to bring an ITM network to mobile devices for efficient inference, providing improvements in latency up to 100x with only 3% accuracy loss. Finally, we develop an efficient bronchoscopy guidance network with structured pruning and efficient operations, that provides a reduction of x4 of NN size and an improvement of ≈14% in accuracy for position localization.

Acknowledgements

I would like to especially thank my thesis directors Prof. Dr Jordi Carrabina Bordoll and Dr. David Castells-Rufas for the opportunity, guidance, and orientation. Also, for all the freedom and support in conducting research, that, in the end, has spammed many topics, companies, and ventures, and has been the seed for its final success and completion.

Special gratitude is devoted to Ernesto Biempica, who guided and help me in the first and most important phase of this thesis at Kostal Eléctrica. He let me grow and learn, not only about technical topics but also about business relationships, goals, and philosophy.

Author thanks the staff of Samsung R&D UK for the research stage opportunity, with special gratitude to the distinguished researcher, Dr. Mete Ozay, for his support and guidance during my research stage. I also would like to thank Paul Wisbey, Frederik Laboyrie, and Cristian Szabo for their support during the project, as well as Albert Saa-Garriga, Karthikeyan Saravanan, and Daniel Ansorregui for their support and orientation.

Special thanks also to Masoud Daneshtalab and Mohammad Loni, for the opportunity of conducting a research stage at Mälardalen University. Also to all the people that I met: I had an excellent time in the middle of the weary pandemic.

Thanks also to Prof. Dr. Débora Gil Resina and Dr. Carles Sánchez, researchers at Centre de Visió per Computador at Universitat Autònoma de Barcelona. Thanks for the opportunity in the last project of the thesis, and for teaching me more about research and collaboration.

Gratitude to Jordi Naqui, Sergio Bravo, Marc Bernabé and other Kostal members. They helped and supported me during the first steps of this project.

To the whole Microelectronic and Electronics Department, for the support and research.

A la suerte y la casualidad, compañeras. No estaría aquí si no fuese por una llamada del Prof. Dr. Jordi Carrabina mientras yo estaba en Villaciervos con mi tío Fernando viendo antiguas casas de pastores allá por el 2018. Una llamada que, en mayor o menor medida, cambió el curso de los años por venir.

Pero ante todo, y sobre todo, a mi famila, la que está, y la que ya no está. En especial a Clara Pons Duran, Maria Cruz Carazo Revuelto, Manuel Borrego

Carazo, Francisco Borrego Pastor, Felicidad Revuelto Renta, Fernando Carazo Revuelto y Vicente Carazo Revuelto. A todos, gracias.

List of Publications

The following journal articles [1, 2] were published and are included in the thesis as publications:

- J. Borrego-Carazo, D. Castells-Rufas, E. Biempica, and J. Carrabina, “Resource-Constrained Machine Learning for ADAS: A Systematic Review,” *IEEE Access*,vol. 8, pp. 40573–40598, 2020.
- D. Castells-Rufas, J. Borrego-Carazo, J. Carrabina, J. Naqui, and E. Biempica,“Continuous touch gesture recognition based on RNNs for capacitive proxim-ity sensors,” *Personal and Ubiquitous Computing*, Nov. 2020
- J. Borrego-Carazo, C. Sánchez, D. Castells-Rufas, J. Carrabina, D. Gil, "BronchoPose: an analysis of data and model configuration for vision-based bronchoscopy pose estimation", submitted to *Computer Methods and Programs in Biomedicine*, 2022.

The following conference papers [3, 4, 5, 6] were published and are included in the thesis as publications:

- J. Borrego-Carazo, D. Castells-Rufas, J. Carrabina, and E. Biempica, “Capacitive-sensing module with dynamic gesture recognition for automotive applications,” in *2020 23rd International Symposium on Design and Diagnostics of Electronic Circuits & Systems (DDECS)*, pp. 1–6, IEEE, 2020.
- J. Borrego-Carazo, D. Castells-Rufas, J. Carrabina, and E. Biempica, “Extending SpArSe: Automatic Gesture Recognition Architectures for Embedded Devices,” in *2020 19th IEEE International Conference on Machine Learning and Applications(ICMLA)*, pp. 7–12, IEEE, 2020.
- J. Borrego-Carazo, D. Castells-Rufas, E. Biempica, and J. Carrabina, “Unsupervised Embedded Gesture Recognition based on multi-objective NAS and capacitive sensing,” *IFSA Publishing*, Nov. 2020.
- J. Borrego-Carazo, M. Ozay, F. Laboyrie, and P. Wisbey. “A Mixed Quantization Network for Computationally Efficient Mobile Inverse Tone Mapping.” British Machine Vision Association, 2021.
- J. Borrego-Carazo, C. Sánzhez, D. Castells-Rufas, J. Carrabina, D. Gil, "A Benchmark for the evaluation of computational methods for bronchoscopic navigation.", Computer Assisted Radiology and Surgery (CARS), 2022.

The following publication [6] have been published, with the author of the thesis as co-author:

- Gil, D., Hernàndez-Sabaté, A., Enconniere, J., Asmayawati, S., Folch, P., Borrego-Carazo, J., & Piera, M. À. (2021). E-Pilots: A System to Predict Hard Landing During the Approach Phase of Commercial Flights. IEEE Access, 10, 7489-7503.

Contents

Declaration of Authorship **iii**

Abstract **viii**

Acknowledgements **xi**

List of Publications **xiii**

List of Figures **xvii**

List of Tables **xxiii**

1 Introduction **1**
- 1.1 Brief historic background of neural networks in AI 2
- 1.2 Why efficient neural networks? 5
- 1.3 Neural Network Optimization 8
- 1.4 NN frameworks for resource-constrained hardware 12
- 1.5 About this thesis 15

2 Applied Optimization Methods **19**
- 2.1 Quantization 19
- 2.2 Neural Architecture Search 27
- 2.3 Pruning 35
- 2.4 Efficient operations 44

3 Deployment frameworks for Resource-Constrained Hardware Platforms **53**
- 3.1 Microcontroller Units (MCUs) 56
- 3.2 Mobile Platforms 60
- 3.3 Embedded GPU boards 63
- 3.4 Other platforms 64

4 Capacitive Sensing based Gesture Recognition on MCUs **67**

4.1 Introduction . . . 68
4.2 1D Gesture Recognition . . . 73
4.3 SpArSe Extension . . . 93
4.4 2D Gesture Recognition . . . 102
4.5 Conclusions . . . 109

5 Inverse Tone Mapping on Smartphones **111**
5.1 Introduction . . . 112
5.2 Background . . . 113
5.3 A Mixed Quantization Network for Mobile ITM . . . 115
5.4 Experimental Analyses . . . 119
5.5 Conclusion . . . 133

6 Bronchoscopy Navigation **135**
6.1 Introduction . . . 135
6.2 Methodology . . . 138
6.3 Experiments and results . . . 145
6.4 Conclusions . . . 150

7 Conclusions and Future Directions **153**
7.1 Conclusions . . . 153
7.2 Future directions . . . 155

8 Bibliography **157**

List of Figures

1.1 NN word trends from the 1940s until the end of 2010s. We can see, more or less distinctively, the different phases: first, cybernetics, second, connections, and finally deep learning era. Note: cybernetics has separated from NNs meaning. 3
1.2 (Top) Illustration of increasing complexity and size of models pursuing a higher accuracy (acc.) over the years. FPO stands for floating point operations. Image taken from [7] with permission from the authors. (Bottom) Illustration of the size and operational complexity of different networks versus the attained accuracy. Image taken from [8] with permission from the authors. 6
1.3 Proposal of organization of methods for accelerating or enabling NN inference in resource-constrained devices. The three most important branches are optimization, architecture design, and hardware acceleration. 9
1.4 Two illustrations of the diverse components and steps of scientific development: from research to commercialization. Picture (A) broadly illustrates the 4 main areas of scientific development and the organizations involved in them. Taken from [9] with permission of the authors. The present thesis is placed between applied research and the application of knowledge. Picture (B) states the different Technology Readiness Levels (TLRs). The current thesis spans from TLR 2, and as far as TRL 7. Picture of public domain from Wikipedia, Technology Readiness Level. 14

2.1 Example of the different accuracies achieved with post-training quantization depending on the type of quantization for Mobilenet V1 and V2 (mvv1-mv2) and Resnet networks. Figure adopted from [10] 24

2.2 Difference in accuracy between quantized from scratch models or models quantized from a floating point trained model. Figure adopted from [10]. 25

2.3 (Left) Illustrative simplification of the pruning procedure targeting connections and neurons in a fully connected network. (Right) Visual example of shape-wise pruning [11]. Image adopted from [12]. 36

2.4 Graphic illustration of the three choices for scheduling pruning: one-shot, iterative, and progressive. Adapted from [13]. 41

2.5 Example of efficient operations. (A) Residual connection [14]. (B) Depth-wise separable convolution [15]. (C) Depthwise separable convolution with inverted residual and linear bottleneck (right) and without residual (left) [16]. (D) Shuffle blocks, with (right) or without (left) spatial resolution reduction [17]. 47

2.6 Illustration of the internal operations for both LSTM (left) and GRU (right). Illustration taken from [2] 49

2.7 Number of operations for both the LSTM and GRU RNN cells as a function of the hidden state size, n_h. 49

2.8 Latency vs Top-1 Accuracy for different models using batch size 1 on an NVIDIA Jetson TX1. Image taken from [8] with permission from the authors. 50

3.1 Number of Giga operations per second versus power consumption of publicly available AI accelerators and processors. Figure extracted from [18] with permission from the authors. 55

4.1 Examples of normalized capacitance reading to various events over time. 70

4.2 Different proximity sensor principles. (A) Self-capacitance sensing. (B) Mutual capacitance sensing. 74

4.3 Capacitive functional foil for touch sensing. Sensitized areas are depicted as light blue rectangles. 76

4.4 Capacitive sensor readout system. 76

4.5 Example of the frame-based recognition approach for an input stream of 30 samples in which a tap gesture occurs on sample 12. The input samples X are grouped in 5 frames of 6 samples each. Frames 3 and 4 are classified as belonging to a Tap gesture for sensor 1, while other frames are classified as non-gesture . . 80

4.6 Dataset annotations for different gestures in different conditions. (A) tap gesture with a neoprene glove. (B) normal tap on button 2. (C) finger approach, which is not considered a valid gesture. (D) swipe right gesture. 83
4.7 Detail of the RNN structure for any cell type. The input at each step is a sample, x, with the corresponding features for each sensor. The output of the class is for the whole sequence. (Left) Example of RNN-based system with 1 layer. (Right) A system with 3 layers. 84
4.8 (A) Plot of the training procedure for the best LSTM configuration of the optimization procedure. Loss in blue and average recall (AR) of the touch classes in green. Dashed lines correspond to validation (Val.) set values and solid to training set (Tr.) values. The minimum value of the validation loss delimits the transition edge into the over-fitting region. (B) Time of all the trials run during the optimization process. 90
4.9 (A) Evolution of the minimum achieved error during the optimization process. After the first 2 h of running the optimization process, the error reductions are harder to achieve. (B) Evolution of the size of the parameter during the optimization process. The size of the model is often increased to achieve lower error rates, but big error reductions are also achieved by the change of other hyperparameters. 91
4.10 Confusion matrix for the best LSTM network design. 92
4.11 Confusion matrix for the best GRU network design. 93
4.12 Industrial design of the final product. From top to bottom (1) microcontroller PCB, (2) capacitive sensor foil, and (3) mechanical enclosure. 94
4.13 Example of CoST samples. First, from the left, grab the gesture. Next, tickle gesture. Third from the left, scratch and, finally, squeeze. Values of pressure are standardized by mean subtraction and standard deviation division. 97
4.14 (A) Pareto frontier for our implementation of Sparse and CIFAR 10 Binary. Notice the different possibilities to choose from regarding RAM, size, and performance. Each point corresponds to a specific configuration of the search space. (B) Full search procedure for CIFAR-10 Binary with the modified SpArSe. Color degradation shows the progress of the search. 99

4.15 Illustration of the capacitive foil. Each of the dots represents an electrode, emitting each one a distinctive measurement of its row/column on the surface. The foil is pasted underneath the plastic cover and connected to the microcontroller through the lower-left pins. 103

4.16 Examples of numbers drawn at the sensitive surface. From left to right: a one, a five, a six, and a three. The black points correspond to the electrode pairs through which the finger has passed. In an image configuration they correspond to 1s while the background is set to 0. 103

4.17 Best architectures resulting from the optimization procedures for both the CNN and FC autoencoders. The elements in the image represent the feature maps. In the convolution case, the size of the feature map is indicated by the vertical and depth numbers, while the horizontal number indicates the number of feature maps. In the FC case, the number indicates the number of neurons in each step, and the images are flattened at the beginning and reshaped at the end. 106

4.18 Optimization procedure for CAE. The y-axis represents the Model size and the x-axis the error (per pixel classification). Both axes are on a logarithmic scale. The color illustrates the evolution procedure as each of the steps of the Bayesian optimization procedure. 107

4.19 Cluster visualization of the different gesture numbers through t-SNE. 108

5.1 **Mobile ITM from single LDR images using mixed quantization network (MQN).** Employment of mixed quantization methods and efficient blocks in MQN help reduce the computational complexity of HDR image reconstruction (shown at the bottom row) from single LDR images (shown at the top row), and enable its deployment to mobile platforms, achieving a latency of ≈21ms with a Samsung Note 20 Exynos 990 MPSocC. . . 113

5.2 Illustration of the base backbone and high precision head that comprise the MQN. IRLB is used for fast inference and gated attention mechanisms [19] for improvement of feature representation learning accuracy. The dotted line indicates the separation between the fully quantized architecture and the dynamically quantized head. Input is added to the output of the head to produce the overall output. 115
5.3 Depiction of (a) Channel Attention (CA) block, (b) Spatial Attention (SA) and Channel Spatial Attention (SCA) block. *Conv* refers to a standard convolution, a soft sigmoid form denotes the sigmoid activation, the rectilinear symbol denotes ReLU activation and the $\otimes$ denotes element-wise product. 118
5.4 Illustration of feature maps learned at the Att. 4 layer of the MQN depicted in Figure 5.2. The rows show in order results obtained without using an attention mechanism, followed by using SA, CSA, and CA. The first column shows the predicted HDR image $\hat{H}$ and the rest shows the feature maps learned at different channels of the Att. 4 layer. 123
5.5 Analyses of the quantized $Q(\mathbf{f})$ and floating point $F(\mathbf{f})$ features $\mathbf{f}$ learned by the MQN at the ConvBnReLU3 layer at Figure 5.2 using a) three sample input images with b) predictions $\hat{H}$. Visualization of (c) the first channel f_1 and (d) the second channel f_2 of $\mathbf{f}$, (e) the difference map $\Delta f_2 = \|Q(f_2) - F(f_2)\|_1$. We show the probability mass function (PMF) of $Q(f_2)$ and $F(f_2)$ in (f). . 124
5.6 Visual analyses of the effect of training models with and without using $\mathcal{L}_{FR}$ on predictions $\hat{H}$. It helps models gain structural coherency and improve color details. 126
5.7 Training (left) and validation (right) loss plots for each loss combination established in Table 5.3. 127
5.8 Visual comparison of results obtained using, from left to right; input LDR images, HDRCNN [20], ExpandNet [21], SingleHDR [22] and our proposed MQN. All images are produced with Balanced TMO from the suite Photomatix [23], similarly to [22]. . . 128
5.9 Distribution of HDRVDP Q score values for three test datasets and 5 best competing methods in each case. 130

5.10 Latency and accuracy trade-off comparison with the HDR Eye dataset between competing methods and our proposed solution. Accuracy is measured by the HDR-VDP Q value score. Latency is computed both in the mobile (CPU) and GPU platforms. 131
5.11 Three examples of extreme cases of overexposure in HDR Real dataset. Order from top to bottom is: input, HDRCNN [20], ExpandNet [21], SingleHDR [22], our method and ground truth. 132

6.1 Details of the synthetic dataset for bronchoscopy tracking and calibration. 6.1a, an example of synthetic frames from a trajectory from patient P18's lower left lobe. 6.1b, positions visited by the trajectories corresponding to two different patients: P18 and P20. Both cases only show points pertaining to the lower right and left lobe trajectories. 137
6.2 Architectures for bronchoscopy calibration and tracking. (a) is the baseline network without temporal information management, as in [24]. (b, c, d) include different mechanisms to manage temporal information across predictions. 140
6.3 Effect of data increase in an outer-patient (red) and in an intra-patient setting (green). Average Position Error is represented by a dashed line and Average Cosinus Error is represented by a solid line. The X-axis represents the number of sequences per lobe and patient available. 147
6.4 Results for the pruning experiment. Size represents the relative size of the pruned networks with regard to the original one. Note the improvement in rotation prediction of the networks pruned to a 90%, 70%, 50%, and 30%. 148
6.5 Position trajectory comparison with the two state-of-the-art methods, OffsetNet (green), our method (orange), and the ground truth (blue). 149

List of Tables

3.1 Differences between the CMSIS-NN Legacy API quantization and the new TF Lite compatible scheme. Extracted from the CMSIS-NN Github page. 58
3.2 An overview of the popular deep learning frameworks for mobile terminals. Inspired in [25]. * Previously contained inside Caffe2. We have covered only the major frameworks. A more broad list is included in [26]. 62

4.1 Range of the hyperparameters for optimization. 89
4.2 Initial network results. 89
4.3 Best network designs found by the optimization process. 91
4.4 Results for the original version of SpArSe and our implementation. Accuracy is in % and Size corresponds to the model weight in KB taking into account only weights and not code. The maximum RAM is also in KB and corresponds, in our case, to the computation specified in Section III-A. Our results are run in a single RTX 2070 Ti, while the original paper uses four RTX 2080. GPUD corresponds to Graphic Processing Unit days. 99
4.5 Results for the CoST dataset with hold-one-subject-out testing, as in [27] and [28]. * indicates that it has been estimated from the network details in the original paper. ** In this case, authors did not use leave-one-subject-out testing and hence only MS and WM are directly comparable. 101
4.6 Results for the CoST dataset with hold-one-subject-out testing for CNN and CNN+RNN search spaces. The first three rows are for results with CNN and the last three with CNN+RNN. Each row corresponds to the test result of different configuration points in the Pareto front. 101
4.7 Results for the optimization procedure with SpArSeMod. MF corresponds to maximum feature map size, MS to model size, L to latency, and V-M to the V-Measure. 107

5.1 Analyses of accuracy and latency measurements (on the SN20E990) for different attention mechanisms. None indicates that no attention mechanism is used and B indicates backbone. Blue and red indicate the best and the second-best accuracy, respectively. 123
5.2 Results obtained using different quantization schemes. Latency (L.) is measured on the deployment platform (SN20E990). . . . 123
5.3 Analyses of accuracy for different loss combinations.. Blue and red indicate the best and the second-best accuracy, respectively. 125
5.4 Comparison with other state-of-the-art single image HDR reconstruction methods. Performance metrics and latency values were reproduced with the same evaluation criteria and original codes. Blue and red indicate the best and second-best accuracy. P. indicates the number of parameters, L. M. indicates latency for mobile CPU, M. RAM the maximum RAM memory consumed by the model, and O. the number of operations in multiply-accumulate units. Performance values are given in HDRVDP-Q score. *FHDR[29] uses recurrence: the present value is computed taking into account two iterations. 127
5.5 Comparison with state-of-the-art single image HDR reconstruction methods for PSNR and SSIM performance metrics. Values reproduced with the same evaluation criteria and original code. Blue indicates the best and red indicates the second best accuracy. 131

6.1 Comparison among bronchoscopic tracking studies with regards to data and evaluation characteristics. Notably, none of the methods share a dataset (currently there is no publicly available dataset for this task) or publish their code. Moreover, metrics, although aiming to measure the same quantities, are different or lacking in some cases. Metrics shown are umbrella terms for measuring the position error (PE), angle error (AE), and the number of correctly tracked frames. Tracking type [30] refers to the type of information provided by the tracking: global type positions the bronchoscope in macro terms, e.g. 3rd bifurcation, while local, give information with regards to position and angle of the bronchoscope. 136
6.2 Results for the different loss combinations evaluated with the different metrics. Values show mean and standard deviation. . . 146
6.3 Ablation results for the different architectures proposed in Section 6.2.3 with regards to performance metrics L_2 and CE. . . . 147

6.4 Ablation results for the different architectures proposed in Section 6.2.3. Latency is computed with an NVIDIA Jetson AGX Xavier embedded GPU board, using *trtexec* benchmark, and TensorRT 8.4 as conversion and deployment environment. Sample input has dimensions (1, 10, 3, 256, 256), being (batch size, sequence length, channels, height, width). 148

6.5 Comparison with the state-of-the-art method for bronchoscopy tracking and calibration in terms of position error (L_2), angle error, and the number of parameters. 149

6.6 Comparison with the state-of-the-art method for bronchoscopy tracking and calibration in terms of latency in two GPU boards: GTX 1080 Ti and an embedded GPU board, NVIDIA Jetson Xavier. Results for input with dimensions (1, 10, 3, 256, 256), where each dimension corresponds to (batch size, length of sequence, channels, height, width). TensorRT 8.4 is used as a deployment framework and *trtexec* as a benchmark utility. The latency benchmark is performed with unpruned networks since sparsity is only available for Ampere-based architecture boards. 149

List of Abbreviations

AI	**A**rtificial **I**ntelligence
HW	**H**ard**w**are
NN	**N**eural **N**etworks
DL	**D**eep **L**earning
HMM	**H**idden **M**arkov **M**odels
LSTM	**L**ong **S**hort-**T**erm **M**emory
CNN	**C**onvolutional **N**eural **N**etwork
FC	**F**ully **C**onnected
GAN	**G**enerative **A**dversarial **N**etwork
NAS	**N**eural **A**rchitecture **S**earch
SOTA	**S**tate **o**f **t**he **A**rt
TF	**T**ensor**F**low
MCU	**M**icro**c**ontroller **U**nit
FPGA	**F**ield-**P**rogrammable **G**ate **A**rray
GPU	**G**raphic **P**rocessing **U**nit
GP-GPU	**G**eneral **P**urpose - **G**raphics **P**rocessing **U**nit
CPU	**C**entral **P**rocessing **U**nit
NPU	**N**eural **P**rocessing **U**nit
TPU	**T**ensor **P**rocessing **U**nit
2D	Two **D**imensional
3D	Three **D**imensional
NLP	**N**atural **L**anguage **P**rocessing
RL	**R**einforcement **L**earning
BN	**B**atch **N**ormalization
HMI	**H**uman **Machine** **I**nteraction
HDR	**H**igh **D**ynamic **R**ange
TM	**T**one **M**apping
ITM	**I**nverse **T**one **Mapping**
LDR	**L**ow **D**ynamic **Range**
IRLB	**I**nverted **R**esidual **L**inear **B**ottlenecks

A Clara Pons Duran,
Felicidad Revuelto Renta,

y a todos los que estuvieron y ya no están:
Fernando Carazo Gil
Andrés Borrego Gallardo - Juana Pastor Cobo
Consuelo Duro Revuelto

Chapter 1

Introduction

Neural network (NN) development, usage, and commercialization have exploded in the last decade, even though they were created in the 40-50s and further developed later. NNs have succeeded thanks to three main factors [31]: data availability, training of algorithms, and suitable training hardware. This conjunction has allowed for advancements and state-of-the-art results in areas like computer vision and natural language processing, albeit at the cost of higher computational loads and requirements [32]. Models with abundant data available and powerful training hardware have grown in complexity and size, restricting their application to certain conditions, and also increasing their training cost and energy consumption [7, 33, 34].

Alleviating such conditions has multiple benefits: from reducing CO_2 emissions [35] to a more accessible and fairer AI development [7]. Research efforts in such direction have focused on tasks such as how to implement data-efficient training or the development of hardware accelerators, among many lines of research. Betwixt them, an important research objective has been the reduction of model complexity while maintaining performance, that is, making NN models more efficient. In this latter sense, methods such as quantization, pruning, or Neural Architecture Search (NAS) are examples of techniques that allow reaching smaller efficient models, enabling their deployment in platforms with restricted computational resources. There has been a huge development of software (complementary to the existing NN development frameworks) devoted to porting and adapting NN models to those application-specific hardware platforms, such as mobile phones, or ultra-low power micro-controllers. The conjunction of optimization methods and standardized deployment frameworks has allowed both to port NN models to many applications that were previously restricted due to the limitation of the computational resources. However, with regards to optimization methods and deployment frameworks, there are still limitations and points of improvement, like the further reduction of hardware requirements of some models, the extension of techniques to new operations and applications, or an increased standardization and support for deployment frameworks.

All in all, from optimization procedures to conversion and deployment frameworks, the procedure of developing efficient NN-based models and deploying them to constrained hardware has certainly improved, albeit with still some limitations. In such a sense, this thesis is framed by such improvements and limitations: first, with the development and improvement of NN optimization techniques, and second, with the use and development of software for porting the optimized models. All with a special focus on industrial and practical cases which are the main drivers of the developments.

This first chapter provides context and a guided introduction to the main focus of the thesis: adaptation and/or development of NN models for deployment into resource-constrained platforms. First, in Section 1.1, a brief historical background is provided to be able to put the work in context, as well as to understand the historical reason for such a topic. Second, in Section 1.2, the research purpose of the topic is introduced, stating its importance and how has been dealt with by the research community. Section 1.3, describes the conceptual scope for the work and how the research field is organized. In section 1.4, the different resource-constrained platforms considered are introduced as well as how their development and use have been linked to the development of optimized NNs. Finally, in Section 1.5, the thesis topic is defined, further specifying its amplitude, purpose, and appropriate nuances.

1.1 Brief historic background of neural networks in AI

The objective of infusing sapience in inanimate objects has a long cultural tradition in humankind [36, 37] and modern inventors and thinkers often have followed such purpose in their undertakings and thoughts [38, 39, 40]. In such pursuit, a source of inspiration has been the human brain itself: from the first experimental studies about the brain organization and functioning, [41, 42, 43], to its analysis with modern techniques like optogenetics [44, 45], understanding how the brain enables thought and understanding has been one of the major inspirations for AI. And such is the starting point for neural network models and deep learning.

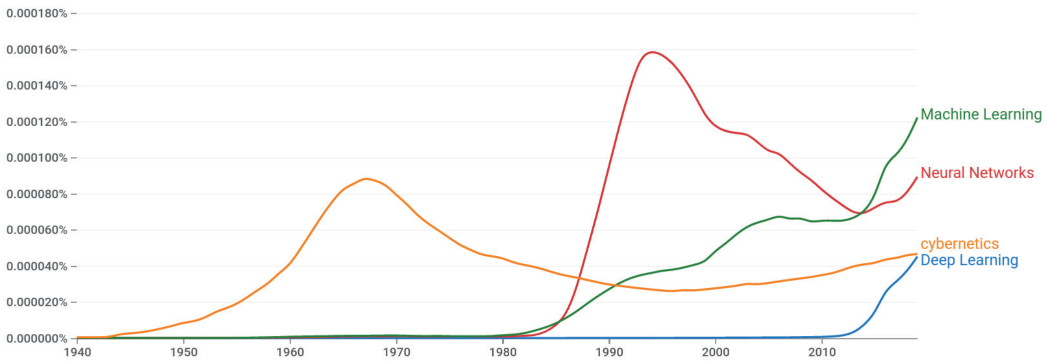


FIGURE 1.1: NN word trends from the 1940s until the end of 2010s. We can see, more or less distinctively, the different phases: first, cybernetics, second, connections, and finally deep learning era. Note: cybernetics has separated from NNs meaning.

First phase, Cybernetics. The first recognized AI studies [31] were inspired by the physiological and functional relationship among neurons in the brain, as well as how to process logical propositions through them. McCulloh & Pitts [46] established the first artificial "neurons" network model, a weighted linear model, $f(\mathbf{w}, \mathbf{x})$, where $\mathbf{w}$ represent the connections and $\mathbf{x}$ the inputs or stimulus. Such a model was shown to be able to represent computable functions and construct boolean operatives through network connections, but connections and their strength were manually set. Later, Hebb [47] developed a method for the update and modification of such connections and their values, setting the start for learning algorithms.

Immersed with other AI research streams, like logic and theorem solvers [48, 49] or programming languages [50], artificial neural network theory expanded. Learning methods were improved with works such as Adaptive Linear Elements (ADALINE) [51, 52], or the Perceptron [53, 54, 55, 56], which was the first model showing weight learning conditioned to input categorization [57]. Works such as [58] showed that an ensemble of elements connected between them could represent an individual concept.

However, albeit an initial enthusiasm, the incapacity for solving scaling difficulties [59] and the limitations of first NNs [60], such as only being able to represent linear systems, produced backslashes in funding and research that hindered advances until the 80s.

Second phase, Connectionism. Although AI was immersed in a so-called *winter* due to the failure of expert systems to deal with uncertainty and their incapacity to learn from experience, during the late 80s and early 90s neural network models experienced new advances. Such improvements were condensed in the research advance of parallel distributed processing [61, 62], which fostered the concept of connectionism, where a large number of computational units could develop intelligent reasoning when connected and intertwined. Altogether with such development, other advances helped promote the field. The review and further research of the back-propagation algorithm [63, 64, 65, 66], which had already been developed in the 60s [67, 68], allowed training multi-layer NNs with internal representations, finally establishing them, mathematically, as universal function approximators [69]. Additionally, NN hardware mapping and usage experienced a renewed boost [70, 71, 72, 73, 74] up-heaving the importance of such relationship.

Regarding speech recognition, and after important advances in language modeling with Hidden Markov Models (HMM) in the 80s, notable developments were made to the field using NNs, where the identification of mathematical obstacles [75, 76] and the development of the Long Short-Term Memory Network (LSTM) [77] stand out.

During the late 90s and early 2000s, there was notable progress in neural network research. Convolutional NNs (CNN) were developed based on the neocognitron [78], a network based on the mammalian visual system. From then on, neural network models continued to obtain increased performance, and improvements were made in their modeling [79, 80]. However, and once again, unrealistic claims, the incapacity to translate them to results, and the surge of new models, such as kernel machines or ensemble methods, drew funding back from neural network research until the mid-2000s.

Third phase, Deep Learning. At the time, NNs were thought to be especially difficult to train, probably due to the high computational cost of training algorithms for the available hardware and the subsequent lack of experimentation. Nevertheless, interest was drawn again upon NNs with the work [81] that showed an efficient way to train a deep belief network using a greedy-layer strategy. Advances followed with other architectures [82], and, especially, with a focus on the importance of depth [83, 84], which popularized the term *deep learning*.

Interest in deep learning finally detonated with the work [85], which used deep convolutional NNs and GPU training to win the ImageNet Large Scale Visual Recognition Challenge by a notable margin and a drop in the error rate of more than 10% compared to previous editions. Since then, deep NNs have begun to be used extensively in other tasks and fields, such as speech recognition [86, 87, 88], natural language processing [89, 90, 91], recommender systems [92, 93], or reinforcement learning [94, 95], establishing new state-of-the-art solutions and applications.

Additionally, research has also flourished with new advances. Techniques for improved training and generalization, such as dropout [96, 97, 98], ReLU and variants [99, 97, 100, 101, 102], batch normalization [103, 104], or weight decay [105, 106, 107], have been developed. New architectures and models have been engineered improving state-of-the-art results: generative adversarial networks (GANs) [108, 109, 110], transformers [111, 112, 113], new RNNs and CNNs [114, 115, 116, 117], graph NNs [118, 119], as well as NAS, to find even better performing architectures by exploring the configuration space. Efforts have also been made towards improving the explainability and interpretability of NNs and to understand the reason for their learning capacity [120, 121]. Libraries and frameworks [122, 123, 124] have been standardized and open-sourced, pushing the possibility to compare and build upon research. Representation learning, unsupervised training, and other learning strategies, such as one-shot learning, have driven the boundary of knowledge with regard to how information is processed inside NNs and how much is needed to create valuable representations.

All in all, we have briefly visited the history of NNs and their trends (in Figure 1.1 the word trends related to NNs can be visualized). As seen, there have been and continue to be great developments related to NNs. Improvements that have ground on specific developments in the first decade of the XX century: data abundance, hardware improvement and generalization, and the availability of training algorithms. In the next section, will question the need to add efficiency as the main driver for NN development and see how academics have dealt with it.

1.2 Why efficient neural networks?

With the advent of the World Wide Web and the spread of personal and portable devices, the digitization of society has increased and the quantity of data generated has largely augmented, facilitating the creation of bigger datasets. The datasets during the first phase were small (tens to hundreds of samples), with enough data to prove that NNs could learn certain functions, while in the

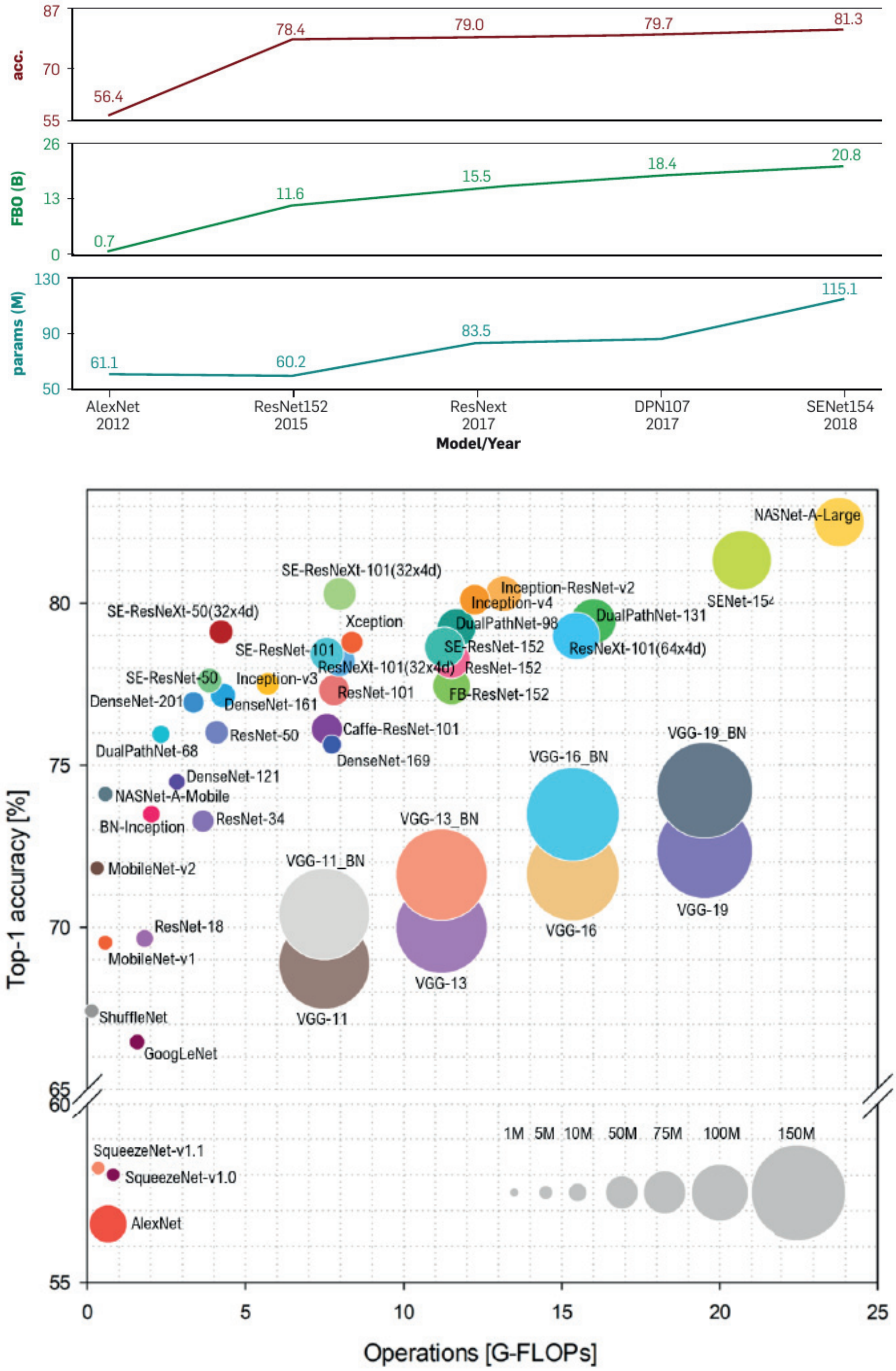


FIGURE 1.2: (Top) Illustration of increasing complexity and size of models pursuing a higher accuracy (acc.) over the years. FPO stands for floating point operations. Image taken from [7] with permission from the authors. (Bottom) Illustration of the size and operational complexity of different networks versus the attained accuracy. Image taken from [8] with permission from the authors.

second phase they had already augmented in size (thousands) and complexity, such as in the case of MNIST [79]. Finally, in the third phase, datasets increased in complexity, as with CIFAR-10 [125], and later, from 2010 and onwards, in size, reached sample numbers of hundreds of thousands and tens of millions (ImageNet [126, 127], Youtube-8M [128], WMT [129], and many others).

In parallel, computing power has increased, is cheapened, and is generalized. Early NNs were developed to be implemented in the CPU, which hindered training them. Even if a careful implementation can yield appropriate inference results in a CPU [130], current models are insufficient for training. On the contrary, GPUs have supposed a breakthrough for training NNs. Traditionally developed for graphics rendering, their inherent parallelism capability and higher memory bandwidth than CPUs, make them more suitable for NN training, where there are large numbers of parameters, gradients, and activations that can be updated independently if they are in the same layer.

Original GPUs were specialized hardware only devoted to the graphics domain. However, they increasingly added more subroutines for other tasks, thus becoming more flexible, and soon they were adapted for scientific and NN computation [131, 132] showing improvements over CPUs. After the creation and expansion of GP-GPUs and associated languages, like NVIDIA CUDA, its usage for training deep learning networks expanded and soon was adopted by many researchers [133, 134]. Finally, after the creation of deep learning frameworks (Tensorflow [135], Pytorch [123], MNext [124], and others), which abstracted such hardware capabilities, GP-GPU usage for NN training and inference exploded, boosting research and advances in deep learning.

This situation, altogether with the existence of suitable algorithms for training NNs and data availability, has led to an increase in model size, complexity, and accuracy, as illustrated by Figure 1.2, generating the current success enjoyed by deep learning models. Regarding size, the first models had dozens of neurons, mainly limited by hardware capabilities, but since the second phase, models have more or less doubled their size every 2.4 years [32], attaining the size of billions of parameters [136] and even hundreds of billions of parameters [137]. An increase has also occurred at the complexity level, where the connections have grown from tens [51] to tens of thousands per neuron [138], and new and more specialized operations types, such as attention [112], have specialized information processing. Altogether both increases have brought new accuracy SOTA in many tasks such as semantic segmentation, object detection, machine translation, text generation, and others, pushing at the same time the requirement for more hardware resources.

As seen, improvements in accuracy have been accompanied by an increase in computational costs and data required. While models grew in complexity and size, the time spent training increased substantially, and so the hardware usage, thus augmenting the energy consumption and carbon-related emissions [34]. Processing power and memory requirements also increased, letting to platform separation for training (in the cloud) and inference (also in constrained devices or edge environments). Moreover and until now, hardware development has been able to match the requirements of new and costly models, however, this tendency might not hold in the near future [139, 140], impeding further increase in model size and complexity in a feasible and useful manner. As an added problem, the increase in complexity has been accompanied by the need for increasing annotated data [32] to avoid overfitting, resorting then to long processes of dataset production with all the costs associated and the bias that labeling induces in the learning process. Additionally and until recently, little attention has been given to methods and procedures to reduce training times or training adaptation to new instances [141], hardening the problem of transfer learning.

In summary, there are several reasons for promoting efficient NNs based on two main grounds: operability and data. With regards to data, the goal is to promote NN efficiency to reduce the dependency on the quantity of data, thus avoiding massive labeling, but also in what is learned from data, promoting easier task shift. Concerning the operability case, it is mandatory to reduce complexity and hardware requirements in order to reduce the energy consumption both in training and inference, but also to help port NN utilization to tasks and situations that are resource constrained [142, 143]. The present work is focused on the latter case: how to make NN more efficient so as to port them to applications with resource-constrained devices.

1.3 Neural Network Optimization

Given such context, there have been continued research efforts to promote efficiency in NNs. In the present work, we fundamentally based our developments on operability, specifically focusing on the model and how it can be

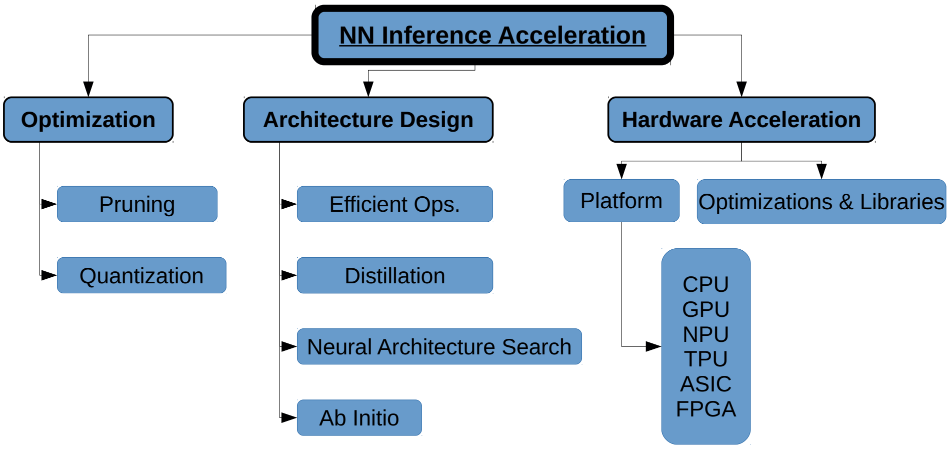


FIGURE 1.3: Proposal of organization of methods for accelerating or enabling NN inference in resource-constrained devices. The three most important branches are optimization, architecture design, and hardware acceleration.

adapted and made more efficient so it can be used in constrained environments. The focus has not been on data issues or efficiency through hardware-specific implementations [144].

There have been multiple and varied research efforts that have tried to make NN models more efficient so they can be deployed, developed for, and used in constrained hardware environments, that is, platforms that previously did not have the required computational resources for such models to be appropriately run on them. Among such different methods, six stand out as the main procedures for model efficiency improvement: quantization, pruning, distillation, neural architecture search, efficient operations, and *ab initio* model development. Below, those methods are briefly introduced and organized according to their functionalities.

Quantization is a general concept present in multiple scientific fields, such as quantum physics [145] or signal analysis [146], and implies the discretization of a continuous space into a discontinuous one in which elements are isolated from one another. In the NNs case, quantization discretizes the numerical space available for weights and activations by using a fixed point representation and often also by reducing the bit depth. It was one of the first methods developed [147, 148, 149], but has gained strength and robustness in development in the past decade [130, 150, 10]. Its main benefits are size reduction and inference acceleration, although it might also provide minimal accuracy improvements due to regularization. The reduction of the bit depth provides a reduction of model and feature map size, thus reducing the memory (static and dynamic) required to run it. By using fixed-point representations, it provides reduced latency when used altogether with hardware suited for such formats, such as CPUs or NPUs. Although it is commonly used for inference, it has also been applied to training [151, 152] and it recently regained interest in reducing resource consumption and accelerating the training procedure [153, 154].

Pruning intends the reduction the complexity of the network, in terms of size or operations, by suppressing elements of it and at the same time minimizing the impact on accuracy. Initially developed in the late ’80s [155], and early ’90s [156, 157], new research has emerged in the last ten years [158, 159], transitioning from its application in initial small networks to the bigger modern networks [1]. The principal point, however, is still the decision measure, that is, the evaluation procedure that selects which weights or activations should be deleted. With regards to such criteria, three different categories can be determined [160]: magnitude-based [155, 158, 161, 162], which select weights based on their local saliency, similarity-based [163, 164], which identifies redundant values, and sensitivity-based [165, 166], focusing on the effect on the loss. More categorizations can be posed [12], but with regards to hardware deployment, there is a distinction that affects enormously their latency reduction benefits and not only size: if they are applied to individuals (unstructured) or groups (structured) [167]. While in the latter case latency reduction is an immediate benefit, in the former the deployment framework must allow for sparse computations [168, 169, 170].

Distillation focuses on transferring the knowledge of a bigger model or ensemble of models, also known as *Teacher*, to a smaller model, known as *Student*. Initially developed by [171], and further matured by [172], it has been further extended to different architectures [173, 174] and tasks [175, 176]. Some research has been devoted to its principles [177, 178, 179], however, the principal ideas remain the same. The *student* is trained with both the class probability distribution (*soft targets*) produced by the *teacher*, softened through a temperature parameter, and also the real class labels (*hard targets*). The temperature parameter can soften or harden the labels and it controls the distribution of information transferred to the *student*, thereby enabling information

---

[1] Network in [157] had 2600 parameters, while modern networks can have from millions to billions, thus changing the feasibility of certain objectives and algorithms

on secondary classes or values.

Efficient operations aim to reduce the computational burden in terms of memory used and latency of operations while maintaining the informative capacity, enabling their deployment and usage in more constrained platforms such as mobiles [16, 180]. Such reduction is usually achieved through the combination of simpler operations and the way they are combined, a procedure known as *operation factorization*. Outstanding examples are the GRU cell [181] compared to LSTM cell [77], or depthwise separable convolutions [15, 16] and ShuffleNet units [180] compared to standard convolutions. Note that the design of such operations has been mostly manual. However, recently, NAS methods have been used to automatically define efficient blocks taking into account different constraints such as the number of FLOPS or the memory used [182, 183]. Finally, another possibility, although less used, is to change the mechanisms of operations, as in circular projections [184] or deformable convolutions [185, 186].

Neural Architecture Search builds NNs in an automated manner given a search space, a search algorithm, and an objective [187]. Although initially developed to improve performance, it has also been oriented to multiobjective search [188]: by taking into account different objectives, like memory usage and accuracy, efficient and well-performing NNs can be built. Hence, Pareto optimal NNs can be developed and deployed in differently resourced platforms, such as micro-controllers or mobile phones [35, 189]. The three most important components of NAS algorithms are the search space, the search algorithm, and the objective: the search space defines the available operations and parameters for the network, and the search algorithm samples new network proposals and values them according to the objective. Although it can deliver well-performing results, even beyond manually building, it is costly in terms of computation time [34, 190] and extremely dependent on the search space.

Ab initio methods focus on building NNs which are efficient by design and from inception. This means they do not apply a method *a posteriori* to make the network more efficient. By definition, there can be a matching zone with efficient operations, since those are also efficient from the beginning. However, the difference is in the degree of efficiency or latency achieved and how models are built: while efficient operations are usually modular, *ab initio* models are usually built as a whole. Depending on the task, such methods are prone to be used or not, image enhancement is a task where they often are employed. Outstanding examples are look-up table (LUT) usage [191, 192], bilateral space modeling [193, 194] and Laplacian Pyramids [195, 196], among others [197].

After taking a glimpse at each optimization method[2], it can be observed that, although every method has its particular features, they can be divided into two sets depending on whether the optimization is done given a predefined network or it is defined from the beginning. Quantization, pruning, efficient methods, and in a certain way distillation, are limited to the original network they are applied to, and their procedures are thus bounded to a certain degree. Meanwhile, NAS, to a certain extent, and *ab initio* development, can provide a more flexible optimization given the intervention of the developer. Such distinction is the most important when proceeding with the deployment because it will establish the limits of our final model in terms of performance, latency, and accuracy. An illustration of the organization of such methods, altogether with hardware acceleration, is presented in Figure 1.3.

All in all, these methods establish the base mechanisms to make a model more efficient or to develop it more efficiently. Each method with its benefits and drawbacks can be used to reduce the resource requirements of the network and thus more easily deploy them in a resource-constrained platform. Although it is not the main focus of this thesis, the deployment platform, as well as related factors, such as the availability of frameworks for deployment or adaptation, play an important role in the procedure of application development and the usefulness of the mentioned optimization methods.

1.4 NN frameworks for resource-constrained hardware

The success of NNs promoted the creation of open-source, well-maintained, and developed deep learning frameworks, such as Caffe [198], Chainer [199], MXNet [124], Tensorflow [135], PyTorch [123], and others. Altogether with the previously introduced optimization methods, which focus on reducing the computing requirements of NN models, the former success has also led to the development of libraries and frameworks for porting them to resource-constrained or specialized hardware, thus broadening the range of applications and environments where NNs can be deployed.

---

[2] *Conditional computation* is not included since, although it could optimize a system through compositionality, it is not per se an efficiency development method.

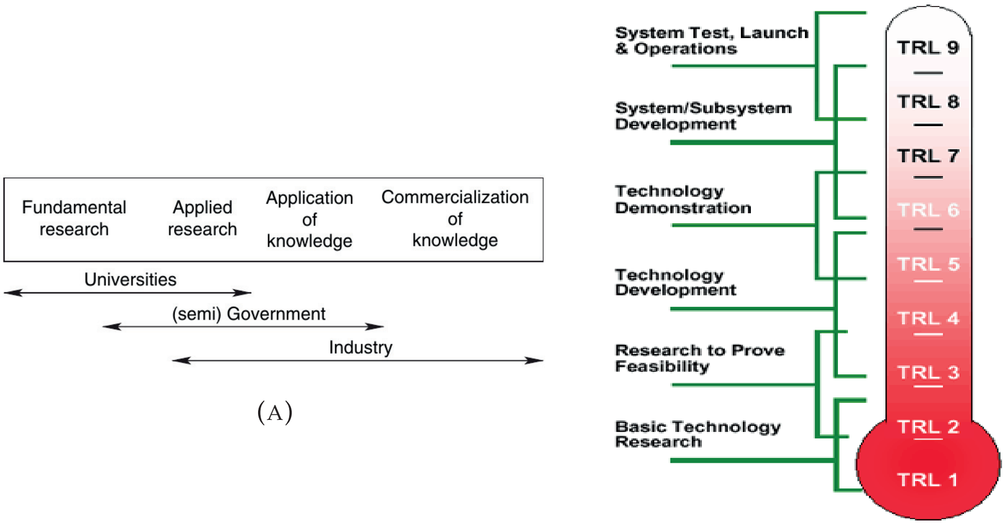
In the case of mobile phones, the first official library was Tensorflow Mobile [200], which appeared in 2015 and had only support for CPU deployment. It was substituted by Tensorflow Lite (TFLite) [201] in 2017, which incorporated kernels for inference in hardware accelerators, and had support for NNAPI [202], thus allowing for direct hardware usage through the Android system. With the development of TFLite, and the adoption of optimization procedures such as quantization, TF closed the DL pipeline: training, optimizing, and deploying could be done using the same framework. Dismissing the first attempts of hardware vendors, such as Qualcomm [203] or ARM[3], to provide access to hardware accelerators (like GPU, DSP, NPU), the NNAPI has allowed for a more standardized connection to such systems directly from the deployment framework despite its drawbacks [204]. This access has even become easier with the introduction of TFLite Delegates [205], which helps vendors and developers standardize hardware accelerator access to a common high-level API, as in the case of [206], which provides a universal GPU access. Despite the prominence of TFLite, other frameworks are also adding support for model deployment on smartphones, like PyTorch with Pytorch Mobile [207], even though with greater restrictions than TFLite concerning operations and hardware access.

Another hardware platform that has experienced a notable development regarding hardware and software connection is microcontrollers. First attempts were purely driven by hardware vendors, such as NXP [208] or Texas Instruments [209], and provided deployment environments for their platforms and the connection to the training frameworks. However, such connections were highly limited to high-end platforms and had severe constraints concerning available operations. A first important step was made by processor core IP vendor ARM, which provided a NN deployment suite, CMSIS-NN [210], for their whole range of Cortex-M and A chips. Afterward, more frameworks appeared for both converting and deploying NNs to microcontrollers, as in the case of μTensor [211], NNOM [212], or Apache TVM [213], which lately has extended its range to more platforms and procedures. Finally, as in the case of mobile phones, training frameworks have also begun to include support for deployment in MCUs, like TFLite Micro [214], albeit with limited operability.

Albeit with differences, the same process has been repeated for other platforms, such as FPGAs with OpenVINO framework [215], or embedded boards,

---

[3] https://community.arm.com/developer/tools-software/graphics/b/blog/posts/smile-to-the-camera-it-s-opencl



(A)

(B)

FIGURE 1.4: Two illustrations of the diverse components and steps of scientific development: from research to commercialization. Picture (A) broadly illustrates the 4 main areas of scientific development and the organizations involved in them. Taken from [9] with permission of the authors. The present thesis is placed between applied research and the application of knowledge. Picture (B) states the different Technology Readiness Levels (TLRs). The current thesis spans from TLR 2, and as far as TRL 7. Picture of public domain from Wikipedia, Technology Readiness Level.

with NVIDIA's TensortRT [216]: from platform-specific and vendor linked framework implementations, to generalist and broad frameworks for deployment, NNs have experienced a growth in the availability and generality of deployment, albeit with still outstanding limitations.

All in all, the development of deployment frameworks and their linkage to training frameworks is still a subject of needed research and effort. Many limitations still exist in those frameworks in terms of support for operations, optimization procedures, and hardware accelerator access. We can consider that the success of application deployment is tightly bonded to two factors: (1) the link between general purpose and platform-oriented development frameworks and (2) the support and suitability of the selected framework chain to the specialized hardware platform instance.

1.5 About this thesis

As shown in previous sections, there is a need to produce more efficient NNs solutions to widespread their application domain, especially in resource-constrained environments, but also to optimize their computing requirements. Such need has been partially covered by the initial development of optimization and development procedures, namely quantization, pruning, distillation, NAS, efficient operations, and *ab initio* development. They allow either reducing the computing requirements of a model or developing a new functionally-equivalent model with lower requirements. However, there are still improvements needed in developing further these methods and making them available for industrial applications.

As previously seen, along with the optimization improvements, there has been a surge in software for deploying NNs to resource-constrained or specialized hardware platforms. Hardware platforms, like mobile phones and micro-controllers, have experienced a surge in deployment frameworks that, albeit still with important limitations, allow them to deploy NNs and develop new applications.

The conjunction of the research of optimization methods with the development of deployment frameworks has allowed for the development of models in industrial settings with limitations in resources (like connection or computing power). Such a fact has enabled not only the creation of new applications but also the improvement in results or conditions of already existing applications.

In such a setting is placed the motivation for this thesis: the advancement of optimization methods, their conjunction with deployment software, and the application of both in industrial settings. In this thesis, we will implement and improve examples of each optimization method, developing models for different industrial application and their deployment in resource-constrained hardware platforms with NN deployment software available. This thesis does not focus on specialized hardware for NN accelerated inference, but rather on how to adapt or develop NNs to run efficiently in hardware not specifically designed for the latter purpose. Then, the **objective** of the present thesis is stated as the following:

To develop, improve, implement, and apply certain cases of the optimization and development methods (quantization, pruning, efficient operations, and NAS development) in different application and industrial environments with the focus on resource-constrained hardware that will execute the NN deployment software for efficient inference (algorithmic and computational).

Therefore, there are 3 main components of the thesis: optimization methods, deployment software for hardware platforms, and selected industrial applications. Concerning the optimization methods, the present thesis covers all of them except distillation with more or less depth, in some cases directly applying them, due to the industrial nature of this thesis, and in others extending and further improving them. In the case of hardware platforms we have focused on three: mobile phones, micro-controllers, and embedded GPU boards; thus limiting also to specific deployment frameworks for such platforms, mainly TFLite for mobile phones, CMSIS-NN for micro-controllers, and TensorRT for GPU boards. Finally, with regard to industrial applications, we have tackled four different problems: gesture recognition with capacitive sensors, inverse tone mapping (ITM), bronchoscopic virtual tracking, and meningitis segmentation.

Specifically, the contributions at the research level have been:

- Analysis of the SoA regarding embedded machine learning in Advanced Driver Assistance Systems [1].
- Implementation, analysis, and research of minimal RNNs for gesture recognition in resource-constrained MCUs [3, 2].
- Implementation, research, and extension of Bayesian optimization based NAS methods with conditional Gaussian Processes for searching minimal NNs [4, 217, 218].
- Development of NN converter between DL training framework (PyTorch) and deployment framework (CMSIS-NN) [219, 220]
- Analysis, research and development of an efficient NN for inverse tone mapping in smartphones [6].
- Research and development of efficient convolutional and recurrent-based NNs for improved and efficient video bronchoscopy tracking.

From a research perspective, this thesis is placed conceptually between applied research and its bridge to product development and the application of knowledge, as illustrated in the central parts of Figure 1.4a. Under the TLR scheme, illustrated in Figure 1.4b, this thesis is placed between TLR 2 and TRL 7: we improve research methods proving their feasibility and also perform technology demonstration. All in all, the intention is to provide a better link between research and application for the rising constrained artificial intelligence domain.

The rest of the thesis is organized as follows. Chapter 2 provides a general theoretical background for the optimization methods applied in the experiments, and Chapter 3 details information about deployment frameworks and related platforms used. The next chapters are devoted each to a different industrial setting and cover the introduction, background, experiments, and contributions to each case. Chapter 4 covers gesture recognition on microcontroller units. Chapter 5, Inverse Tone Mapping in Smartphones, and Chapter 6, two medical applications on embedded GPUs. Finally, Chapter 7 states the main conclusions of the thesis and further research directions.

Chapter 2

Applied Optimization Methods

The purpose of the present section is to provide the necessary technical background and context for framing the proposed developments and research. In the form of theoretical descriptions or as a review of the recent advances on the topics, the intention is to position the reader with the necessary background and fundamental concepts among which the thesis is found.

There are four main sections, each devoted to an optimization method used in the thesis projects: quantization, neural architecture search, pruning, and efficient methods. In each of them, the literature available regarding the topic is reviewed and briefly discussed, outlining the main characteristics and functioning of the core of the topic. Each section has the same structure. First, an introduction to the topic, followed by a brief description of the recent historic evolution. Next, a concise fundamental review where the fundamental concepts of the topic are treated, followed by a discussion on important characteristics. Finally, we finish with the conclusions.

2.1 Quantization

2.1.1 Introduction

Neural networks are computing intensive in terms of operations, mainly multiplication, and size. Optimizing resources to reduce the size, time, and the energy consumption is of outstanding importance in order to accommodate NNs in commercial products which are economically viable. They are usually trained and used with `32-bit float` data types for both weights and activations. Thus, one way to reduce resource consumption is to use reduced bit-depth integer data types and computations [130, 150, 10]. By converting only weights, model size can be reduced. However, if activations and operations are also performed on smaller data types, latency improvements can be achieved altogether with model size reduction, especially when using specialized deployment frameworks and appropriate instruction sets are available.

However, when performing inference in a reduced data type (typically 8-bit integers) with a network that was trained on a bigger bit-depth data type, losses in accuracy can happen. To remediate this solution, several options exist. First, by working with a quantization scheme that more closely matches the original data distributions of each layer, some of the accuracy can be restored (for example, a notable improvement can be obtained between the power of 2-based symmetric quantization and a scale and offset-based scheme). Moreover, if quantization is developed also during training, *quantization-aware training*, these schemes can be closely matched to the evolution and values of the higher bit-depth data type values of the network. Additionally, other schemes and further data type reductions (below 8-bit integers) have been developed, obtaining further model size reductions but needing more complex schemes to avoid the accuracy drop [221, 222]. Moreover, such works have been added in deployment frameworks due to their complexity and also the difficulty of translating operations into hardware instruction sets.

In general, quantization allows to reduce the model size and latency, but usually with a reduction in the accuracy. Its usage is widespread in common development and deployment frameworks for some common data types such as 16-bit floats or 8-bit, enabling its use and its benefits. However, for other data types its direct application is difficult due to the lack of support in common deployment frameworks and hardware instruction sets. In the present section, we are going to review the main types of quantization schemes, how quantization is conducted, and how the accuracy drops associated with it can be avoided to the maximum. All this, with special relation to the developments used in the thesis.

2.1.2 Brief History

In [147] researchers were already studying the effect of quantization in neural networks, even the possibility of reducing such adverse effects by inducing quantization in the backpropagation process.

Since the second wave of neural networks, there has been interest in reducing the hardware and computing requirements of NNs. A way to pursue such an objective was to reduce the arithmetic precision of operations and values and their representation, thus reducing computing time, size, and hardware area. Works such as [151, 152, 223, 224] experimented with reducing the bit size, the representation (either fixed or floating point), and the implementation on specialized hardware of back-propagation and inference with neural networks. It was generally seen that reductions in precision and the use of fixed point representation could be beneficial, especially at inference time. Recently, the topic has regained interest with works such as [225, 153, 130], where it is shown that fixed-point low precision, such as 16-bit or 10-bit, is already sufficient for training deep NNs with minimal degradation by paying attention to the rounding and dynamic range schemes.

During the past years, quantization has gained a good pace both in research, development, and adoption in major development frameworks, where int-8 has been widely implemented for inference. Additionally, extreme (1-4 bit) quantization has also been researched, introducing such heavy quantization even during training time [154].

2.1.3 Fundamental Review

There are different options when applying quantization to a NN with regards to quantization scheme, bit-depth, and target structure. With regards to quantization scheme, stand out symmetric quantization, affine quantization, and affine fake quantization during training [10, 226, 150]. In the case of the bit-depth, we have every possible bit-depth available under 32-bits, however, typical values are 8, and less used 2 and 4. Finally, one can target weights only or operations and weights, being able to apply quantization layer or filter-wise.

Symmetric Quantization Having a 32-bit floating model with minimum and maximum weight values in range (x_{min}, x_{max}), the model can be converted to a quantized range $(-N/2, N/2 - 1)$ with a scale factor, s (where N corresponds to the maximum value achievable with the bit depth, b, for example $N = 256$ when using $b = 8$ bits). The quantization operations are as follows

$$x_q = x_r \cdot s^{-1} \tag{2.1}$$

$$x_Q = clamp(-N/2, N/2 - 1, x_q) \tag{2.2}$$

where x_r is the original floating point number. As seen, in this case, we are doing symmetric quantization around 0. If the floating numbers are unsigned the scaled number can be clamped to $x_Q = clamp(0, N-1, x_q)$. In order to dequantize we only need to apply the quantization scale inversely :

$$x_{r'} = x_q \cdot s \tag{2.3}$$

Specifically the factor s can be chosen as the following when using symmetric power of 2 quantization: $s = 2^{b-1-\log_2(x_m)}$, where x_m is the absolute maximum between x_{max} and x_{min}.

The clamp function is defined as:

$$\begin{aligned} clamp(a,b,x) = a \quad & if \quad x < a & (2.4)\\ x \quad & if \quad a < x < b & (2.5)\\ b \quad & if \quad x > b & (2.6) \end{aligned}$$

Affine Quantization In the previous case, the center of symmetry is 0. However, in most cases, that is inefficient and induces losses because the data might not be centered in it. Thus, the quantization scheme can be improved by adding an offset value z that handles such displacement. Now, after scaling it, the quantized value is displaced by z

$$\begin{aligned} x_q &= x_r \cdot s^{-1} + z & (2.7)\\ x_Q &= clamp(0, N-1, x_q) & (2.8) \end{aligned}$$

where now the scale factor is $s = 2^{b-1-\log_2(x_{max}-x_{min})}$ for power of two quantization. The offset z is then the integer value corresponding to the float value $x_{min} + (x_{max} - x_{min})/2$. Then the dequantization operation becomes

$$x_r = (x_Q - z) \cdot s \tag{2.9}$$

In this case, as noted in [10], the naive convolution leads to a reduced throughput due to the use of larger accumulation buffers (16 or 32-bit). However, this can be partially solved through optimized convolution kernels, as detailed in [227].

Simulated Quantization The usual procedure for quantization is to train in float and then, once the network has been trained, then quantize it by either examining the maximum values for weights or by making forward passes and storing extreme values for weights and activations. The main issue is the drop in the performance metric introduced by quantization schemes as detailed in the present section. This is most commonly due to [150]: 1) different output channels differ in the range of values (sometimes by more than 100x), making it difficult to group them under the same quantization scheme, 2) outlier weights which cannot be included during quantization.

A solution to cover these outliers and differences missed by post-training quantization is to use simulated quantization during training [150, 10]. In this configuration, weights are still stored in float, as well as the activations, and the backpropagation step of gradients takes place as usual. However, during the forward pass, quantization is applied and undone, faking its effect in each of the operations: weights are quantized, as well as the inputs, and activations. Outputs are finally dequantized to recover float values. Specifically:

$$x_{out} = (clamp(0, N-1, x_r \cdot s^{-1} + z) - z) \cdot s \tag{2.10}$$

where the quantization has been applied and undone (in this case in an asymmetric manner). With regards to the maximum and minimum values, in the case of weights, the extreme values are directly collected. In the case of activations, as they depend on the input, a possible strategy is to collect them continuously and apply a moving average to the seen extremal values. It is also possible to add the simulated quantization to the backward pass [10], but then, as the derivative is mostly zero for the quantized step, the quantizer has to be modeled in a different manner, for example, through a *straight through estimator* [154, 228].

Parameters and Granularity We have previously mentioned that, for example, in a power of 2 based quantization, the scale parameter could be chosen as $s = 2^{b-1-\log_2(x_m)}$, where x_m is the absolute maximum between x_{max} and x_{min}. However, there are more ways. For example, TensorRT [216] selects the scale and offset parameters as those that minimize the KL divergence between the float distribution and the quantized one. The selection of the parameters

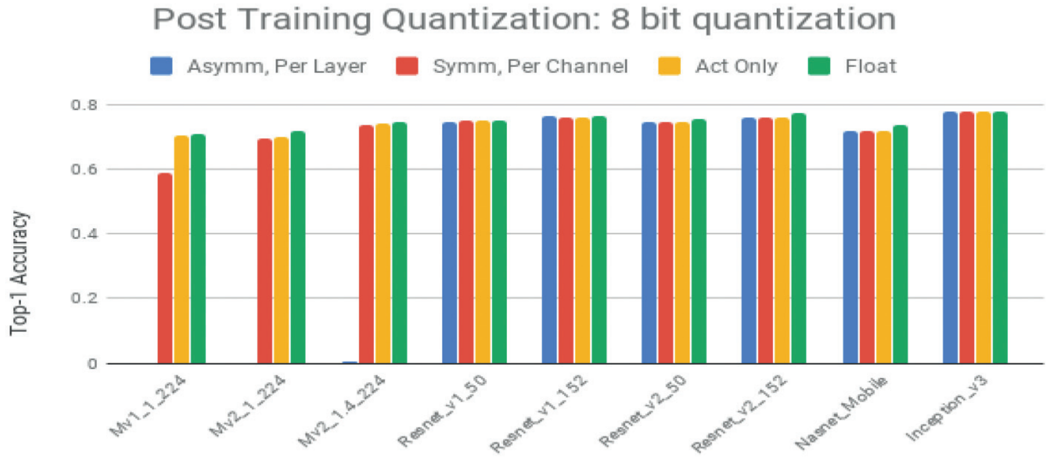


FIGURE 2.1: Example of the different accuracies achieved with post-training quantization depending on the type of quantization for Mobilenet V1 and V2 (mvv1-mv2) and Resnet networks. Figure adopted from [10]

affects notably the accuracy drop between the float model and its quantized version.

Additionally, we have referred to the maximum and minimum values. Those values are collected once the model has been trained (for post-training quantization) by running a set of samples (known as a calibration set) through the model. Hence, these values can be taken for different elements of the network. They can be obtained for a whole layer or, as another example, for a single channel of a layer. This difference affects severely the subsequent performance of the quantized network, as detailed in [10]: the more precise and granular the quantization the better. However, that induces more computational complexity for the quantized kernels. Thus, usually, quantization is limited to channel or layer and not ported at the weight level.

Moreover, the network can be quantized only at the weight level, achieving only model size reduction, or both weights and activations, enabling latency improvements by performing operations in the converted data type. However, in this latter case, accuracy drops take place. In Figure 2.1 accuracy is plotted for different quantization configurations for MobileNet-V1, V-2, and ResNet networks.

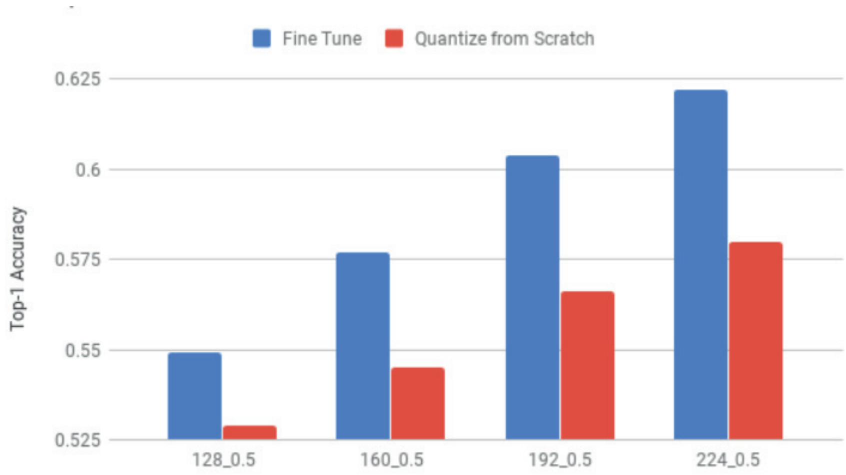


FIGURE 2.2: Difference in accuracy between quantized from scratch models or models quantized from a floating point trained model. Figure adopted from [10].

2.1.4 Discussion

Low precision in DNNs has been explored widely, with important works such as [153], which explores the accuracy results on the ImageNet dataset at different precisions (32-bit float, 16-bit float, 20-bit fixed point, and a fixed point-mixed precision scheme, 10-bit for weights and 12-bit for activations). Training is also performed with lower precision arithmetic. Going back as far as 1991, for example, in [151], we can see that training with a fixed point for reducing the drop in accuracy was already possible. In 2011, the latency reduction with 8-bit integer fixed point arithmetic and post-training quantization was of 2x without accuracy loss. All these works showcase that quantization was already available and ready before the current deep learning frenzy, then the obvious question is, why some improvements have translated into industrial applications and others not? One important reason is the suitability of the data format for the inference hardware. For example, 16-bit floating type fits perfectly GPUs and 8-bit integers generally fit well CPUs in MCUs, however, other data types require the adaptation of instruction sets and the inclusion of the quantization operations, hindering the implementation in these cases.

Such is the case of extreme quantizations, with data types of 4, or fewer bits. They haven't shown sufficient benefits compared to more standard quantization types, as to overcome the hardware and implementation difficulties and bring them to general hardware deployment platforms. Hence, they are still restricted to the research and development environment [222].

More importantly, there are factors during quantization that are in fact being incorporated in deployment and development suites. One of such topics is the importance of how the parameters are selected since it can affect the results of the quantization process. For example, the distribution of quantized weights are distributed is of utmost importance [229]. As detailed in [230], CNN weights follow gaussian distributions when weight decay is applied. Moreover, it has been shown that weights generally follow Gaussian Mixture Models [231]. Hence, being able to define a non-uniform quantization distribution would allow reducing the quantization error.

In the same sense, the granularity selected for quantization is of importance [232] and has been incorporated into most suites. In general, the more specific the granularity the better, but with a trade-off with the difficulty of implementing the operations. In Figure 2.1, quantization per channel delivers better results than per layer.

An important factor that we have not explicitly tackled is the combination of quantization and other compression modalities. A good example is a combination with distillation where it can help reduce the accuracy drop of the quantized model by having a floating point trained teacher [232, 233, 234]

Finally, another important topic, is the benefit of quantization-aware training [235, 10] using simulated quantization to avoid accuracy drops. Moreover, as depicted in Figure 2.2, how the quantization is applied during training is also important, with much bigger benefits once the model has been trained with floating point values.

2.1.5 Conclusions

In the present section, we have reviewed quantization for neural networks, its characteristics, the different possibilities, and its adoption in major development frameworks and hardware platforms. As main conclusion, we could point out the benefits in model size and latency reduction for most models, but with the caveat of a variable decrease in accuracy. However, such drop can be reduced with different techniques such as for example, quantization-aware training.

With regards to its final usage in applications, albeit there has been a lot of research with different quantization schemes and data types, the most common ones and suited for hardware platforms have become the only ones widely available for usage (16-bit float, or 8-bit integer). However, some development frameworks, such as PyTorch, have started easing the use of other data types for quantization, promoting its research and use.

All in all, quantization supposes an excellent way to decrease the computing requirements and accelerate inference while maintaining accuracy or playing a trade-off with a drop on it.

Finally, we haven't covered quantization extensively, but the most used methods and techniques that have been involved in this thesis. For those interested in more details about the methods presented and other possibilities regarding quantization, as well as the results provided by them, we suggest the following reviews among others. [226] gives a concise summary of the most used methods and how they function. [10] provides more detail for fake quantization in a quantization-aware training context as well as extensive testing of the different methods and the results obtained, [236] focuses on the quantization process of a network and the results on extremely quantized networks, and, finally, [237] focuses specifically on the efficient component on inference.

2.2 Neural Architecture Search

2.2.1 Introduction

Part of the success of deep learning in visual tasks comes from the manual design of hierarchical feature extractors [85, 238, 14, 16], which automate and condensate the feature engineering process. With time, the complexity of such architectures has increased as the search for increased performance has continued. Such a building process has become time-consuming and error-prone. Thus, Neural Architecture Search (NAS), which consists of the automatic building of neural network architectures, is a natural progression step. Currently, NAS has already over-passed manually generated networks in tasks such as image classification [190, 239] or semantic segmentation [190]. Deeply tied to AutoML [240, 241], it is also linked to hyperparameter optimization [242, 243], and meta-learning [244].

The main core components of NAS are the search space, the search strategy, and the performance evaluation strategy [187]. The search space defines the configuration options available to build the network, the search strategy defines the way new configurations are sampled or parameters of the architecture changed, and finally, the performance evaluation strategy defines how the performance of proposed architectures is estimated or predicted. The simplest case with regard to evaluation is to fully train the networks and seek to achieve the highest validation accuracy. However, this is extremely costly in terms of resources, both energy and time. Thus, lately, new ways of predicting/anticipating the performance of architectures without the need for training or, in general, ways to reduce the search time [187], have become an important line of research.

Moreover, most first NAS frameworks only focus on the accuracy of the NN, delivering complex or inefficient networks. Lately, this has been remedied by taking into account other objectives such as memory consumption or latency [188, 182], thus delivering NNs that are efficient or comply with certain resource restrictions. Additionally, in order to push efficiency further, some studies orient their NAS frameworks to hardware platforms, in a general approach [245, 35], or targeting more specific platforms, such as mobile [246], MCUs [188] or FPGAs [247], among others.

Such advances have pushed NAS forward, but there still are many lines of progress to advance the automatic building of NNs, such as reproducibility or multi-task/multi-objective environments, which make NAS a current state-of-the-art problem. In the current section, we review briefly the history of NAS and its fundamental components. We also discuss its main problems and their relation to the present thesis to finally conclude with the principal points illustrated.

2.2.2 Brief History

Evolutionary algorithms were used in the 90s and first 2000s to evolve neural architectures, as with the first notable work [248], and also to jointly find the optimal weights [249, 250, 251]. A good review of such methods can be found in [252]. However, it must be noted, that most current evolutionary methods only use evolutionary algorithms to develop the architecture but not to find the optimal weights [187, 253, 254], where back-propagation is the most common algorithm.

Bayesian optimization gained momentum in the 2010s, advancing state-of-the-art NNs architectures [255] and finally setting a cornerstone with the first NAS-built NN winning over human experts [256]. Finally, NAS became an established research topic after [257] obtained competing results in the CIFAR-10 and PennTreebank datasets by using reinforcement learning.

Since then, the field has advanced not only focused on achieving better performance but also taking into account other important factors. In this sense two directions stand out: first, reducing the search cost in terms of energy and time, and second, including in the NAS objective other important factors of the NN, such as memory consumption, latency, or energy.

2.2.3 Fundamental Review

Search space The search space defines the possibilities for building the final network: what elements can be used, how are they can be defined (for example, which values can be assigned), and how they can be connected.

In its more simple case, the search space is defined by a *sequential architecture* and its components. The number of layers, what is the type of each layer, and hyperparameters for every specific type of layer (number of neurons, kernel size in convolutions, etc) conform to the typical components of the space. Some of the first NAS research projects were based in such a search space [256, 258]. It is important to note the conditionality of this type of search space (for example, the fact that there are seven layers, impacts the fact that the kernel size of the 8 layers is not needed), which has a big impact on the modelization of the space and the search of new networks, as we will see in the search strategy section.

Modern neural networks use tricks to improve convergence or performance that involve multiple connections among layers, such as residual or dense connections. The addition of such a feature to the search space, brings the second different type of NAS search spaces **branched architectures** [259, 260, 239]. The main difference with respect to the previous is that each layer now has another parameter: the input connections coming from other layers.

Taking inspiration from manually designed neural networks that are built by repeatedly using a pattern with different parameters, as in the case, for example, of the MobileNet family [15, 16], another type of search space was defined: **cell** or **block** based search space. In this case, the search is performed on the contents, parameters, connections inside the cell/block, the types of different blocks (for example, [190] details two types of blocks: normal cell or reduction block, which reduces the dimensionality), and the way and quantity to stack or scale them. This brings two main benefits: *a priori* reduction of the search space and better generalization in transfer learning due to great reusability. This method has been largely and successfully in many research projects [261, 262, 263]. However, the problem of how to organize the newly found cell in the overall architecture is still present. Such configuration is called the *macro-architecture*: how are cells (*micro-architecture*) combined to form the overall network? Although first attempts used handmade rules [190] or [263] imitated other architectures, the combination of both spaces is needed [187], as they are intertwined forming a hierarchy of choices and configurations [264].

In general, the more complex the configuration or search space, the more complex is going to be the modeling of its relation to the performance of each network.

Search Strategy The object that searches through the search space, models its relationship with the performance, and samples new architectures, is the search strategy. There are four main search strategy types depending on which underlying algorithm is used: bayesian optimization, evolutionary algorithms, reinforcement learning, and gradient-based methods. We will briefly cover the first three.

Reinforcement Learning (RL) is suited to frame the NAS problem with its components: state, agent, action, and policy. The natural choice is to assign the generation of the architecture as the action of the agent, the search space to the action space, and the performance of the network with unseen data to the reward [257, 265, 266]. The policy, however, has not have a direct assignment and there are different choices, such as, for example, using an RNN policy trained with REINFORCE algorithm [267] to sample sequentially the architecture [265]. Nevertheless, this is not the only option to frame the problem with, and there exist many more, for example, by assigning the architecture as the state and the actions are morphisms [268] applied to it [258].

Evolutionary algorithms (EA) are also well suited for NAS. The population in this case consists of a set of models and mutations are modifications to a model, for example changing a parameter to a layer, adding one layer, connections, and so on. When a sample is selected, the offspring set (mutated version of the parent) is trained and evaluated on the validation set to be added to the population. It is important to note that most modern methods [189, 253, 254] use EA only for NAS and not for training the networks, which are trained using gradient-based methods. Where most methods differ is in the way of updating the population, generating offspring, and sampling parents. For example, with regards to offspring generation, an option is to generate them randomly, but inheritance, that is, maintaining learned weights, and morphisms can also be used [189]. Another classic example is how to update the population, where deleting the worst performing individual is an option [269], but also removing the oldest one [239].

Bayesian Optimization (BO) uses a surrogate model to model the relationship between the search space and the performance metric obtained by each configuration, that is

$$p = f(\mathbf{x}) \tag{2.11}$$

where p would be the metric, f the surrogate model, and x the architecture configuration vector. Thus, the surrogate model has to be fitted. Each trained architecture and the corresponding metric conform to a sample, with all the samples

$$S = \{(\mathbf{x}, f(x)), (\mathbf{x}_1, f(x_1)), ...\} \tag{2.12}$$

one can fit the surrogate model to model the relationship such as the following difference is minimal [270]

$$\sum_{\mathbf{x}, f(\mathbf{x}) \in S} (\hat{f}(\mathbf{x}) - f(\mathbf{x})) \tag{2.13}$$

where $\hat{f}(\mathbf{x})$ is the predicted metric by the surrogate model.

Methods differ in the model used for the surrogate model. A possibility is to use tree-based models, like the Tree-parzen estimator [271] or random forest models [272]. These models are well suited for conditional spaces. Common use has been to employ them to jointly search for architectures and hyperparameters [255, 273, 256]. Many advances have been produced in this direction, with, for example, Monte Carlo Tree Search [274] or hill climbing algorithms [260].

An alternative to tree-based models is to use Gaussian Processes (GPs) [275]. Apart from the surrogate model, in this case, an Acquisition Function is used to sample the next points in the search space, trading-off different strategies the predicted metric, and the uncertainty associated. The main difficulty with

GPs is that they are not well suited for modeling conditional spaces. For this reason, a special kernel (the function that measures the distance between points in the modeled space) has to be defined. An example is the Arc Kernel developed by [276] and implemented in PyTorch in this thesis [277], or the one presented in [278].

Finally, an alternative to the previous two models is to use a neural network as a surrogate, like RNNs [279] or autoencoders [280], among other possibilities.

Performance Evaluation The natural and naive way to evaluate an architecture is to train it and evaluate it on unseen validation data according to some accuracy or performance measure. However, this can be extremely costly, in terms of energy and time if every time a new architecture is found by the search strategy it has to be trained from scratch (spending up to thousands of GPU days [239, 269].

There are ways to reduce the time spent training and searching, among which stand out: using lower-fidelities (or proxy metrics), curve extrapolation, weight inheritance, and one-shot NAS, among others [187].

Lower-fidelities estimate the performance of the full metric by using a modified context, including training for fewer epochs [190], training on a subset of data, on smaller images [281], among others. The main risk, in this case, is that the difference between the full context and the lower fidelity is too big and the ranking of evaluated networks changes [282]. This risk can be reduced using incremental fidelities [283].

Curve extrapolation [284, 273, 285] (acquisition functions could be seen as a form of curve extrapolation) pursues to estimate the evolution of learning curves so as to disregard architectures that perform poorly before training them fully.

Weight inheritance consists in using weights of previously trained networks for the new sampled network, thus reducing the training time until convergence and the number of total GPU days [286, 258]. The new networks can be produced from the original network by means of morphisms or architectural changes [287]. Another advantage of this process is that the network search space is unbounded, allowing also for diminishing morphisms [189].

Finally, the one-shot architecture search trains only one network, and all the other sampled networks are defined as a sub-graph of the original with the corresponding weights shared [288, 245, 289]. Subnetworks can be then evaluated without training them, speeding up network evaluation and reducing the GPU days employed. This also helps to produce a full Pareto frontier of networks with regards to computing resources [35, 245]. The main difference between one-shot NAS architectures is how they train the main architecture. For example, ENAS [290] use an RNN controller to sample architectures from the main architecture and trains the one-shot model taking into account approximated gradients. An important limitation of one-shot NAS is the restriction to the original network and the subspaces that it defines.

2.2.4 Discussion

Most existing work on NAS has focused on image classification, however, it has already been applied to other fields such as image restoration [291], detection [292], image segmentation [293], and many other tasks. Initially centered on only achieving improved accuracy, new methods have appeared that focus on multi-objective NAS, taking into account other metrics, such as memory consumption, model size, or latency [294, 295]. However, this multi-objective frame requires the balancing of the different objectives to find the Pareto frontier, pushing for the incorporation of multi-objectives in search methods [296, 297].

Moreover, including the multi-objective setting, NAS methods have also started focusing on hardware-related platforms, building methods focused specifically on a platform or on being efficient for hardware [298]. For example, studies such as Sparse [188, 4], MCUNet [], or MicroNets [299], have targeted MCUs, pursuing the development of NNs automatically deployable in MCUs following a set of hardware constraints. Also stands out the case of mobile platforms, with studies targeting specifically mobile platforms [246, 298, 300, 301].

Additionally, the reduction of search time and energy has also acquired renewed importance. Early NAS methods required vast amounts of energy and computation time, proving inadequate given the improvements they provided. For example, [265] used 800 GPUs for between three and four weeks to achieve their results. Since then, an effort has been applied to reduce time search with more efficient search methods, low fidelity proxies, and other schemes [290, 302, 303].

An important problem of NAS concerns the search space. State-of-the-art NNs are usually manually engineered for performance through an expertise-requiring process, and thus it is difficult to design an open or standardized search space in which to find an outperforming NN. On the contrary, usually, such engineered architectures are often used as common blocks to design search spaces, introducing restrictions to the type of architectures discovered. The classic *exploration vs. exploitation* problem takes a renewed view in NAS, where it plays an outstanding role. In such terms, trying to find search spaces that are more general and transferable stands out in the works [304, 264].

Following a similar discussion, an important discussion is centered around the usefulness or worthiness of NAS compared to random search (RS). For example, [239] performs a comparison between RL, GA, and RS-based NAS. Among the findings, stand out the fact that, in small datasets, like CIFAR10, GA and RL outperform RS but only with a small margin of around 0.5% error on the test set. Similar findings have been shown in [264], where margins were even smaller and on a larger dataset: the validation error difference between RS and GA on the ImageNet dataset was of around 0.7%. However, two main a factor not taken into account is the time spent to obtain such results and the fact that NAS is highly dependent on the search space configuration.

Additionally, there is a reproducibility problem [189, 305] when comparing NAS methods, for example, with regard to the search method. The main point is that results do not only depend on the search space and dataset but too many hyperparameters or training strategies. Including a learning rate cosine annealing schedule, using better data augmentation, or regularization, among many other factors, can help improve results. Thus, it is necessary to standardize training conditions, as well as the search space and data. A step towards such purpose is found in [306, 307, 308], where the search space has been standardized, and in [309], where the hyperparameters are also considered. The idea of a full AutoML suite, where everything is standardized so conditions among runs are maintained, has already been proposed in [187].

Finally, another important concern about NAS is that it has little associated explainability: it does not assess why certain architectures are better than others or why similar architectures have such a marked difference in performance.

2.2.5 Conclusion

In the present section, we have reviewed NAS functioning, and its components, and pointed toward several points that were of interest for the present thesis.

Overall, NAS is an excellent technique to produce networks that are high performing. Moreover, the latest advances in multi-objective NAS have helped deliver in an automatic manner networks that target not only accuracy but other constraints, such as memory or energy consumption, model size, or latency. This has also fostered NAS for specific platforms like MCUs or mobile ones, helping to bring ML to hardware-constrained environments.

However, there are still major lines of improvement for NAS, including explainability and reproducibility. Also, the improvement of search efficiency and its worthiness regarding resources used are major breaking points. Additionally, NAS for resource-constrained platforms can still be improved in many terms, such as, for example, the integration with the software environments, and deployment frameworks (with all their restrictions regarding operations).

We have not covered extensively the NAS topic, but rather introduced it and highlighted the most important concepts with regard to this thesis. If the reader is interested, detailed reviews exist for this topic, among which we recommend [187, 270] for the basic components and understanding of the status and evolution of NAS, and [310] for a more in detail review of current challenges and solutions. For GA-based NAS-specific review, we suggest [311].

2.3 Pruning

2.3.1 Introduction

Pruning is a compression technique that focuses on the deletion of a certain part of a network's weights to reduce the model size, or additionally the latency of the model while producing a minimum accuracy drop. That is, given a network, $f(x, \boldsymbol{\Theta})$, where x is the input and $\boldsymbol{\Theta}$ the weights of the network, a pruning procedure would dismiss, according to a certain evaluation or scoring metric, a subset of the weights and select the rest $\boldsymbol{\Theta}'$ as the final weights. The proportion of the final weights with respect to the number of initial weights is termed the *sparsity*, s, of the final model, $s = 1 - \frac{||\boldsymbol{\Theta}'||_0}{||\boldsymbol{\Theta}||_0}$, and the dismissed

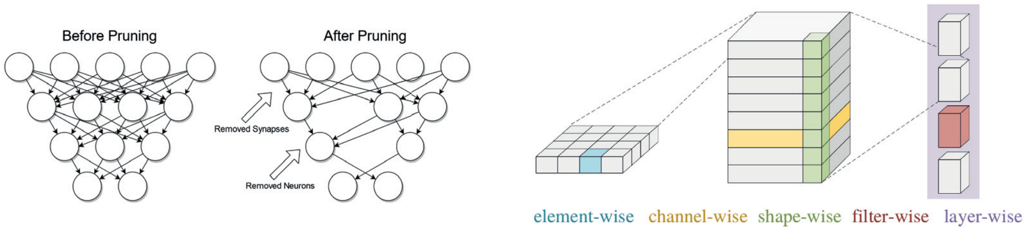


FIGURE 2.3: (Left) Illustrative simplification of the pruning procedure targeting connections and neurons in a fully connected network. (Right) Visual example of shape-wise pruning [11]. Image adopted from [12].

parameters are usually assigned a 0 value or deleted. A simplified illustration of pruning can be found in Figure 2.3

When applying pruning to a network there are four important factors to consider which will affect the final result of the process:

- **Valuation Metric**: how are the weights ranked and valued?
- **Structure**: is pruning applied to single weights directly (*unstructured*) or is it applied to entire filters (*structured*)?
- **Distribution**: how is sparsity distributed through the entire network?
- **Scheduling**: when is pruning applied?

It is important to consider that a pruning method does not only deliver a sole sparse final model but a collection of models with different sparsity-quality and which are generated during the pruning procedure.

Algorithm 1 Standard pruning procedure.

```
Require: Network with weights $\theta$, Dataset $D$, number of pruning rounds, $N$, the fraction of weights pruned per round $p$ Output: Network with pruned weights $\theta'$ and sparsity level $s$
  $\theta \leftarrow \theta'$
  for $n \leftarrow 1$ $toN$ do  $S \leftarrow$ compute_saliencies($\theta'$)
  $\theta' \leftarrow$ prune_p_weights($S, ||\theta'||_0 p$)
  $\theta' \leftarrow$ finetune($\theta'$)
  end for
```

2.3.2 Brief History

Originally developed in the late 80s [312, 165], it gained attention with the seminal works of Optimal Brain Damage (OBD) [157] and Optimal Brain Surgeon (OBS) [313]. Both works are examples of second-order derivative usage, by magnitude and change respectively, to select weights to delete. Also, both set up the traditional algorithm for pruning, as explained in Algorithm 1.

During the 2000s few developments and improvements were made with regard to pruning for neural networks. However, linked to the success of NNs in the 2010s, pruning research recovered attention. Seminal works like [158], which restated the procedure of Algorithm 1, opened the door for new approaches. Since then, new developments with regards to scheduling [314, 315, 316], scoring metrics [317], distribution, and fine-tuning [318] have pushed forward pruning research.

However, a double change of paradigm was initiated by the works [161], further extended in [319], and [320, 321]. In [161], the mindset moves from sparsifying a network to being able to find an internal subnetwork equally competing with the bigger one, and in [321], from learning meaningful weights to just focusing on the architecture, thus without the need of training for pruning. These changes have set off a set of fresh research to better understand pruning, instead of merely focusing on the benefits at the size or latency levels.

2.3.3 Fundamental Review

Valuation metric The main component of the pruning procedure is the evaluation and ranking of weights (or filters) to prune them. Such evaluation can be focused on the magnitude, redundancy, or sensitivity of the weights or filters w.r.t the loss [160].

Magnitude-based pruning

In general, magnitude-based pruning can be separated into three groups [160]: data-dependent, data-independent, and optimization methods.

Data-independent methods consider only the characteristics of weights or filters. For example, it is widely accepted that the magnitude of the weight is indicatively proportional to its importance [322]. Thus, the first naive pruning procedure would be to prune all zero-valued weights or those below a threshold. [158] is an example of how threshold pruning, altogether with iterative pruning, can help to reduce the size factor of a network. Importance magnitude is usually computed through a p-norm,

$$||\mathbf{x}||_p = \left(\sum_{i}^{N} x_i^p\right)^{\frac{1}{p}} \tag{2.14}$$

where $\mathbf{x}$ would be a set of N weights. The same metrics are applicable if pruning is going to be applied filter or layer-wise [323], by removing filters with the least accumulated value of weights. Other examples of data independent methods involve the minimization of co-variance between original and pruned filters [324], deleting filters near the median value of other filters [325] or the Average Percentage of Zeros (APoZ) [326], which forms a synergic combination with ReLU activations. An important note, however, is that direct thresholding can delete weights that later on can become important. In such a sense, [327] proposes a system to keep track of which weights are marked for deletion, without deleting them until a further step.

Data-dependent methods make use of data samples to estimate related measures in feature maps. Variance is a common measure [328], where filters are valued with regards to the variance in the respective feature maps, as well as entropy [329], which helps prune filters whose feature maps carry less information.

Finally, optimization-based pruning methods base the procedure on finding the minimum set of filters that best represent the original feature maps. This can be done through feature map approximation through least squares, as in [330, 331], or by computing the effect of deletion of filters and the difference between original and new feature maps [332].

Sensitivity-based pruning

Instead of considering its value or its norm, sensitivity-based pruning ranks the weights by their impact on other measures, usually the loss. Such impact is generally measured by perturbing or deleting the weights and computing the effect on the loss. Two main groups of sensitivity-based strategies can be distinguished [160].

First, *importance methods* measure the effect of directly deleting parameters by adding accompanying parameters to nodes or groups of nodes, or through penalty components in the loss. [165] is an example where the importance of a node is measured by the derivative of the loss with respect to its accompanying parameter, in this case, named *attentional strength*. Works such as [159] or

[333] extend importance methods to channels and filters, in this case through penalty terms added to the loss, as in Lasso regression

$$\mathcal{L}' = \mathcal{L} + \lambda ||W||_2 \tag{2.15}$$

where W are the weights, λ the regularization factor, and $\mathcal{L}$ the loss. Such penalty term forces weights to reduce their value, setting importance differences among them. l-1 norm can also be used to drive weights to 0, being named Ridge regularization.

Second, **Taylor's approximation methods** measure the impact of weight deletion through a Taylor expansion of the loss:

$$\Delta\mathcal{L} = \frac{\delta\mathcal{L}^T}{\delta W} + \frac{1}{2}\Delta W^T H \Delta W + \mathcal{O}(||\delta W||^3) \tag{2.16}$$

where H is the Hessian matrix. Then, most methods distinguish w.r.t. which order they approximate to, usually neglecting third or superior orders. [157] consider only second-order terms since the network is supposed to be in a local minimum and further suppose the Hessian to be diagonal, that is, weights are uncorrelated. [313, 334, 335] consider such later assumption as incomplete, and add an approximation to the full hessian. [336] extend Taylor expansion methods to channels and filters but only taking into account first order parameters, and further extend their work to consider the squared loss change due to parameter deletion [337].

Most of the previous methods do not take into account weight interaction, assuming no correlation. [338] consider the full Hessian matrix approximated through a Fisher Information Matrix represented by a Kronecker Factored Eigenbasis [339], thus taking into account correlations between weights. In the same line, [340] uses a Collaborative Channel Pruning method to assess the impact of a combination of channels through Taylor's approximation of the loss.

Finally, although most of the methods compute saliency or penalty through the loss, there are other possibilities: BN penalty addition [159], applying LASSO to scaling factors [333] or adding dropout hyperparameters [98] are examples of some of them.

Redundancy-based pruning

Magnitude-based pruning can end up deleting weights that even if their value is small can play a role, as in the case of networks with largely saturated weights [341] or large minimum norms [325], and the same happens with a penalty based scoring [327]. As an alternative, *redundancy* [163, 342, 158, 343, 344] can be another option to prune weights. Given a sufficiently large network, is plausible that there are duplicated parameters or groups of weights that perform similar solutions. Then, those that are redundant can be deleted or grouped. Among the different typologies for identifying redundancies stand out similarity measures [342, 345, 346], clustering [347, 348], distance and geometric features [325] or statistical significance [328, 330].

Structure Pruning can target the network with different granularity and order. It can be applied to individual weights or connections in a random manner, that would *unstructured* pruning, or it can target bigger units such as filters or entire layers, that is, *structured* pruning [167, 11, 349, 13].

The majority of research is targeted at unstructured pruning because it can deliver the bigger sparsity with the minimum effect on accuracy. However, the main downside of unstructured pruning is that without special hardware operation kernels it cannot provide latency improvements. Only if the model is compressed it can deliver size improvements. Both aspects are due to the fact that sparsity is unordered and cannot be deleted. On the contrary, structured pruning, as it targets whole blocks in an ordered manner, can be used for decreasing latency by deleting such blocks.

However, a notable example, of how, with adequate hardware kernels, unstructured pruning can help increase efficiency is found in [350], where they develop sparse kernels and include them in the XNNPACK library for fast sparse CPU computation, delivering an improvement of 1.3-2.4x with respect dense equivalents.

Distribution How pruning is distributed along the network affects both accuracy and inference cost. That is, pruning all layers equally performs worse than distributing the amount of pruning wisely [351, 323, 336]. As an example, it is sufficient to consider a convolution layer: its application to a spatially bigger input involves a greater computation cost than to a smaller one, and thus pruning will have more effect in the first case [352].

In [350], examples of pruning distribution criteria can be found, where hard-coded distribution policies are implemented for pruning. For example, albeit

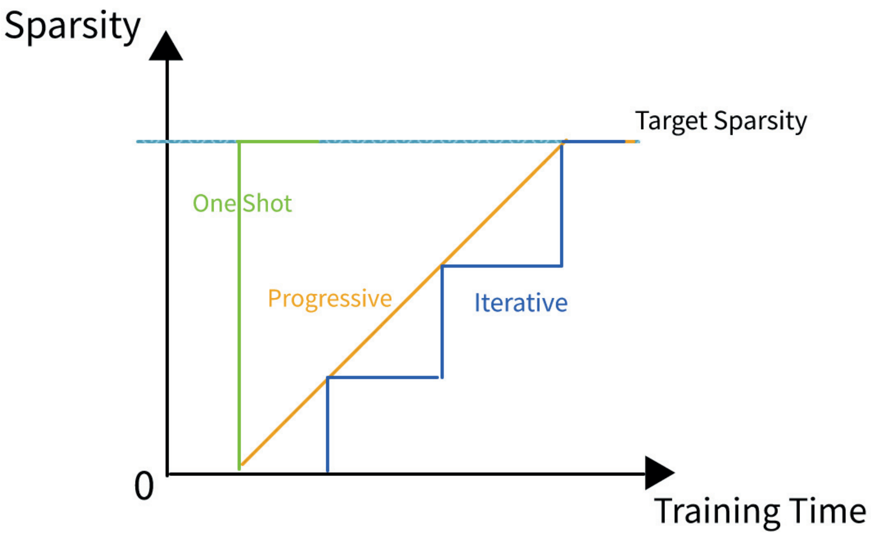


FIGURE 2.4: Graphic illustration of the three choices for scheduling pruning: one-shot, iterative, and progressive. Adapted from [13].

the final layer in a classification model contributes significantly to the number of parameters, its pruning can affect severely the accuracy and so it is not normally pruned. The same usually happens for the first layer.

Scheduling Another important factor with regard to the pruning procedure is when to apply it and if the network is retrained or fine-tuned afterward. Then, we have two main steps that distinguish pruning scheduling: pruning and training. The traditional and common way to apply theses steps [157, 334, 158] is to iteratively train and prune. The pruning procedure would be as follows. First, a network is trained until reasonable results are obtained. Second, weights are valued according to the saliency metric and then a subset of them is pruned. Third, the network is fine-tuned with new sparsity values. Finally, repeat steps two and three N times. The whole procedure is illustrated in Algorithm 1.

However, there are two more possibilities for pruning scheduling [353, 13]: (1) one-shot, when the network is initially pruned up to a predefined sparsity and then fine-tuned, (2) progressive, when the network is trained but at the same time pruned progressively during training iterations until reaching objective sparsity and, then fine-tuned again. However, it is commonly agreed that both iterative and progressive pruning outperform one-shot pruning given the same conditions [318]. An illustration of the different schedules for pruning can be found in Figure 2.4.

Although usually only the final pruned model is used, there are pruning algorithms that require the models from some previous pruning iterations. The idea is that, while the network is learning the pruned weights might change, thus, if one preserves the mask of pruned weights, one can avoid unnecessary pruning [313, 338, 13]

2.3.4 Discussion

Lottery Ticket Hypothesis. An important discussion surrounds the value of pruned weights, and whether they are useful for convergence by providing a mask for the initial bigger network. The Lottery Ticket Hypothesis [161] states that "*a trained network contains a subnetwork, which can be trained to be at least as accurate as the original network using no more than the number of epochs used for training the original network*", and establishes the usefulness of the mask provided by pruned weights at iteration K to initially select a subnetwork. To clarify the procedure, it is as follows: pick a network f and train it until convergence at iteration K, prune it and obtain the mask of the remaining weights, m, then pick the original network at initialization and prune it with mask m, delivering the subnetwork f_m, which can be trained until the same k iterations and obtain similar or better results with a portion of the original size. This idea has attracted quite an interest due to the traditional belief that a pruned network could not be retrained from scratch to attain similar performance to its original network [353].

However, [315, 351] were unable to replicate the usefulness of pruned weight initialization compared to random initialization, with the exception of large-scale data with unstructured pruning. In a further study, [319], refine the hypothesis to the use of the mask at another intermediate iteration k coming from later stages of training. Subsequent research has revolved around further questions and their answers, concluding, for example, that pruning can be beneficial for transfer learning [354, 161] between different tasks and even types of problems, like NLP or RL [355], or that pruning is optimizer independent [356]. All in all, it can be concluded that iterative pruning and reinitialization with late-stage training weights help to improve efficiency.

Pruning at Initialization. A new branch of pruning research, named Pruning at Initialization (PaI) [353], stems from the works [320, 321, 357, 358] and focus on pruning before training or training slightly to discover relationships among weights and prune them. SNIP [320, 321] is one of the first works in this direction, centering attention on the saliency of weights through random loss preservation. That is: how much does the loss change if weight is absent? The idea is translated to training dynamics at the first steps of training by [357] using a *gradient signal preservation* (GraSP). They include a perturbation in a second-order Taylor approximation of the loss (equation 2.16) to obtain the saliency of weights,

$$\mathbb{S}() = 2^{T}\mathbb{H} + \mathcal{O}(||||_{2}^{2}) \tag{2.17}$$

where $\mathbb{S}$ is the saliency, the perturbation vector, the gradient, and $\mathbb{H}$ the Hessian.

Another research branch of PaI is not focused on the loss or training, but rather on the topology of the network, and solely based on it, the pruning can take effect. [359] focus on sub masks learned by the LTH but without retraining the subnetwork, arguing that it already has knowledge. [358] further extend this idea by keeping scores for weights but without training them, obtaining good results on ImageNet. Theoretically, the idea is finally developed by [360], which defines the *strong* LTH: inside an over parametrized network, there is a subnetwork with notable performance and without training.

Inference speedup. Due to the lack of sparse kernels in hardware platforms, unstructured pruning only serves for size reduction and through the usage of compression tools like GZip [226]. There are, however, specialized kernels, but with limited application, for sparse matrix-vector multiplication [361, 362, 11, 13]. Examples are [350, 363], who developed sparse convolutional kernels with a decrease in latency of 1.3-2.4x, around a 2x factor reduction in the number of parameters, and a 3x reduction in FLOPS with respect the previous generation while maintaining top-1 accuracy.

On the other hand, structured pruning, if appropriately managed, e.g. by choosing appropriate partition blocks and adequate kernels, can conduct both latency and storage improvements [364]. Thus, the latter is preferred for efficient inference in the case no suitable implementation is available.

2.3.5 Conclusion

In general, it has been proven that pruning strategies work: pruning models are able to consistently outperform random pruning [352, 331, 351, 346] and compress models showing minimum drops, or even increasing, in accuracy. Moreover, for a fixed number of parameters, pruned models tend to outperform models trained from scratch with no sparsity induction process. However, there are two main downside points as stated by Blalock *et alli* [352]. First, pruning does not deliver as many benefits as changing to a better-performing architecture. And second, there are not sufficiently standardized benchmarks and conditions in the pruning community as to allow fair comparison among methods and distinguish improvements and better working methodologies [352, 160].

Notably, if one's purpose is to decrease latency, although there are sparse kernels available [350, 363, 361, 362], their support is limited and hence is better to apply unstructured pruning at the filter or layer level.

It has also been seen that there has been a change of paradigm in pruning: from traditional deletion and training based to sub-network and architecture-based pruning. This new direction has opened a new line of research that still has questions and difficulties to focus on, e.g., they still under-perform simple traditional pruning [351, 353].

Finally, pruning is a prominent and current research topic, and we have tried to cover the main aspects of its w.r.t our focus, not being able to include all information about it, for example, combinations with other optimization methods like distillation [365, 366] or its relation to energy efficiency [367]. To the reader interested in deepening the knowledge about pruning, and also its use for developing efficient models, there are plenty of resources available. Among them the following reviews are suggested: [160] focuses on metric scoring, [226] has a brief review with a focus on model deployment, [352] questions comparability in pruning, [353] focuses on the change of paradigm made by the Lottery Ticket Hypothesis and its comparison to previous schemes, and [12] review both pruning and quantization.

2.4 Efficient operations

2.4.1 Introduction

The early and subsequent success of CNNs for classification came at the cost of higher computational complexity. Soon these networks and operations were applied to other tasks such as semantic segmentation or super-resolution. The higher spatial resolution required for this further increased the computational complexity. The need to reduce it and bring these networks to platforms with fewer resources, entailed and initiated a field of research specially devoted to making neural network operations more efficient.

A possible name for the field would be deep learning efficient operations research. Its focus is the analysis and development of deep learning operations so as to reduce their computational cost while ensuring the least accuracy or performance drop. Although important works have been devoted to CNNs, other architectures have also been the subject of the previous pursuance, as in the case of RNNs or attention mechanisms.

An important difference with respect to other optimization methods is that the benefits are applied before training and there is no need for subsequent procedures during and after training. Notably, the first efficient operations designed were handcrafted and manually designed. Lately, however, there have been recent works improving the state-of-the-art by combining NAS and efficient operations, to search for best-performing operations automatically.

Overall, efficient operations are an important optimization method, delivering the performance benefits from the start, and being specifically useful if the target platform is already defined. In this section, we review some of the most important works on the matter, describe their methodology, and discuss their advantages compared to other types of optimizations.

2.4.2 Fundamental Review

The first notable example of improving an architecture by making use of more efficient operations is the substitution of FC layers with convolutional layers in visual tasks. Fully connected layers break spatial relationships since they operate as vectors, and being a dense operation leads to an explosion in the number of parameters. Meanwhile, convolutional layers, by reusing spatial filters of low dimension are able to both conserve spatial information and reduce the number of parameters employed. Additionally, the use of pooling layers helped reduce further spatial dimensions, decreasing the number of operations performed.

Explicitly, for an image of size $[C, H, W]$, a fully connected layer with N neurons has a total of $C \cdot H \cdot W \cdot N + N$ parameters, while a convolutional layer with F filters of dimension $[k, k]$ would imply $(C \cdot k \cdot k + 1) \cdot F$. Typically, for images $k << H\, W$ and $F < N$, which illustrates the decrease in the number of parameters. LeNet [79] was one of the first architectures to introduce convolutional and pooling layers, and since then and until transformers, most of the state of the art networks for visual tasks (AlexNet [85], ResNet [14], Inception [368], among many others) have continued to use convolutional layers.

Depth-wise Separable convolutions constitute an improvement in efficiency for regular convolutions. They operate in two steps:

1. Perform a depth-wise 1x1 convolution with F_o output channels
2. Perform a kxk convolution without changing the number of channels, F_o

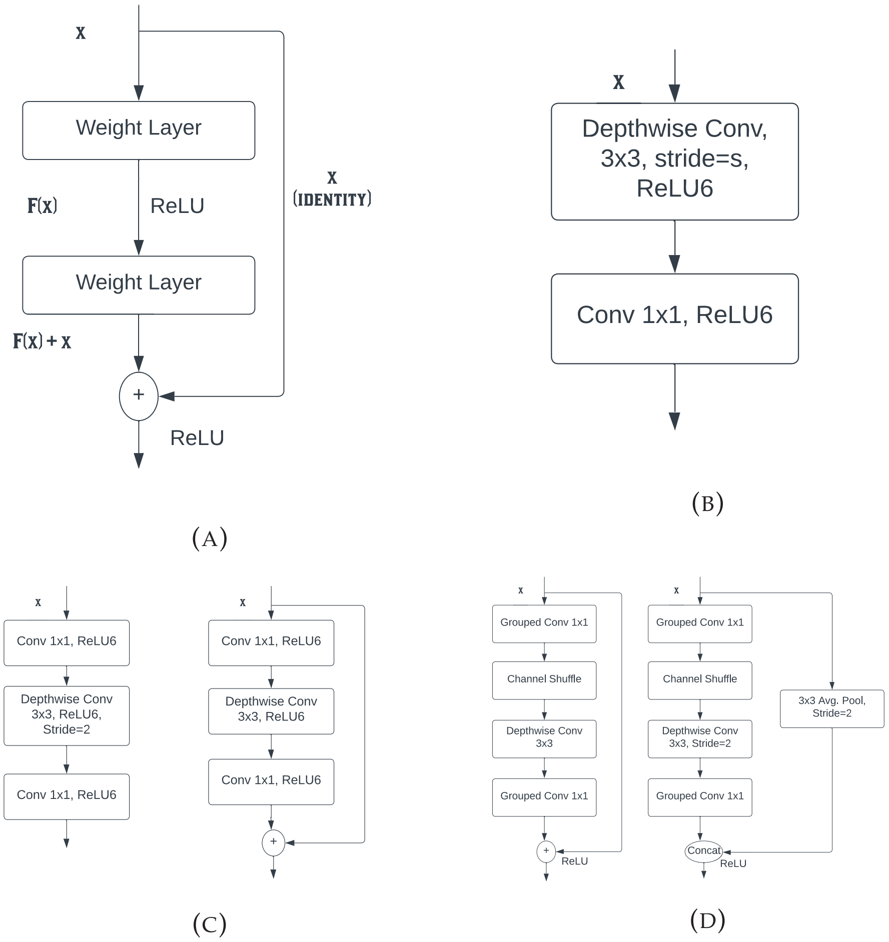
Then, if the dimension of the input is $[F_i, H, W]$, the resulting computational cost is $(1 \cdot 1 \cdot F_i \cdot F_o \cdot H \cdot W) + (k \cdot k \cdot F_o \cdot H \cdot W) = H \cdot W \cdot F_o \cdot (k^2 + F_i)$, which is lower than the cost of a traditional convolution $H \cdot W \cdot F_o \cdot k^2 \cdot F_i$. More details can be found at [16]. It has proven, as in the case of Xception [369] or ResNeXt [370], to be a way to increase convergence and keep a good trade-off between accuracy and the number parameters.

The idea is further extended in MixNets by applying differently sized convolution kernels [371].

Residual mechanisms were introduced in [14], and although they did not introduce explicit efficiency improvements in inference, they helped build larger networks by mitigating the vanishing and exploding gradient problems. Furthermore, they have served as a base for posterior improvements in efficiency, such as the next operations. An illustration is found in Figure 2.5.

Inverted residuals and linear bottlenecks Both improvements were proposed in [16]. Inverted residuals consist in, contrary to traditional residual connections [14] which perform a $wide \rightarrow narrow \rightarrow wide$, performing a $narrow \rightarrow wide \rightarrow narrow$ sequence regarding the number of channels. By applying a point-wise, a depth-wise convolution with higher spatial filters, and finally another pointwise operation the number of parameters can be reduced. The total computational cost for an inverted block is then $H \cdot W \cdot t \cdot d'(k^2 + d' + d'')$, where t is the expansion rate from narrow to wide number of channels, and d' and d'' are the number of channels respectively.

As the number of parameters and operations is reduced, the explanatory capacity of the network is also reduced. Thus, in order to avoid the damage that non-linear activations like ReLU cause to the information contained in an option to add the residual information without a non-linearity, thus the name linear bottleneck (bottleneck indicates the fact that the residual is connecting filters with low numbers of channels).



(A) (B)

(C) (D)

FIGURE 2.5: Example of efficient operations. (A) Residual connection [14]. (B) Depth-wise separable convolution [15]. (C) Depthwise separable convolution with inverted residual and linear bottleneck (right) and without residual (left) [16]. (D) Shuffle blocks, with (right) or without (left) spatial resolution reduction [17].

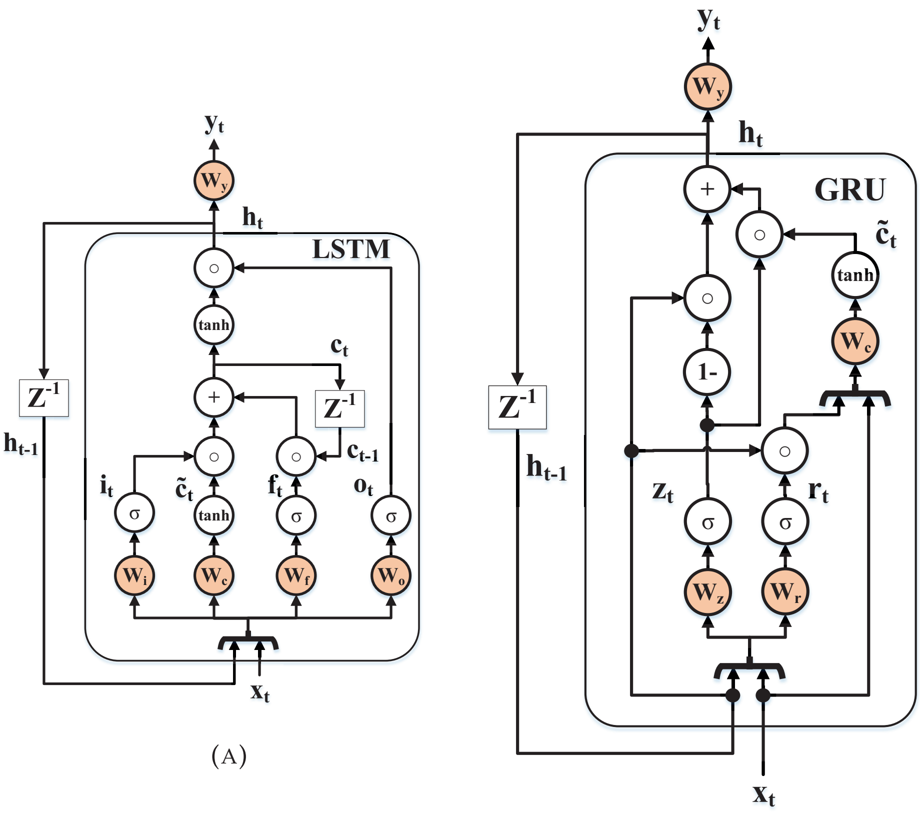
Shuffle blocks are composed of pointwise and depthwise separable convolutions but with the addition of group convolutions and channel shuffles [17, 180]. Since pointwise convolutions can suppose an expensive addition to the number of operations, and on small networks with a limited number of channels they can damage accuracy. To avoid such complexity, in [180], they apply point-wise convolutions to groups of channels. Next, to avoid information being shared across channels they apply a shuffle layer, which cross ports input channel information to other groups. With both mechanisms, they are able to maintain accuracy and deliver a notable speedup (achieving equal accuracy on ImageNet as AlexNet, they provide a speedup of x13). In Figure 2.5 (D), a detail of ShuffleNet blocks with and without spatial reduction is illustrated.

NAS built operations. Since many modern deep neural networks are built by stacking the same (or similar) base blocks to configure the main feature extractor, there have been efforts to search automatically for the architecture of efficient blocks. In works like MNASNet [246], the authors define a search space for all the possible unit operations (convolution, depthwise convolution, addition, concatenation, and many others) and search for an efficient base block. They built the overall network by stacking these base blocks together. Moreover, they can drive the overall qualities of the base block through the different objectives that the network has to attain, such as accuracy, latency, or others. Other examples of works would be FBNet [298], ProxylessNAS [245], or PC-Darts [372], among many others (usually included inside cell based NAS).

Attention mechanisms [373, 374] have constituted an important attempt to improve the learning capacity of CNNs. Among the studies devoted to attention, there have been some that have tried to maintain a trade-off between the overhead added to the network and the increase in performance. Examples of such types of work are Squeeze and Excitation (SE) modules [375], which, for example, only adds a 0.26% of GFLOPS relative increase when added to ResNet-50, or Channel Block Attention Modules (CBAM) [376], among many others like Bottleneck Attention Modules (BAM) [377] or ULSAM modules [378]. Overall, they provide an efficient manner to improve the capacity of networks with little inference overhead.

RNNs have also experienced an evolution to more efficient architectures at the operation level. In order to improve the learning capacity of vanilla RNNs and vanishing gradients, LSTM [379] was developed. However, LSTM included operations which could be reduced to avoid the extra computational complexity: it maintained two states, the input activation vector C_t and the hidden state, h_t, and performed a set of operations on each vector to add the information to the input information, totaling $4n_h^2 + 56n_h + 12$ number of parameters, where n_h is the hidden state dimension. An illustration of the network can be found in Figure 2.6 and the number of operations is detailed in Figure 2.7.

Such operational and size complexity was reduced with the development of the GRU cell [181]: it reduces the number of stored states to one and reduces the number of operations. The number of parameters is reduced to $3n_h^2 + 45n_h + 12$. The number of operations is detailed in Figure 2.7 and in Figure 2.6 an illustration of the network can be found. More details for the operation comparison between LSTM and GRU can be found at [2].

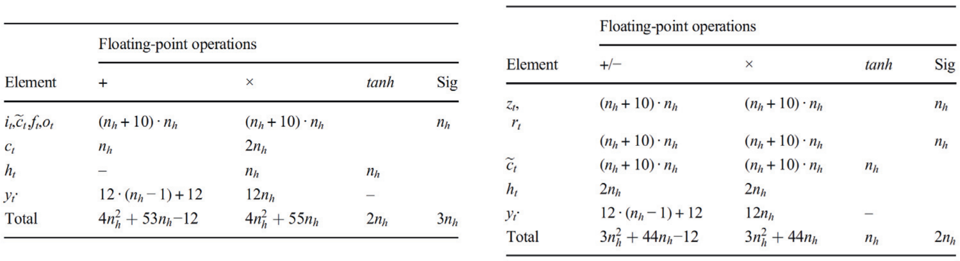


(A)

(B)

FIGURE 2.6: Illustration of the internal operations for both LSTM (left) and GRU (right). Illustration taken from [2]

.



	Floating-point operations			
Element	+	×	*tanh*	Sig
$i_t, \tilde{c}_t, f_t, o_t$	$(n_h + 10) \cdot n_h$	$(n_h + 10) \cdot n_h$		n_h
c_t	n_h	$2n_h$		
h_t	–	n_h	n_h	
y_t·	$12 \cdot (n_h - 1) + 12$	$12n_h$	–	
Total	$4n_h^2 + 53n_h - 12$	$4n_h^2 + 55n_h$	$2n_h$	$3n_h$

(A)

	Floating-point operations			
Element	+/−	×	*tanh*	Sig
z_t, r_t	$(n_h + 10) \cdot n_h$	$(n_h + 10) \cdot n_h$		n_h
	$(n_h + 10) \cdot n_h$	$(n_h + 10) \cdot n_h$		n_h
$\tilde{c}_t$	$(n_h + 10) \cdot n_h$	$(n_h + 10) \cdot n_h$	n_h	
h_t	$2n_h$	$2n_h$		
y_t·	$12 \cdot (n_h - 1) + 12$	$12n_h$	–	
Total	$3n_h^2 + 44n_h - 12$	$3n_h^2 + 44n_h$	n_h	$2n_h$

(B)

FIGURE 2.7: Number of operations for both the LSTM and GRU RNN cells as a function of the hidden state size, n_h.

Apart from this great advancement, there has been profuse research on how to optimize RNNs cells at the operation level [380, 381], but also at exploring the size limits at which RNNs can be used [382], for example in the case of embedded platforms [383, 384].

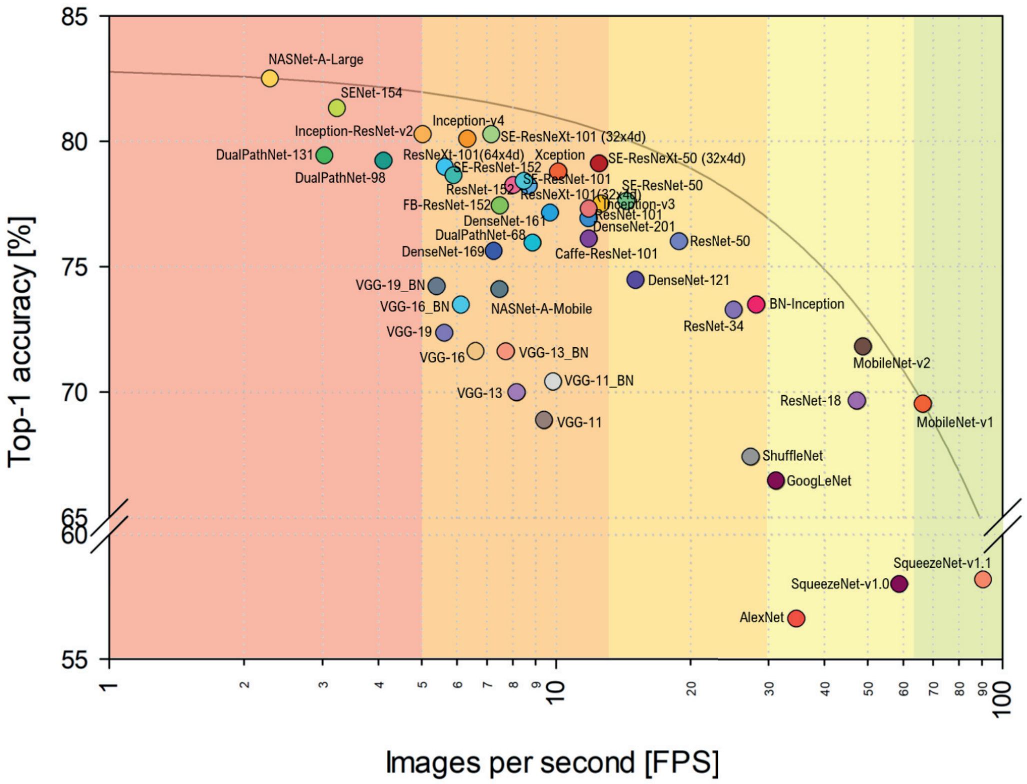


FIGURE 2.8: Latency vs Top-1 Accuracy for different models using batch size 1 on an NVIDIA Jetson TX1. Image taken from [8] with permission from the authors.

2.4.3 Discussion

Efficient operations suppose an excellent way to increase efficiency in neural networks. However, as there is no free lunch, the evaluation of the efficient operation is based on the trade-off between latency (or other measures) and accuracy. Often, such equilibrium is difficult to maintain and an improvement in both criteria is not achieved. Such situation can be seen in Figure 2.8 in some particular cases. For example, in the case of the MobileNet family, from V1 to V2 there is a substantial decrease in latency but also an increase in accuracy. The same happens with the ResNet-50 and the addition of SE modules, which improve accuracy but decrease latency. Thus, improving an architecture solely by adding efficient operations is difficult if a Pareto improvement is targeted. Improvements at the architecture level, such as those defined by the EfficientNet family [182] or DenseNets [385], suppose bigger improvements.

Nevertheless, despite this difficult trade-off, efficient operations suppose an excellent means for developing NNs specifically for some platforms. As the performance of operations is especially linked to the deployment framework and also the platform used, depending on which operations we focus the results will be suitable or not. A clear example is MobileNet blocks previously mentioned, which help develop networks that can work better on CPU-based inference, as in mobile phones or other platforms. An example can be found in [6], where IRLB blocks are used for deploying a network to a mobile CPU.

2.4.4 Conclusions

In the present section, we have reviewed some efficient operations that helped make NNs more efficient and deploy them to specific hardware. However, as discussed, maintaining a good trade-off between accuracy and latency (or other system measures) is difficult. Moreover, important improvements are often achieved not by substituting blocks for more efficient ones *ad hoc* but by redesigning the overall architecture either for the accuracy or targeting a specific platform. All in all, efficient ops help improve NNs but the important consideration is the platform and the deployment criteria.

As a side note. There are many other efficient operations not covered here such as GhostNet [386] or SqueezeNets [387]. We have not been able to include all advances in efficient operations and we have restricted to those that have had an impact on the thesis. We have also not included Transformers [112] because they have fallen out of the scope of the present thesis. However, they supposed an increase in performance for both vision and NLP tasks and also a reduction in training convergence with regards to the number of FLOPS [226]. Efficient transformers are discussed in [388].

Chapter 3

Deployment frameworks for Resource-Constrained Hardware Platforms

In the previous chapter, we have illustrated and summarized some of the techniques available for reducing the computing requirements of a model. Specifically, we have focused on quantization, pruning, NAS, and efficient operations. With such techniques or procedures, we are able to reduce the computing requirements of models so as to ease or adapt them to a resource-constrained platform.

However, that does not directly mean that we can run it on the desired platform. First of all, there has to be a deployment and inference framework that allows us to run such an optimized model by using the specific hardware resources of the platform. Secondly, the framework has to support the specific operations of the model (for example, if the model uses depthwise convolutions, the framework has to be able to use them). And finally, the framework has to be able to exploit or handle the optimizations that the model has been subjected to (a classic case is the lack of sparse operators for convolutions which reduces unstructured pruning to just a size reduction). Without a proper deployment and inference framework that is able to handle our model and its optimizations, all the developments stated in the previous chapter are worth less than wet paper (at the application level). Such is the importance of the deployment framework.

As they are intimately related to hardware, deployment frameworks are usually specialized for a specific platform and their dedicated computing instructions. Moreover, the specialization usually goes beyond broadly classified hardware (MPSoC CPUs, GPUs, MCUs, FPGA, etc.) and reaches detailed HW types, company-specific developments, or specific families of devices (e.g. different ARM architectures). Such is the case, for example, of CMSIS-NN [210] which is specialized for Cortex-M MCUs, OpenVINO, specialized to Intel hardware (CPUs, GPUs, FPGAs, and others), or the STM32Cube.AI [389], which is focused on deploying ANN on a fixed range of STM devices.

It is important to distinguish clearly the role of deployment frameworks inside the development pipeline. After training a model and optimizing it, one does end up with a file with the weights and activation functions that encode the network topology. Next, one wants to deploy it to the desired hardware platform or accelerator [18], but cannot do so because the way the model is stored is not readable and usable in such a platform. That is the role of the deployment framework, to enable the deployment of the model in a hardware platform or accelerator and the usage of specific hardware resources by the model. Such a process involves a change in the format weights are stored. However, it can also involve optimizations, such as quantization, since many platforms operate on limited data types. With regard to these optimizations, depending on the relationship between the development and the deployment framework, they can be performed in one or another. In Figure 3.1, different hardware accelerators are plotted with respect to the number of operations they can handle and the energy consumed. They are also separated with regard to their native data types and their nature as hardware platforms (chip, board, or whole system). Such complexity and differences should put into account the difficulty to adapt and bring a neural network to specialized hardware. Importantly to note is that usually, the deployment framework is thought only for inference and not for training, as well as the platforms it is intended to.

Also important to note is that, albeit as of today it might appear that there are many deployment frameworks devoted to many hardware platforms and allowing many different optimizations and operations, it has been due to an explosion in development in the past five years. Back in 2017, there were almost no public deployment frameworks to bring NNs to hardware devices, and those that were available had a limited matrix support[1]. One had to rely upon under-supported specific libraries or use a general computation library and build the support for operations. Around that time, however, an increase in the development of deployment frameworks for most platforms started. As most development frameworks had several years of development and were

---

[1] We refer to matrix support as the NN operations that the deployment framework can handle. As examples, one can check the CMSIS-NN matrix support or the Tensor RT matrix support.

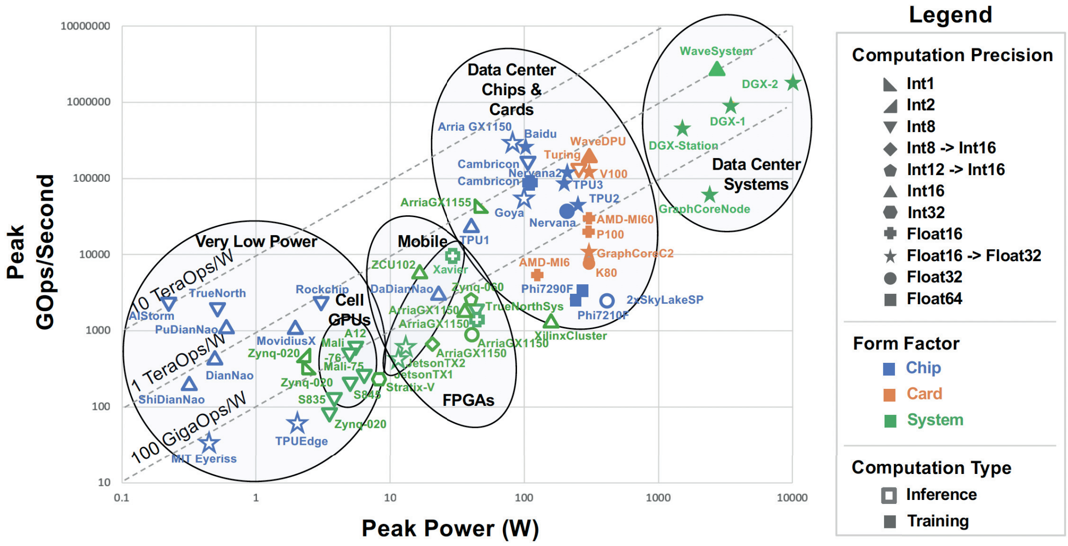


FIGURE 3.1: Number of Giga operations per second versus power consumption of publicly available AI accelerators and processors. Figure extracted from [18] with permission from the authors.

mature and stable enough, the desire to extend NNs to as many applications and circumstances as possible demanded the development of the necessary software. To pick two outstanding examples, among the many possible examples, one could state the case of Tensorflow Lite [201], which was released in 2017 and has brought ML models to platforms such as smartphones, or OpenVINO, which allows the acceleration of NNs in Intel hardware, including FPGAs, and was released in May 2018.

All in all, when this thesis began there was limited support with regard to deployment frameworks. And in that situation is where we frame part of the developments and contributions. In the first developments, I had to develop my own inference libraries for GRU or LSTM for a RISC-V CMU or had to implement my own conversion framework, as in the case of Torch2CMSIS [219] (enables format conversion and quantization of PyTorch developed NNs to Legacy CMSIS API). Later on, with the next projects, the increasing availability of deployment frameworks turned the problem to know if the model and optimizations were supported and what could be performed. In both cases, however, the importance of the deployment framework is major since it marks the successful deployment of the NN or its total lack of applicability.

In the present section, we are going to cover the deployment and inference frameworks with a focus and orientation toward devices instead of other classifications. More specifically, we aim to cover only deployment frameworks related to hardware platforms used in the industrial applications of the present thesis. That is, we are going to cover mainly frameworks for MCUs (Section 3.1), smartphones (Section 3.2), and embedded GPU boards (Section 3.3), although we will also briefly cover other devices such as FPGAs boards (Section 3.4). Then, with each framework, we are going to review which operations they cover, the optimizations that they are able to handle, and in general, the benefits and limitations that each framework has, providing orientation for the successful joint development and deployment of NNs.

3.1 Microcontroller Units (MCUs)

Deploying effectively a NN to a resource-constrained MCU can be a daunting task [229]. MCUs usually use general-purpose processing cores (single-core in most cases) which lack many high-end CPUs (from the operating system and virtual memory management to thread-level parallelism or vectorized instructions), and operate on lower frequencies (8- to 200 MHz, compared to 1 to 2 GHz from GPUs), and might not have Floating Point Unit (FPU), adding overhead to inference with floating point numbers. All these reasons can convert a fully functioning application in a non-constrained environment to not feasible projects in an MCU-based environment. For such reason, optimizations described in Chapter 2 can help deploy those applications. However, there is still one final caveat. The set of instructions[2] [390, 229], programming language, libraries available, *et cetera* are totally totally different from a NN development environment, hindering even more or directly forbidding the deployment.

To solve both problems and bridge the development of NNs and their deployment in resource-constrained MCUs, deployment frameworks have been developed. They convert the network to suitable formats, allow for running NN operations in the MCU processor and with an instruction set, and allow for certain optimizations to be used.

In the next paragraphs, we review some of the most current and common deployment frameworks for MCUs, their main characteristics, and notable points.

---

[2] The ISA (Instruction Set Architecture) is of utmost importance for running NNs in MCUs. Most of them don't have SIMD instructions, most registers are strictly 32-bit, and not all arithmetic and logic instructions are covered. Moreover, most MCUs are based on the ARM Cortex architecture, which is proprietary and thus not freely extendable, whereas the open-source RISC-V ISA support for NNs is still scarce. [229]

CMSIS-NN [391] is a deployment framework developed by ARM for Cortex-M MCUs. Specifically, more than a deployment framework is a set of efficient neural network kernels optimized for running with maximized performance and minimal memory footprint.

The library is separated in two sets of kernel functions *NNFunctions* and *NNSupportFunctions*. The first provides the interface to fundamental neural network functions such as convolutions, poolings, or activations, while the second provides utilities needed for those functions such as data type conversions or activation look-up tables. The kernels are built with support for two data types, *8-bit* or *16-bit*, though their optimizations are focused especially on 16-bit Multiply-and-Accumulate (MAC) instructions.

The first versions, denoted as CMSIS-NN Legacy API, were based on symmetric power of 2 and layer based quantization, with no offsets or layer fusion. In later developments, CMSIS-NN matched TFLite Micro data making a breaking change to the core development, but improving however quantization: now per channel, with offsets, and layer fusion. In Table 3.1 a summary of the differences can be found.

One of the main caveats of CMSIS-NN is that is a deployment library without a standardized conversion utility. Hence, to deploy the model built in the development environment (PyTorch, Tensorflow, ...) one has to follow several steps to adapt the neural network to CMSIS-NN. First, ensure manually that all layers are compatible and supported. Second, check that the data layout of the development environment matches the one in CMSIS-NN, and if not, weights have to be reordered prior to conversion. Third, compute the Q.Q[3] scheme of all the layers and the associated statistics. Fourth, compute the layer shifts. Finally, one can write the inference code with CMSIS-NN. It is due to this reason that Torch2CMSIS was developed as part of this thesis: to automatically go through all these steps and ease deployment of NNs with CMSIS. However, with the current TF Lite compatible API, all those steps are covered by TF Lite prior to obtaining the inference code.

Tensorflow Lite for Microcontrollers (TFLite Micro) is an extension of the TF Lite library (Tensorflow's library for optimized models and deployment to mobile) targeted to MCUs. It has a broad matrix support, which consists of a subset of Tensorflow operations, compared to other deployment frameworks.

---

[3] Bits assigned to the integer part, first Q, and to the decimal part, second Q, in a fixed point numeric scheme

Operation	Legacy APIs	TFL micro compliant APIs
Core loop	No input or filter offset	Input and/or filter offset
Re-quantization	Shift and saturate in one instruction. 5 cycles	Greater than 200 cycles for one output element
Quantization	Per layer quantization	Per-channel quantization
Output offset	No	Per-layer output offset
Fused Activation	No	Yes

TABLE 3.1: Differences between the CMSIS-NN Legacy API quantization and the new TF Lite compatible scheme. Extracted from the CMSIS-NN Github page.

It is built as a generic tool, targeting any 32-bit MCU, making it an extremely portable framework with an appropriately sized memory footprint (the core runtime fits in 16 KB, while uTensor fits on 2 KB). However, this generality makes the lack of connections to specific hardware vendor environments a difficult step for deployment. Moreover, when there are no automatic tools for generating and deploying the inference code.

Interestingly, it supports floating point *binary32* format and 8-bit fixed point integer computations. As previously stated, and reflected in Table 3.1, it supports per-channel quantization, it has input, filter, and output offsets, and it includes layer fusion. It does not support 16-bit integers. Similar to the new CMCIS-NN API, it can make use of the specialized kernels for 8-bit integer quantization to have accelerated inference. The principal caveat of the whole framework, however, is that the network topology is interpreted at runtime instead of being statically compiled, adding a huge drawback for optimizations and increasing memory footprint at runtime.

Overall, it is a really flexible and open framework that has a broader coverage of operations and optimizations than other frameworks, as well as well-established support.

uTensor is a C++ opensource machine learning deployment framework covering Tensorflow as development platforms and models, and Arm MCUs as a hardware platform. It consists of a runtime library for inference, with which the user can build the inferencing code, and a translation tool to convert the model to an MCU-compatible version. It has additional libraries like uTensor_cgen, which automatically builds inference code from a *.pb* () Tensorflow Lite file to *.cpp* and *.hpp* files.

As of the writing of this thesis, uTensor is well maintained and has current support. However, one of the main downsides is the small matrix support, which might be limited for complex networks but more than sufficient for efficient and small classification networks. As a positive point, it supports per-channel and per-tensor symmetric quantization schemes based on the Tensorflow Lite quantization specification.

MicroTVM is an extension of the high-performance compiler for neural networks Apache TVM [392] which targets bare metal MCUs as deployment platforms. Although it is currently only tested on Arm targets, it supports deployment to other architectures, such as RISC-V. Still under development, examples are found supporting networks based on Tensorflow Lite and for classification but the matrix support is unclear, as well as its support for other frameworks. However, it still is a promising possible alternative to CMSIS-NN for low-level MCU acceleration, delivering in some cases performances far superior to it.

Other frameworks. With the raise of TinyML [393], the number of available frameworks for deploying NNs in MCUs has flourished. As it is not our intention to give an exhaustive list and description of all of them, we mention here some of them as well as their main characteristics and where to find more information.

STM32Cube.AI is an example of a company-based (STMicroelectronics) deployment framework. It targets the family of 32-bit MCUs from the same company, and it supports most of the major development frameworks, either directly (Keras and Tensorflow) or through ONNX [394]. It comes with great matrix support and seamless integration with other STM development tools, easing the deployment process. It supports 8-bit integer and 32-bit float data types and operations. However, the fact that is limited to STM platforms and that the base code isn't extendable are major drawbacks.

emlearn [395] is a deployment framework for machine learning systems trained with Sciki-Learn or Keras targeting any device with a C99 compiler available.

MicroAi [229] is a deployment framework supporting any 32-bit MCU independently of the vendor and for Keras developed models. They support, aside from 8-bit integer and 32 float data types, 16-bit integer data, and operations. They use fixed-point quantization $Q_m.n$, instead of offset and scale-based quantization. Despite its publication, there seems to be no currently available code for the framework.

Other inteteresting frameworks include *Gravity* [396], *FANN-on-MCU* [397], or *Micro-LM* [398].

3.2 Mobile Platforms

Although smartphones have recently grown spectacularly in hardware capabilities, most DL frameworks cannot be used directly for inference in the mobile platform for different reasons (third-party library dependency, lack of software-hardware link, among many others [25]). Such a situation, and the need to have hardware-specific optimizations, have determined the need for specific mobile-devoted deployment frameworks. In parallel, the growth in mobile phone usage and the increase in available data that has come with it has also motivated the development of such frameworks to ease the development of applications and systems.

Most frameworks are written in C++ [26] to enable high performance and access to current mobile hardware accelerators and processors (mainly CPU, GPU, and DSP). Most of them are also developed by major technology companies such as Google (TF Lite), Tencent (NCNN), or Xiaomi (MACE) (although there are some exceptions with universities or start-ups), and are also majorly open-source to promote community contributions and user accessibility. They mostly have a low memory footprint, being significantly lower for frameworks only targeting mobile inference [26]. An important point to comment is that many of these deployment frameworks have profited from the development of hardware accelerators thanks to important acceleration APIs such as XNNPACK, ARM Compute Library, QNNPACK, OneAPI, GEMMLOWP [227], Intel OneDNN[399], among many others.

In the next paragraphs, the characteristics of the most common mobile deployment frameworks are described.

Tensorflow Lite (TF Lite) . One of the first deployment frameworks for mobile platforms was the TF Mobile platform, released in 2015, which allowed deploying DL models to Android smartphones but with any hardware acceleration (CPU inference only). Later on, in 2017, it was re-implemented as Tensorflow Lite, which now included support for many hardware accelerators through the android NNAPI. As hardware and mobile vendors have continuously added more support for NNAPI, this has led to a good improvement of TFLite, which has culminated with the introduction of TF Lite Delegates.

TF Lite delegates are a software abstraction layer that enables hardware acceleration on mobile platforms by using on-device accelerators such as GPU [206] or DSP (such as the Qualcomm Hexagon) without the need to write explicit code for each accelerator. They represent a major advantage since they help DL developers abstract from hardware-specific code and allow hardware vendors to write their own delegates for their hardware.

Overall, TFLite is a lightweight and fast framework that allows the deployment of Tensorflow models to mainly Android but also iOS platforms. It has a good set of development tools, which ease the deployment of models in Android platforms, such as the integration with Android studio. Built around the TFLite Library, it includes the TF Lite Task library, with out-of-the-box integration with many DL tasks, and the TF Lite Support Library, which helps with other functionalities, such as data pre- and post-processing. It has the support of many optimizations provided by Tensorflow, such as quantization or pruning (albeit with limitations for sparse kernels [226]). Moreover, it has APIs for many programming languages and support for many hardware architectures. However, at some point, if the developer wants to carry out specific tasks, deep knowledge of Android is required to go back and forth from the Java Native Interface (JNI).

PyTorch Mobile is the framework developed by Facebook Inc, after Caffe2, and connected to the PyTorch development framework, which eases the connection between both settings: development and production. It supports Android and iOs platforms. It has beta support for GPU and DSP through third-party libraries (Vulkan and Google NNAPI, respectively). It has optimized inference with floats through the XNNPACK [363] library (floating point optimized operators for ARM, WebAssembly, and x86 platforms) and with integers 8 quantization trough QNNPACK. Notably, its connection with the development library makes it really easy to export a model through the *just in time* (JIT) and related tools that can be consumed in the deployment framework. Contrarily, the access to some efficient procedures or functionalities in the Android environment is not so seamless and requires interaction with C++ code through the JNI, hindering the building of the whole application.

MACE ((Mobile AI Compute Engine) is the deployment and inference framework developed by the company Xiaomi for mobile heterogeneous computing platforms. It supports hardware acceleration on GPUs and DSPs (through the Qualcomm Hexagon SDK). Written in C++, it allows to consumption of

Name	Company	Android	IOS	CPU	GPU	DSP	Time	Open source	Training
TensorFlow lite	Google	✓	✓	✓	✓		2017	✓	
PyTorch Mobile*	Facebook	✓	✓	✓	✓	✓	2017	✓	✓
core ML2	Apple		✓	✓	✓		2018		
NCNN	Tencent	✓	✓	✓			2017	✓	
Feather cnn	Tencent	✓	✓	✓				✓	✓
SNPE	Qualcomm	✓		✓	✓	✓		✓	
MACE	Xiaomi	✓	✓	✓	✓	✓	2018	✓	
Paddle model	Baidu	✓	✓	✓	✓	✓	2017	✓	

TABLE 3.2: An overview of the popular deep learning frameworks for mobile terminals. Inspired in [25]. * Previously contained inside Caffe2. We have covered only the major frameworks. A more broad list is included in [26].

Tensorflow, Keras, or ONNX models directly. It has a good matrix support and it allows quantization in CPUs and DSPs. It also includes benchmarking and debugging tools, easing the continuous development of models and their checks.

Core ML is the deployment framework developed by Apple for their mobile OS, iOS. It allows hardware acceleration only on GPUs and the Apple Neural Engine and it is not open source hindering its extension and usage. Apart from the inference run-time library, it provides a set of tools (Core ML Tools) for adapting models from common training frameworks, although it allows also building and training models with the Create ML application. To help deploy models and develop specific applications it has a set of libraries supporting common tasks in different DL fields, such as Computer Vision (Vision library) or NLP.

NCNN is Tencent's deployment framework for many platforms, including mobile ones, Android and iOS. Purely written in C++ does not rely on third-party libraries for acceleration, although it only supports GPU as hardware accelerators. It can consume models from major development frameworks and it allows for quantization in integer-8 and half float precision. It does not have a well-documented API.

DL deployment is a currently ongoing research and development topic. We have not intended to cover all the frameworks but to provide insight into some of the major frameworks available and the current situation. Important frameworks not covered are the Qualcomm framework Snapdragon Neural Processing Engine (SNPE), the Alibaba's Mobile Neural Network (MNN) [400], PaddleLite from the PaddlePaddle development framework, or Tencent's FeatherCNN, among others. In Table 3.2, an overview of the most popular mobile deployment frameworks for DL is given. To access a more extended review on the topic, we suggest the reader the following surveys or reviews ([401, 25, 402, 26]). An important note to mention is that, although DL frameworks for mobile platforms allow for easier deployment and integration, development frameworks can also be, in some cases and given the hardware capabilities of smartphones, in the production platform, as detailed in [26].

3.3 Embedded GPU boards

GPUs, since the explosion of deep learning, have been the preferred solution for training. Moreover, for computing-intensive tasks, such as image super-resolution, and other image tasks, they have also been the preferred inference platform.

There are different GPU vendors, and although the market of X86-compatible embedded boards has different players, such as Intel GPUs, in the case of deep learning, the predominant player is NVIDIA. Due to the specialized libraries for deep learning, like CUDA or cuDNN, and its usage by major development frameworks it is the *de facto* choice both for training and inference.

There are some applications that for privacy or connection reasons need to perform the task in the device without sending the information, and, in parallel, the task is too heavy to be deployed to the CPU. In such cases, embedded GPU boards or modules, like the Jetson product family, provide good performance in restricted environments.

Albeit not common, there are other alternatives for inference coming from Intel (OpenVINO [215]) and AMD (mROC [403]), but they are not targeted for embedded GPU boards.

NVIDIA TensorRT is NVIDIA inference framework targeting general GPUs and the main deployment framework for embedded GPU boards. It is written in C++ but has also an API for Python. It has a broad matrix support and it has extensions and tools for consuming many development frameworks (like Torch-TRT or Tensorflow-TRT). It has support for inference with 32-bit floating point data, half float, and integer 8 quantization enabling easy access to quantized model inference. Moreover, it allows running sparse kernels on the Ampere family of GPUS with Apex. Moreover, it allows fine-grained control of inference (serial or streamed), data movements, and pre/post-processing in GPUs, due to its linkage to CUDA and other GPU-specific libraries. Finally, it also has support to work with dynamic shapes, high-performance frameworks, like Deep Learning Accelerator (DLA), working with conditionals, and a set of tools for profiling and benchmarking (like NVIDIA Nsight Systems or trtexec). The only main drawback of the framework is that being built with C++ and specialized GPU libraries, makes the debugging a rather not accessible task.

3.4 Other platforms

There are many other platforms that require deployment frameworks to access their hardware capabilities. We do not intend to cover all of them. Thus, we will provide hints to resources and name just a few of the most common platforms, and associate deployment frameworks for completion reasons.

Field Programmable Arrays (FPGAs) offer high computational power, low consumption, and flexibility. However, it comes at a cost: difficulty to program, which starts with the need for specialized hardware languages (Verilog/VHDL) and continues with the need for specific tools provided by suppliers to design the solution.

To solve such complex problems, FPGA providers have provided deployment frameworks that can consume a model with a standard interface, relying upon the optimization of the core FPGA library. Examples of such deployment frameworks are OpenVINO by Intel [215], LeFLow [404], VITIS AI from Xilinx, among many others.

Application Specific Integrated Circuits (ASICs). With the exponential growth of AI usage in many applications and systems, the need to optimize hardware utilization for deep learning operations has also grown. ASICs, which are integrated circuits customized for a particular use, provide good performance and energy used at the cost of reduced flexibility. However, since deep learning is mostly based on a set of specific operations (matrix multiplication, nonlinearities, etc) they stand out as an appropriate choice.

Currently, they have many names depending on the hardware provider (Tensor Processing Unit (TPU), Neural Processing Unit (NPU), Visual Processing

Unit (VPU)) but all of them are virtual components (IPs) oriented to their integration in Application Specific Integrated Circuits (ASIC) and specialized for deep learning or some subset of tasks of it.

Regarding deployment frameworks, some ASICs can directly consume development frameworks as deployment frameworks, as in the case of Google TPUs, which can be used from Tensorflow. Others, such as Myriad X VPU from Intel, require a specific deployment framework in this case OpenVINO. Usually, the use of a deployment framework depends on the hardware company that produces the ASIC, which can link their deployment tools to existing frameworks, as in the case of Cerebras or Graphcore [405]. A good summary of the available ASICs for deep learning can be found in Medium - Hardware for Deep Learning. Part 4 ASIC by Grigory Sapunov.

Chapter 4

Capacitive Sensing based Gesture Recognition on MCUs

In the present chapter, we present the first application environment and project: gesture recognition with capacitive sensors deployed to low-cost MCUs.

Gesture recognition with capacitive sensors is a well-studied, researched, and industrialized topic. From hard-coded heuristic algorithms [406] to computing non-intensive machine learning algorithms [407], solutions, albeit limited in gesture richness, had already been incorporated into low resource computing environments. Nevertheless, the success of NNs with tasks such as image classification or gesture recognition itself has drawn attention to algorithms of the like due to the new possibilities offered. The main constraint has been the perception that they were too computing intensive for their usage in resource-constrained devices. Hence, a research stream oriented towards diminishing the resource requirements of such models or studying their deployment in resource-constrained devices was initiated. Both at the optimization level and the deployment framework, advances have been made from an almost non-explored situation. In late 2018, there were almost no deployment or conversion frameworks for NNs toward MCUs and the tiny ML [393] trend had just started. Although optimization methods had already been studied (quantization, pruning, etc) advances were still to be made, and the focus on the hardware platform was a novel research direction. In such an environment begins the first step of the thesis.

In this chapter, we study the implementation, optimization, and deployment of tiny RNNs in MCUs devoted to gesture recognition in capacitive sensors: from developing small enough RNNs so as they fit in a low-end MCU [2, 3], or improving NAS algorithms [4, 218] for adapting such networks to specific hardware requirements, to automatizing the quantization and conversion of such networks to Cortex based MCUs [5, 219].

The chapter is organized as follows. First, in Section 4.1, the topic of gesture recognition with capacitive sensing and NNs on low-cost microcontrollers is introduced. Next, in Section 4.2, we detail the first experiment with regards to 1d gesture recognition with tiny RNNs. Section 4.3, describes the second experiment, which is devoted to the extension and research of a NAS algorithm. Section 4.4 illustrates the development of an unsupervised automatic approach for developing RNNs and CNNs for gesture recognition on low-cost MCUs. Finally, Section 4.5 draws the conclusions and possible next steps.

4.1 Introduction

Gesture recognition has advanced progressively in the last two decades thanks to both the improvements in state-of-the-art models and the appearance and enhancement of new sensing methods [408]. New technologies, such as KinectTM, together with the decrease in the price of classical sensing methods, for example, depth cameras, helped to create a vast amount of data, which allowed the development of new solutions.

These improvements have spread the usage of gesture or action recognition in a wide range of applications: from security to entertainment. However, the sensing method and the implementation strategy vary depending on the requirements of the application. Camera-based information retrieval tends to capture contextual data that may or may not be necessary. On the other hand and as an alternative, capacitive-based methods tend to be cheaper and do not suffer from invasiveness issues but require complex information retrieval.

In the automotive environment, neither invasiveness nor contextual information is desired for gesture recognition. Invasive sensors might affect the correct driving task and a sensing method that captures contextual information might detect unwanted events. Capacitive sensors suppose an alternative solution for those two problems. Although their first applications were devoted to fluid or particle detection [409, 410], their low cost and short-range properties have spread their usage to many other areas, including automotive applications [411, 412], where they have received a great industrial interest during last decades. When compared with mechanical buttons, the lack of moving parts reduce wear and tear [413], contributing to much better durability and reliability properties. Moreover, its design flexibility allows it to include them in a wide range of products and shapes while enabling cost reduction and budget tightening. Their sensing mechanism is based on the measurement of the value of a variable capacitor affected by the presence or proximity of the object to detect, known as the target. The more the target decreases its distance to the sensor, the better the system is able to detect it. As its name (proximity) implies, this allows to use them as distance detectors (such as in [414]) and non-contact and contact gesture recognition (such as in [406]).

In any case, the distance between the target object is indirectly computed by reading the capacitance value. For low-cost sensors, the tolerances of the building processes, the differences on PCB manufacturing and the different placement of mechanical elements require that every design is calibrated to make sure that the reading outputs are adjusted to an expected output range. Moreover, the diverse capacitive characteristics when approaching different parts of the body (fingers, hand, etc.) and their potion pattern can make it challenging for classic digital signal processing (DSP) event detection algorithms. Figure 4.1 shows some example readings of an individual proximity sensor.

Additionally, capacitive sensors are being progressively introduced in much human-machine interaction (HMI) systems of consumer devices like smartphones, smart watches, wearable devices and fabrics, and many others. As users are more used to touch interfaces, finger or hand gesture control systems are increasingly demanded by premium line car-makers and truck-makers like BMW [415] and MAN [416]. However, the disparities in the mechanical integration of the sensors on different product models increase the difficulty to use a single gesture recognition system for all of them. A common solution is to recalibrate the algorithm parameters for each new product design, but this process requires some investment. Hence, a global gesture recognition system that could embrace such variability could be of prominent benefit to the manufacturer.

RNNs for Gesture Recognition in MCUs Such pursuit has been traditionally approached by gesture classification methods in a two-step process: processing and classification. In the first, data is transformed appropriately to a feature vector at every instant. In the second, this vector is used to classify the gesture by means of a model such as Support Vector Machine, Random Forest, or Hidden Markov Model that take into account temporal evolution. The recent success of NNs, and more specifically, CNNs [85] or RNNs, in tasks such

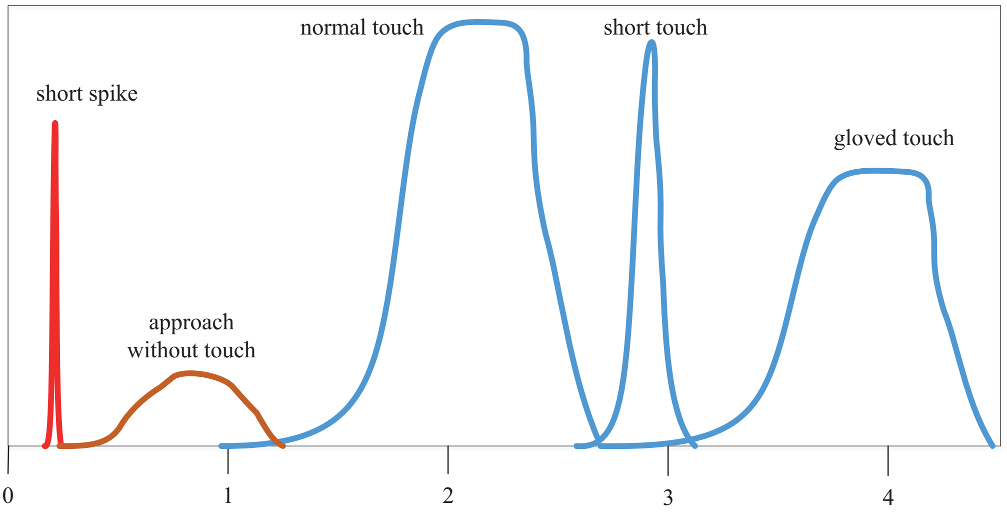


FIGURE 4.1: Examples of normalized capacitance reading to various events over time.

as image classification, has spread its use in the gesture recognition pipelines.

The signals captured by capacitive sensors are complex in terms of spatial and temporal relations [417], thus, RNNs, which are seamlessly suited for temporally structured data, seems a good alternative to implementing such flexible gesture recognition systems. However, despite their expected good performance, they are perceived, as CNNs, too computing-intensive to be embedded in low-cost microcontrollers. This situation is changing with the last advances in optimization and quantization of NNs for resource-limited devices [150].

NAS for Gesture Recognition in MCUs In order to make neural networks more efficient and also to be able to embed them into resource-constrained devices several strategies have been developed by researchers, e.g. pruning [158, 157], quantization [150], efficient operations [15], and approximate computing [418]. Along with these improvements, new software tools to port networks to resource-constrained environments have been developed [201, 419, 216, 420, 211].

However, all these previous optimization procedures are focused on modifying an already trained structure and avoiding huge drops in performance;

the model architecture is not modified. NAS is a different approach: it focuses on finding the best performing architectures by modifying the network itself, usually by means of an evolutionary or genetic algorithm [250]. Lately, NAS algorithms have focused their attention on developing architectures that are both well-performing but constrained in resource consumption and usage [189].

Although delivering state-of-the-art performance and efficiency, these methods still require high computational resources. From these three problems, hyper-parameter optimization, NAS, and making neural networks (NNs) resource-efficient, stemmed off a series of efforts centered in providing NAS solutions for resource-constrained environments [188, 421, 35, 422, 423]. Nonetheless, most of these efforts centered their attention on CNNs and do not include RNNs. And if they include them [290], the focus is not centered on minimizing the memory and size footprint of the architectures found. This lack, altogether with the explosion of new technologies and environments enabling or demanding gesture recognition (virtual reality, smart cars, depth image sensors, wearables, among others), establishes an excellent opportunity for automatizing the building of gesture recognition networks for embedded environments.

Unsupervised 2D Gesture Recognition As we have seen NNs have achieved outstanding performance in tasks such as image classification or speech translation, as well as in our case, gesture recognition with capacitive sensors. There have also been methods developed to reduce their computational requirements for them to be embedded, specifically, in resource-constrained MCUs. However, in most cases, efforts have been directed towards supervised learning. In such cases, labeling is required, which can induce, among other factors, bias in the model and undermine its real-world use [424]. To partially solve this issue, unsupervised methods, such as autoencoders [425, 426, 427] or clustering [428, 429], focus only on the information provided by the data to perform the task and avoid annotation and labeling, both error-prone activities which could induce noise and bias. Nevertheless, the bias or noise are not completely removed since the data itself could still be biased and noisy. In the end, unsupervised methods do not only liberate from the task and effects of labeling but also allow for more freedom in the learning of patterns, by not being constrained to a specific purpose task. Hence, they enable the possibility of pattern discovery and identification outside the constraints of the specific task purpose.

Additionally, and as a parallel matter, we have only explored 1D interaction in capacitive sensing. Due to its short-range nature, exploring 3D interactions is challenging unless increasing notably the size of the sensor. 2D interactions, however, come naturally with capacitive sensors, and, indeed, have been the *de facto* technology for most HMI applications, like smartphone or machine screens [430, 431] that include a touch sensor (often capacitive to get multi-touch functionality) on top of a display. As stated previously, a global gesture recognition system that could embrace the variability of capacitive sensors could be of prominent benefit to the manufacturer. Moreover, in the case of capacitive sensing with 2D interactions, there has been room for improvement of both the error rates [432] and the methods chosen for detecting the increasing complex gestures [433]. This fact, altogether with the restrictions on computing resources of HMI in automotive environments, presents a challenge where the optimization of NNs and their deployment in resource-constrained devices stand out as a prominent opportunity for advancement.

4.1.1 Contributions

The contributions of the present chapter are divided into a three-step experimental process. In them we progressively investigate the development and improvement of RNN and CNN models, and especially their optimization, to prepare them for their usage in a low-cost microcontroller platform devoted to gesture recognition systems with a capacitive sensor.

First, we research the usage of RNNs for a capacitive proximity sensor application in the automotive domain (following the work in [434]). Our proposal is to use uncalibrated raw data to detect touch gestures for the control of car infotainment and comfort systems using low-cost microcontrollers. Our objective in this section is then two-fold. First, the proposal of an RNN is suitable for microcontroller embedding which is able to capture complex spatial and temporal gestures, such as sliding or approximation. And, second, its optimization to obtain maximum performance using Bayesian Optimization.

Second, we implement and modify SpArSe [188], a NAS oriented to microcontroller deployment, while extending it to RNNs and combinations with CNNs. We also introduce latency as a new objective for time-critical applications. Thus, we can automatically develop networks that can be targeted to gesture recognition and which are especially suited for embedded devices and time-constrained tasks. We apply our implementation in a popular gesture classification dataset, the Corpus of Social Touch (CoST [435]), and obtain better results both in terms of accuracy, model size, working memory, and, for the first time, latency when compared to manually developed architectures. In addition, we modify the search procedure by introducing a low-cost fidelity by running fewer epochs at the beginning and increasing them as the program advances. Hence, this forces the search algorithm to be more exploratory at the beginning and employ more time with Pareto solutions by the end of the procedure, shortening the total time needed for finding optimal solutions. The code is available in https://github.com/BCJuan/SpArSeMod. An important additional outcome is the implementation of an special kernel [436] in a well-known open source Gaussian Process library, GPyTorch [437].

Third, we explore and develop an unsupervised solution for 2D gesture cognition with a bi-axial planar capacitive sensor. We present an auto-encoder neural network that, in junction with K-Means (KM) and optimized through NAS, can perform unsupervised classification in a low-resource microcontroller for recognizing gestures. Although there have been similar works with other models [438] or with other topics [439], to the best of our knowledge, this is the first work that presents a fully embedded and unsupervised solution for gesture recognition using neural networks and capacitive sensing. An important outcome is a creation and open-sourcing of a library for automatic quantization of PyTorch networks and their conversion to CMSIS-NN framework, Torch2CMSIS [219].

4.2 1D Gesture Recognition

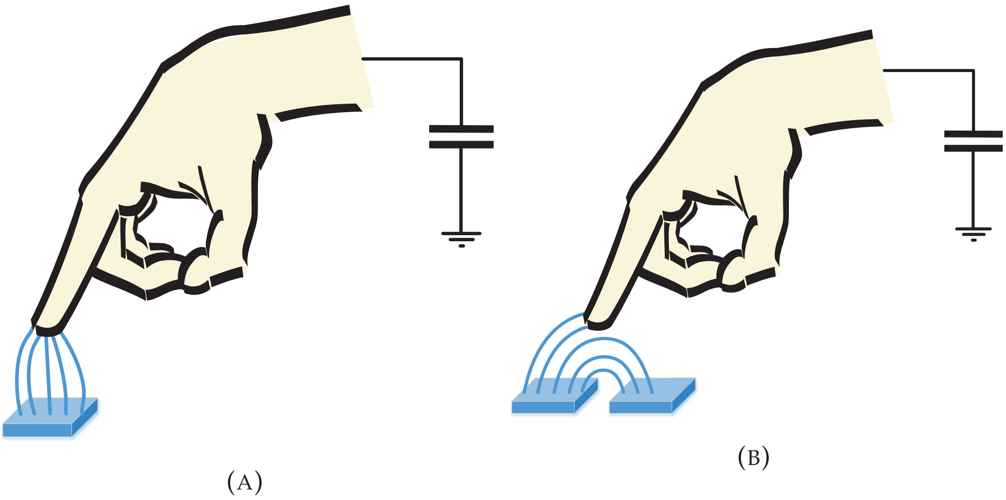
4.2.1 Sensing Mechanism and context

Proximity Sensing through capacitance measurement

The capacitance of a parallel plate capacitor is based on Equation 4.1, where A is the area of the plates, ϵ_0 is the dielectric constant for free space, ϵ_r is the relative dielectric constant of the material separating the plates, and d is the distance between the plates.

$$C = \frac{\epsilon_0 \cdot \epsilon_r \cdot A}{d} \tag{4.1}$$

Capacitive touch sensors are often based on the mechanical deformation that


(A)

(B)

FIGURE 4.2: Different proximity sensor principles. (A) Self-capacitance sensing. (B) Mutual capacitance sensing.

the target produces on some of the capacitor plates. This reduces d and consequently increases C. On the other hand, proximity sensors work on two possible different approaches (as detailed in [440] and depicted in Figure 4.2): the self-capacitance approach and the mutual capacitance approach. Grosse does a finer analysis in [431] by describing a taxonomy of the possible configurations.

In any case, for most proximity sensor designs, it is assumed that the sensing surface is not deformed. In the self-capacitance approach, the sensor has only one electrode, and the target acts as the other electrode of the capacitor. Figure 4.2a illustrates the operation. The measured capacitance is the one created between the electrode and the ground of the sensor system. The value of the obtained capacitance depends on the distance to the target (the closer, the higher), but also on the area of the plates and its relative dielectric constant ϵ_r. Animal tissues ϵ_r can vary significantly depending on the type of tissues but it is above 50 (see [441]) which is higher than air (that has a value slightly higher than 1).

Mutual capacitance proximity sensors are often implemented as planar capacitors, due to their easy implementation on printed circuit board (PCB) technologies. In this case, there are two lateral plates, as illustrated in Figure 4.2b. The electric field is bent between the two plates. When the target object approaches the sensor electrodes, it reduces the electric field between them.

Self-capacitance sensing has typically higher distance sensitivity [442] and it is preferred for low-cost devices since it uses fewer electrodes and fewer pins from the microcontroller to implement the readout circuit. On the other hand, mutual capacitance sensing can achieve higher granularity in close proximity to target positions, which makes it adequate for 2D touch panels. Mutual capacitance sensors also reduce the influence of parasitic capacitances and allow the presence of water, which are both problematic in self-capacitance sensing.

Mutual capacitance sensing is used in several works (like [443]). In the automotive domain, BMW research also presents in [444] a mutual capacitance sensor with some similarities to the one presented here. Nevertheless, in this work, we use a self-capacitance approach to reduce the system cost.

The capacitance variation is usually measured by analyzing some aspect of the temporal response to a signal injected into the system. A good review of different methods is done in [445]. Due to the inverse proportionally with distance d, proximity sensors require a good resolution of the C readings to estimate its value and perform gesture recognition. The process typically involves what is known as capacitance-to-digital conversion.

Some methods, to do this conversion (such as [446]), is based on the use of operational amplifiers (OpAmps) and require a number of external electronic components. But, in low-cost solutions, they are avoided if possible. Another option is the use of the switching capacitor principle [447, 448], by which a capacitor is equivalent to a resistor with a resistance value related to the unknown capacitance. Some microcontroller manufacturers add that kind of circuit to microcontroller lines targeting capacitive sensing application domains.

However, for a low-cost implementation, a solution based on general-purpose microcontrollers is preferred. The charge transfer principle [449, 450] is especially adequate for its implementation in low-cost microcontrollers. Modern microcontrollers include a number of general-purpose input-output ports (GPIOs). Although most GPIOs are digital, some of them are also connected to analog-to-digital converters through analog multiplexors. Thus, some pins can drive VCC or GND (when working as digital output pins), provide a high impedance (or floating) value (when working as digital inputs) and work as analog inputs. In other words, the same simple microcontroller pins can be used to charge, discharge, and measure the decay time of the sensing capacitor [451]. The challenge, for these types of measurement techniques, is to have a good value resolution and quick response time.

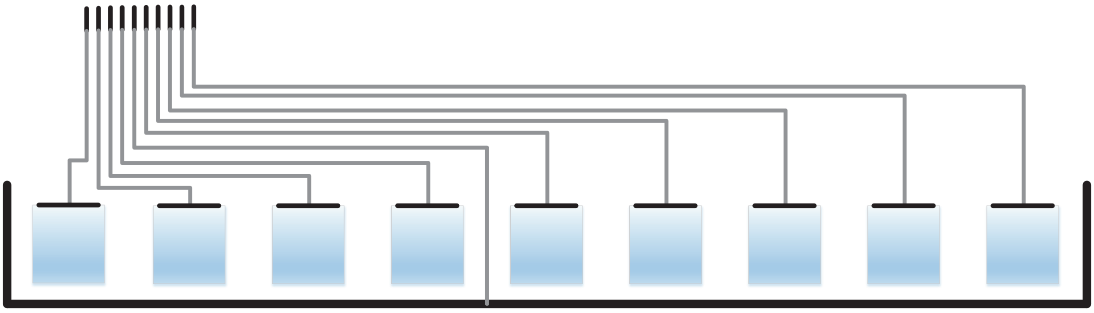


FIGURE 4.3: Capacitive functional foil for touch sensing. Sensitized areas are depicted as light blue rectangles.

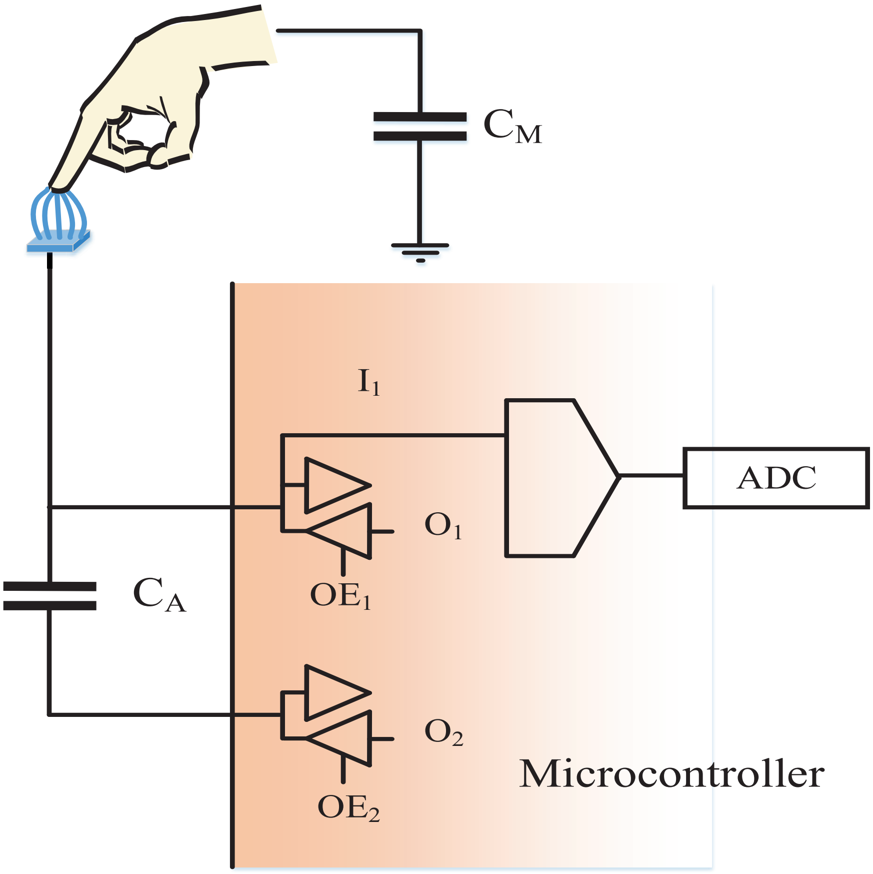


FIGURE 4.4: Capacitive sensor readout system.

Sensor

The proposed sensor is a multi-touch sensor to be integrated into a car dashboard. The intended sensitive distance is limited to less than 5 cm to reduce the unintended activation of the functions it controls, so the self-capacitance design is selected due to its relatively high range and lower cost.

The sensors are included in a capacitive foil embedded into a plastic module specially designed for touch recognition. Both the capacitive foil and the plastic case have special regions where touch events are expected. The structure of the capacitive functional foil is shown in Figure 4.3, and basically consists of rectangular electrodes connected by a wire bus to an external connector.

The sensor foil is connected to a microcontroller that analyzes the sensed signals and determines the appropriate actions to perform. Capacitance measurement is performed by a charge transfer-based approach which is detailed in the patent [452].

The readout system is depicted in Figure 4.4. To perform the measurement, two capacitors are used: the measuring capacitor, CM, which value is unknown, and the integration capacitor, CA, which has a greater capacitance than the expected CM. Two pins (1) and (2) of the microcontroller are used to control the measurement sequence. VCC and GND can be forced by setting the output value of the pin to 1 or 0 and activating the output enable signal (OE) of the pin. On the other hand, if OE is not active, the pin is in high-impedance mode acting as an analog input.

First, CA is discharged by setting both pins to GND. Then, a sequence of CM charging and charge transfer to CA is done for a number of times. CM charging is done by setting pin 2 to high impedance and pin 1 to VCC. The charge of CM is transferred to CA when pin 1 is in high impedance and pin 2 is connected to GND.

After a number of N charge-transfer cycles, the analog value of pin 2 is read through a 10bit ADC and used to compute the capacitance reading. The voltage value is determined by Equation 4.3

$$V_A(0) = 0 \tag{4.2}$$

$$V_A(N) = \sum_{i=1}^{N} \frac{C_M}{C_M + C_A}(V_{CC} - V_A(i-1)) \tag{4.3}$$

We use a fixed number of cycles N. The value for N is determined experimentally depending on the application, as it is a trade-off between latency and prediction accuracy. The overall process is performed for all seven sensors in a time multiplexed fashion. The final acquired data stream consists of 10 channels (9 electrodes plus the shield) of 10-bit signals at a 356 Hz sampling rate.

An alternative method proposed in [453] avoids using the ADC. Instead, they propose to keep repeating the integration cycle until the voltage of CM is higher than the threshold voltage of digital 1 (VIH). Although this method could reduce the cost by eliminating the need for ADCs, it is less deterministic and can be slower due to the variable number of integration cycles required.

4.2.2 Gesture definition and data acquisition

The designed sensor is built having functional flexibility in mind so that it can be used for button arrays or level controllers. In the first case, the expected interaction is based on taping while in the second case finger swipes would be used.

So, our interactive system is designed to recognize two different types of gestures: taps and swipes. Since we have nine sensitive areas, we have the same number of possible tap gesture events plus right and left swipe gestures.

Regarding the functionality of the system, a tap gesture on a specific electrode of the sensor unit is typically used to activate or deactivate a specific option or function of the product. On the other hand, a swipe gesture would be used to relative increase or decrease some value. For all gestures, we expect that a finger has physical contact with the surface.

Other activations of the sensor that do not involve a meaningful previously described event are ignored. This includes approaches without touch, inter-button touches, and multi-button touches.

The duration of the events is undetermined. Tap events have a lower duration bound, but can be very long. Swipe gestures duration has a lower bound limited by the speed of finger movement and an upper bound by the combination of the sensitivity and sampling frequency of the system.

The input data stream X can be modeled as a sequence of n samples from 10 sensors (see Equation 4.4). Although the readings of the ADC give integer values, the process detailed in Equation 4.3 results in floating point values.

$$X = \{x_1, ..., x_n\} | x_i \in \mathbb{R}^{10} \tag{4.4}$$

Event-based gesture recognition can be formalized as a function that maps the acquired data stream X into a sequence of m recognized gestures (as shown in Equation 4.5).

$$\mathcal{F} : X \rightarrow \{G_1, ..., G_m\} \tag{4.5}$$

Each gesture G_i event can be defined as a set of features that uniquely identify them. Those features are the gesture class, the starting time, and the gesture duration (Equation 4.6).

$$G_i = \{t_0, \Delta t, c\} \tag{4.6}$$

In our case, the possible considered gesture classes c, also known as gesture vocabulary, consist of tap gestures for each of the 9 sensors and swiping to left and right directions (also known as slides). In Equation 4.7, T_i denotes a touch gesture in sensor i. S_l denotes a finger gesture moving from right to left, and S_r denotes a finger gesture moving from left to right.

$$c \in \{T_0, ..., T_8, S_l, S_r\} \tag{4.7}$$

Since our system is based on a 1D sensor, the gesture vocabulary is simpler than similar works working on 2D sensors (like [454]), which also include more complex gestures like circle movements, rotations, etc.

Offline gesture recognition systems can analyze the whole X stream to segment and classify the potential existing gesture events. However, for interactive HMI systems, online recognition systems must recognize gesture events in less than a maximum response time L_{max} (or recognition latency).

One solution is to work with a frame-based approach. A frame consists of a subset of the past observed input samples that can be used for gesture segmentation (also known as gesture spotting) and recognition. Frame size should be shorter than L_{max}; otherwise, it is impossible to produce the output at the required time. But this introduces a new problem: the gesture duration could be longer than the required maximum latency ($d > L_{max}$). In this situation, a recognizer could either forecast the gesture or do a piece-wise recognition of the whole gesture.

Sliding windows with some overlapping are typically used in frame-based digital signal processing. In our case, we use a frame-based approach with no overlapping and a frame size of 6 samples. Thus, a frame q_i consist of a group of 6 consecutive samples of the sensor data stream X (see Equation 4.8).

$$q_i = \{x_{6i}, x_{6i+1}, ..., x_{6i+5}\} \tag{4.8}$$

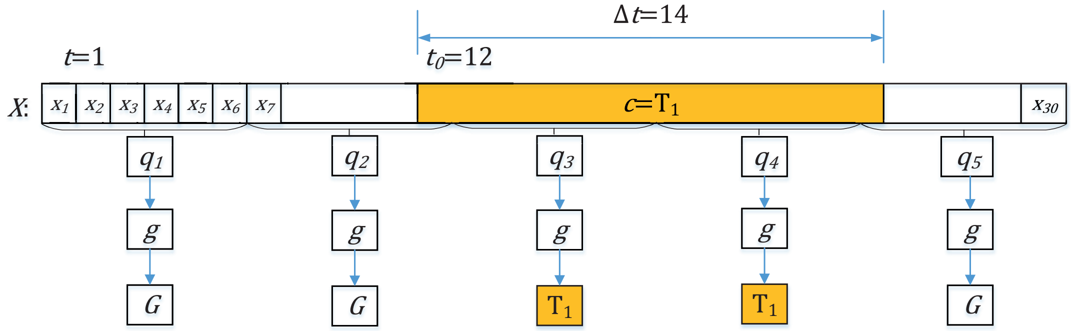


FIGURE 4.5: Example of the frame-based recognition approach for an input stream of 30 samples in which a tap gesture occurs on sample 12. The input samples X are grouped in 5 frames of 6 samples each. Frames 3 and 4 are classified as belonging to a Tap gesture for sensor 1, while other frames are classified as non-gesture

As we use a frame-based approach instead of an event-based approach, the new frame recognition function g only has to map each frame to any of the formerly defined gesture classes and an additional class G that denotes non-gesture.

$$g : q_i \rightarrow c|c \in \{T_0, ..., T_8, S_1, S_r, G\} \tag{4.9}$$

Figure 4.5 illustrates an example of how the frame-based approach would work for a short stream of data containing a single gesture event.

According to [455] and [456], the reaction time of an attentive individual to prime stimuli goes from 100 to 600 ms. For interactive gesture recognition systems in [457], Yin uses a maximum of 1-s reaction time, and Song a 100 ms [458]. In our case, as the sampling rate of the data stream is 356 Hz, the minimum latency of the system introduced with our frame-based approach will be at least 16 ms.

Data acquisition and annotation

To the best of our knowledge, there are no public datasets to test our system, so we collected our own dataset. The purpose is to be able to include as much variety of events as possible with the minimum human cost. A person was instructed to do a collection of gestures over the sensing surface. He did it with bare hands, wearing 2 mm neoprene gloves and 3 mm wool gloves to simulate the variety of possible different scenarios found in the real world.

All events were recorded separately. The dataset consists of 5224 recordings. Each recording contains the signal raw integer values of the voltage sensed by the ADC. Then, an annotation of events was generated. Instead of the classes foreseen in Equation 4.7, the annotation collects information more detail as described in Equation 4.10.

$$Ann_i = \{t_0, \Delta t, isT_0, ..., isT_8, d_{min}, Mat_{target}, S_c\} \tag{4.10}$$

where isT_i is a boolean value that determines if the finger has touched the sensor electrode i during the gesture. d_{min} is the minimum distance of the target to the sensor surface. It is zero if any isT_i is true. Mat_{target} is the material of the target, possible options are bare hand, neoprene glove, wool glove, or wet hand. Finally, the slide class identifier is used to annotate swipe events $S_c \in \{S_l, S_r, \emptyset\}$.

Although we are only interested in the two types of events, taps, and swipes, this annotation allows doing further analysis in the future.

For instance, a multi-button touch would show more than one touch and no swipe. While a swipe gesture would contain multiple touches and swipe information.

An inter-button touch would have a minimum distance of zero, that is, touching the surface, but no touches to any button. Different events are represented and in Figures 4.6a-4.6d, the curves from voltage measuring are represented for different events.

Instead of manually labeling the dataset, we opted for a semi-supervised approach. First, we determined the isT_i, d_{min}, and Mat_{target} for the all the recordings except slide recordings. This information is already available in the dataset recording campaign information. Then, for single touch events, a binary segmentation algorithm [459] based on the saddle point of the first derivative of the sensor data is applied. This allows determining t_0 and Δt. For multi-touch, inter-touch, and approach events, a similar method is applied to determine t_0 and Δt.

For slide gestures, isT_i was not available from the dataset test plan so it was first determined from an individual binary segmentation of each channel. Then, starting and ending buttons are determined to later compute the slide direction S_c and timing information t_0 and Δt.

In order to obtain the training, validation, and test sets, the samples have been separated in a stratified manner according to the 11 positive classes with the following proportions: 60% for training, 20% for validation, and 20% for the test.

For this partitioning, each event has been considered separately: the same proportion has been extracted from each fine-grain event. Thus, the training set consists of 3155 recordings, the validation set of 1035, and the test set of 1034 recordings.

As a single person was used to build the dataset, it contains some bias on the characteristics of the events. This affects especially the swipe events, which seem to have different moving patterns depending on their direction. To solve this problem, more recordings should be done with different people, and data augmentation techniques could be performed as well. These limitations will be addressed in future work.

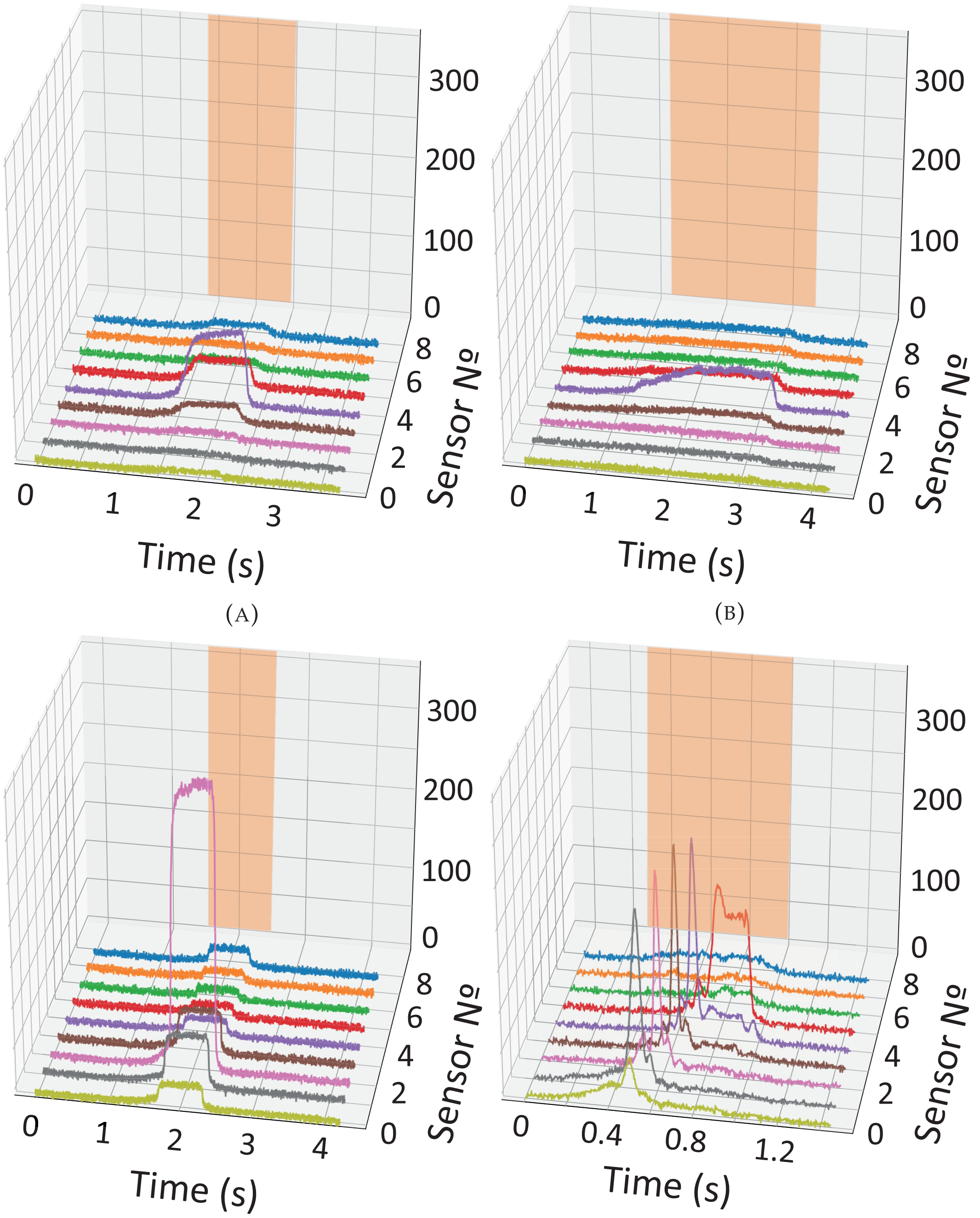
4.2.3 Implementation

Recurrent Neural Network Models

The models considered for use are three versions of RNN: vanilla [460], LSTM [77], and GRU [461]. However, independently of the cell type, their data input and output structures are the same. The input for each cell is a sample point with its corresponding features. The output of the network is the class of the segment related to the sequence of samples. In this way, the process works as looking at a segment but in a dynamic manner. That is, the model classifies segments. After the sequence has been classified, the internal state of the RNN cell is reused for the next segment. The structure of the network is detailed in Figure 4.7.

Regarding the three different RNN cells, vanilla is the simplest RNN network [460]. The number of parameters of a vanilla cell (network with only one layer) is $n \cdot (n + m + 1)$ while the number of operations is $2n \cdot (n + m)$, where n is the number of inputs and m the number of neurons.

LSTM cells are an improvement of vanilla cells created with the purpose of solving the exploding/vanishing gradient problem. They were introduced by Hochreiter in 1997 [77]. They have two hidden states, which serve as input for the next cell, and also different internal operations for efficient and advanced memory management. The number of parameters of an LSTM cell is 4



(A)

(B)

(C)

(D)

FIGURE 4.6: Dataset annotations for different gestures in different conditions. (A) tap gesture with a neoprene glove. (B) normal tap on button 2. (C) finger approach, which is not considered a valid gesture. (D) swipe right gesture.

times the corresponding for the equivalent Vanilla cell, that is $4n \cdot (n + m + 1)$. Regarding the number of operations, they are increasing by approximately 8 times $n \cdot (8n + 8m + 3) \approx 8n \cdot (n + m)$.

Finally, the GRU cell was developed in 2014 by Cho et al. [461] aiming to

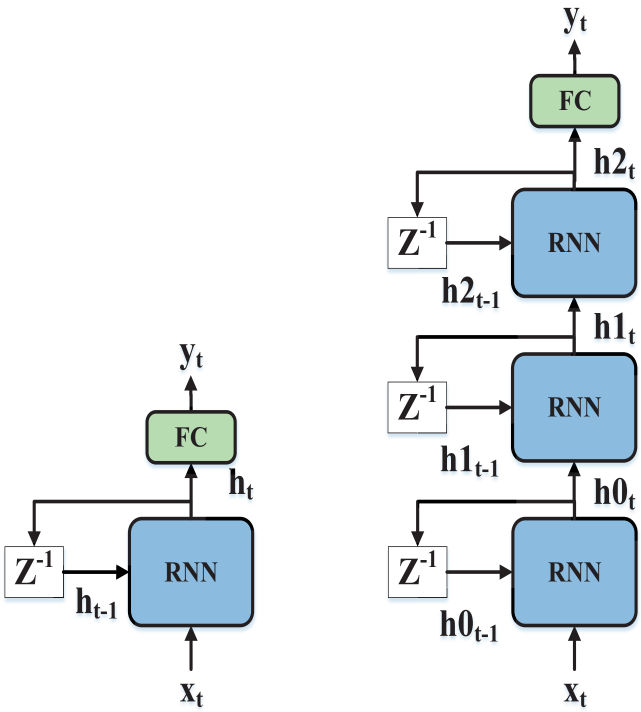


FIGURE 4.7: Detail of the RNN structure for any cell type. The input at each step is a sample, x, with the corresponding features for each sensor. The output of the class is for the whole sequence. (Left) Example of RNN-based system with 1 layer. (Right) A system with 3 layers.

solve also the vanishing gradient problem. In contrast to an LSTM cell, it only uses one hidden state between cells and has fewer internal gates. Hence, they have a lower number of parameters, $3n \cdot (n + m + 1)$ and also lower number of operations, $3n \cdot (2n + 2m + 1) \approx 6n \cdot (n + m)$. GRU has proven to yield similar results than the LSTM cell while being lighter.

Optimization

In this section, the different procedures to obtain the best-performing network are defined. The objective is to obtain the network with the highest accuracy at the minimum cost both in terms of resources and computing time.

This is a multidimensional multi-objective optimization problem. In this kind of problem, we have to simultaneously find the best values (either maximize or minimize) for several cost functions at the same time. Those functions can be interdependent and contradictory.

In our case, we define our cost function set to optimize as Equation 4.11

$$K(p) = (k_{NetPerf}(p), K_{Mem}(p), k_{time}(p)) \tag{4.11}$$

where p is a design space point, which in our case is a vector composed by the concatenation of the design architecture parameters vector, the hyperparameters of the network and its parameters.

For the different cost functions, we consider the accuracy of the network $k_{NetPerf}$, the required memory to store the network parameters k_{Mem}, and the execution time of the network k_{time}. As we will try to minimize the cost, we will express all the cost functions in such a way that they give a lower value for better design space points. We will also add some constraint functions that valid design space points must fulfill.

We use the recall metric to evaluate the ability of the network to correctly detect each class of the events in the acquired stream. However, since the optimization process needs a function to minimize, we define the cost associated with the network as

$$k_{NetPerf}(p) = 1 - Recall_{Avg} \tag{4.12}$$

where the average is computed over all the valid gesture classes.

Platform Cost Functions We are aiming for implementation in a low-cost 32-bit microcontroller with floating point support. The goal of the implementation is to avoid tightening to a specific architecture. We want to be able to change from microcontroller families. The price of different models of a microcontroller family is usually determined by the amount of RAM. We put as requirements a maximum 32 MHz clock frequency and 128 kB RAM memory. We estimate that a fraction of the RAM will be devoted to the program to execute and its runtime memory requirements, but a large part of them will be required to store the network parameters of the design $SParams(p)$. Since we use single precision floating variables, the memory requirements will be derived from

$$k_{Mem}(p) = 4 \cdot SParams(p) \tag{4.13}$$

We decide that the maximum RAM memory budget for the network parameters is 80KB, so we add the following equation to the model constraints.

$$K_{Mem} < 80KB \tag{4.14}$$

We also need to take into consideration the number of operations required to complete the network computation for every sample to estimate the required computing time. Floating-point operations take a different number of processor cycles, depending on the processor floating-point unit implementation and the C-Runtime implementations. We use some constant values for the different types of operations, and we associate a cost in cycles for each of them. We do not stick to a certain family of microcontrollers to be able to easily estimate the performance of the system on different processor architectures. The computing time is then defined by the number of operations and their cost, ending in an effective latency time. Thus, we add the following deadline as a constraint to the system

$$k_{time}(p) < 2.8ms \tag{4.15}$$

Optimization The optimization problem consists of analyzing a design space point and evaluating it with a number of metrics. In our case, RAM usage, execution time, and average recall.

If we want to try to make the networks deeper and less wide, we can see that we could probably use up to 3 layers. Or even more, it depends on the dimension of the hidden state of each layer. The problem is that at each layer we add a new n dimension to the search space, as each layer's hidden state can have a different width.

The training of a network is already an optimization of a multidimensional space with as many dimensions as the number of parameters.

Instead of using an expensive search algorithm for finding the best parameters for every different network architecture, such as grid search, or an algorithm that explores the space poorly, such as random search [462], we choose Bayesian Optimization (BO) [463, 464, 465]. This method offers the best tradeoff between space exploration and time consumption.

BO is a machine learning-based optimization procedure specially designed for global optimization of black-box1 functions without obtainable derivatives. The objective is to solve the problem of finding an optimal point p_{opt} that minimizes the cost function

$$p_{opt} = argmin_{p \in DS} K(p) \tag{4.16}$$

where DS refers to the design space.

The core idea of BO is that the cost of evaluating the objective function K is too high in terms of time, opportunity, or economic costs, and it is modeled by a surrogate continuous function g based on Bayesian statistics. The g function allows us to quickly predict the values of new design points without estimating the real K. Having a fast function predictor allows to the creation of another acquisition function q to obtain new points of the design space that are expected to minimize the cost function K. After the real function, K is evaluated for the new point, the statistical model of g is updated.

The surrogate function g is usually a Gaussian Process (GP) or a Tree Parzen Estimator (TPE) [271]. One advantage of TPE is that allows hierarchical and categorical variables directly without having to modify the surrogate function. Another difference between the two models is the part of the Bayesian statistics model they try to fit. While GPs fits the posterior probability, TPE fits the prior and likelihood functions.

In our case, we use the software Hyperopt [466] to perform the BO process, based on TPE for modeling the objective and acquisition functions. A limitation of Hyperopt is that it does not support multi-objective optimization, but single-objective optimization. Since network accuracy is our priority, we adapt the previous formalization to a single-cost function

$$K(p) = \left\{ \begin{matrix} \infty, & if k_{Mem}(p) > 80KB \\ \infty, & if k_{time}(p) > 2.8ms \\ k_{NetPerf}(p), & otherwise \end{matrix} \right\} \tag{4.17}$$

The combinations that are not meeting the size and performance criteria are directly discarded assigning an infinite cost.

We force three different optimization processes, one for each network type: Vanilla, LSTM, GRU. The network architecture parameters and the training hyperparameters are part of the multidimensional design space.

Hyperparameters and tricks When a new design space point is selected, we quickly evaluate the constraint functions to discard network architectures that do not meet the memory size and FLOPs requirements. The selected space point does not only sample values from the network architecture parameters but from the hyperparameters used for training.

The network architecture training is an optimization process in itself. The training process requires specifying of some hyperparameters. Since different hyperparameters will lead to different NN parameters, they are considered part of the design space to explore.

A difficulty to train RNNs is how to address the time of the sequences during training. A common technique is truncated backpropagation trough time (TBPTT) [467]. One of its important parameters is the number of time steps used. Large values will allow capturing longer temporal events, but if they are too large the vanishing gradient problem arises, so we test different values.

Learning rate (LR) is probably the most important hyperparameter since it controls the amount of learning that a neuron is receiving when backpropagation occurs. A common technique is to use variable LR. We use an exponential Learning rate scheduler defined by Equation 4.18. Where LR_0 is the initial learning rate, DR is the learning rate decay factor (i.e., decay rate), L_{step} is a value that is incremented after each learning step, and D_{step} (decay step) is a factor to modulate the intensity of the decay at each learning step.

$$LR = LR_0 \cdot DR^{\frac{L_{step}}{D_{step}}} \tag{4.18}$$

Some hyperparameters are fixed, and others are included in the optimization process. In some parameter dimensions, we limit the possible values to a discrete set, while in others we provide a continuous range with a uniform or log-uniform distribution to sample from. The hyperparameters with their ranges as defined in Table 4.1.

We use different batch sizes, and we limit the maximum training epochs to 10. As all the training variables are included in the optimization process, it is expected that networks that either overfit or underfit are ruled out during the process. However, to help avoid overfitting, dropout is also included after each cell.

Since detecting touch and slide segments are more important than detecting undesired events, we weigh the importance of those classes by a factor. We allow weight factors from 3 to 8.

Parameter	Range
TBPTT time steps	From 5 to 100 in 5 steps increments
LR_0	From 0.001 to 0.1 in log-uniform distribution
D_{step}	From 30 to 100 in 10 steps
DR	From 0.7 to 0.99 in log-uniform distribution
Batch size	2^n (with n from 1 to 8)
Epochs	From 1 to 10
Class weights	From 3 to 8
Dropout	0.0 to 0.9 in 0.05 steps

TABLE 4.1: Range of the hyperparameters for optimization.

	Vanilla	LSTM	GRU
L_{RNN}	1	1	1
n_h	16	16	16
Parameters	636	1932	1500
Memory (KB)	2.4	7.5	5.8
FLOPS	443 k	1.3 M	1.0 M
Time (ms)	0.2	0.6	0.4
Touch gestures average recall	64.26%	91.59%	80.78%
Slide gestures average recall	0%	28.50%	3.00%
$Recall_{avg}$	69.38%	83.54%	75.00%

TABLE 4.2: Initial network results.

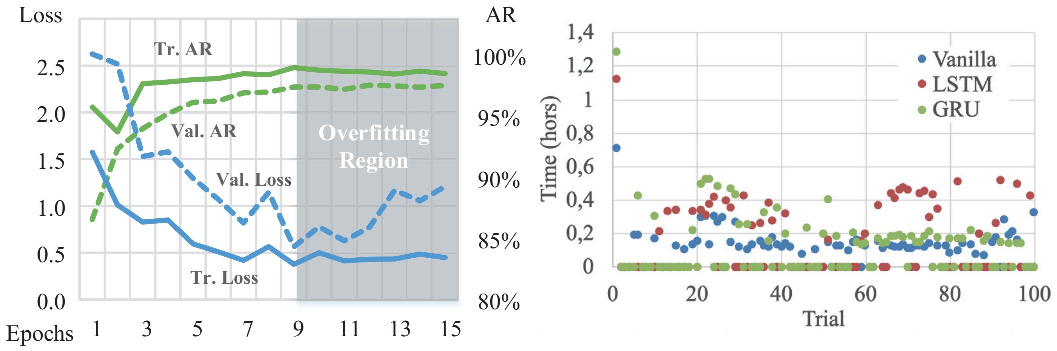
With this process, the optimal performing network regarding its hyperparameters and architecture components is found.

4.2.4 Results

We start the optimization process from 3 design points manually selected to compare the ability of the different RNN cells to capture the time dynamics of sequences.

We select a single-layer network with a hidden state of 16 neurons. The results are shown in Table 4.2. As expected, the LSTM and GRU networks have better performance than Vanilla. However, all the networks are far from an industrially acceptable performance, which we consider should be above 95%. Nevertheless, the amount of available memory and computing time in the final execution platform allows searching for other more complex implementations to find better-performing networks.

We run 100 iterations of the optimization process for every type of network on a computer with two Intel Xeon silver 4210 CPU processors running at 2.2 GHz with a total count of 20 cores and 40 hardware threads and 96 GB of



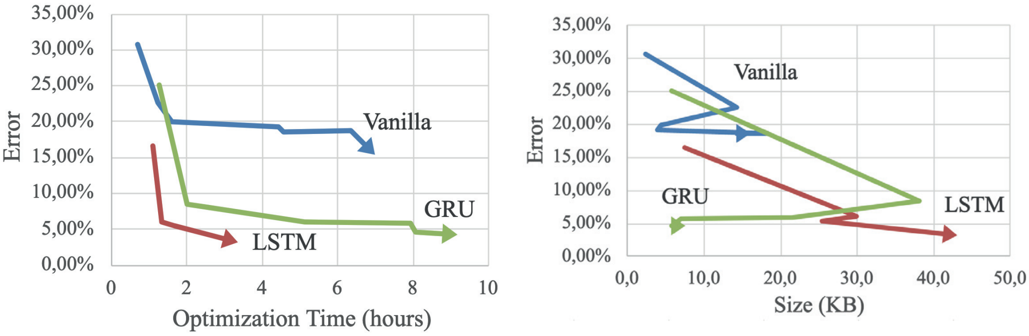
(A) (B)

FIGURE 4.8: (A) Plot of the training procedure for the best LSTM configuration of the optimization procedure. Loss in blue and average recall (AR) of the touch classes in green. Dashed lines correspond to validation (Val.) set values and solid to training set (Tr.) values. The minimum value of the validation loss delimits the transition edge into the over-fitting region. (B) Time of all the trials run during the optimization process.

RAM. We design the networks with Tensorflow 1.13 and run the optimization process using HyperOpt. During the training process, we control the average recall and loss in both the training and validation sets. We consider the model with the last best validation loss to avoid selecting an over-fitted model. A graphical illustration of the evolution of the loss and recall during training is illustrated in Figure 4.8a.

The total execution time of the optimization is more than 10, 13, and 12 h for Vanilla, LSTM, and GRU designs respectively. Figure 4.8b shows the execution time for each trial. Discarded network architectures are quickly evaluated while other design points take a variable amount of time depending on the size of the architecture and the training hyperparameters. The first iteration takes more time as the BO algorithm needs to sample several points before the Bayesian model can be built.

The goal of the optimization is to find networks with better performance, which is equivalent to minimal $kNetPerf$ cost. Obviously, not all the optimization trials obtain a smaller value for $kNetPerf$. Figure 4.9a shows the Pareto front of the trade-off between optimization time and the achieved cost. Although all training processes last for more than 10 h, the best results are found faster for LSTM than other networks. For the LSTM, the minimum is found in trial 23, while for Vanilla and GRU they are found in trial 67. In any case, the optimization process follows a classical pattern in the economics of diminishing returns, i.e., the benefits of additional training are decreasing



(A)

(B)

FIGURE 4.9: (A) Evolution of the minimum achieved error during the optimization process. After the first 2 h of running the optimization process, the error reductions are harder to achieve. (B) Evolution of the size of the parameter during the optimization process. The size of the model is often increased to achieve lower error rates, but big error reductions are also achieved by the change of other hyperparameters.

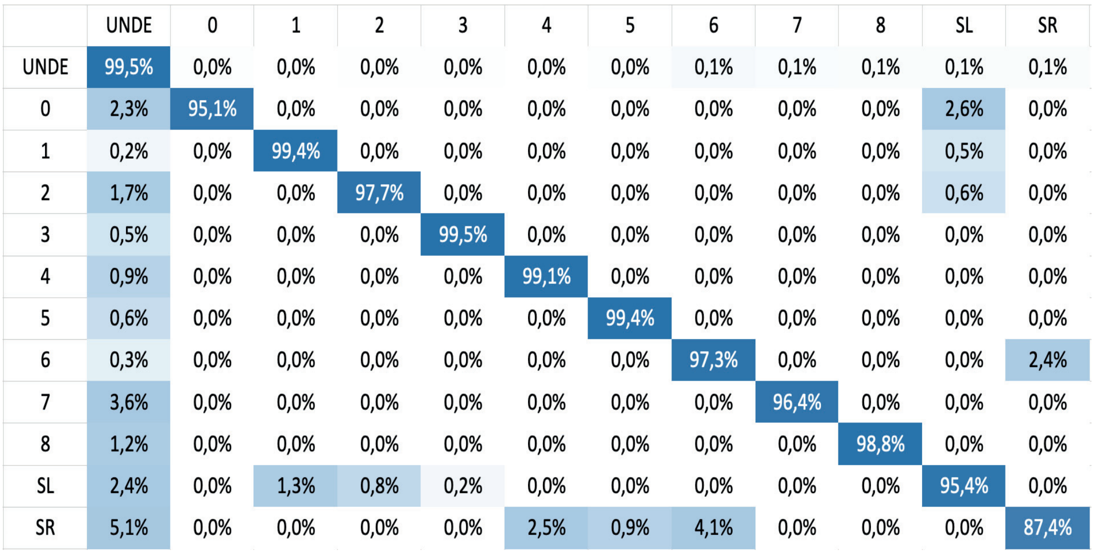
	Vanilla	LSTM	GRU
L_{RNN}	1	1	1
n_h	34	46	12
Parameters	1950	11052	984
Memory (KB)	7.6	43.2	3.8
Touch gestures average recall	95.68%	98.08%	98.69%
Slide gestures average recall	34.94%	91.37%	83.27%
$Recall_{avg}$	84.64%	96.86%	95.89%

TABLE 4.3: Best network designs found by the optimization process.

with time.

The best-obtained networks are described in Table 4.3. The best-achieved performance is given by an LSTM network but with just a slightly improved one with respect to the best GRU network. Compared with the initial results in Table 4.2, the improvement from optimization can be noted especially in the cases of LSTM and GRU. In the case of vanilla, the only class that has improved is touch. Both sliding gestures (left and right) almost have had no increase in classification accuracy. In LSTM and GRU, however, the improvement is mainly located in slide classes.

Both, the LSTM and the GRU are above the 95% minimum acceptable average recall that was part of our requirements. However, the complexity of the networks is very different. Fig.4.9b depicts how the parameter size has been evolving during the optimization process for different network designs. Some



	UNDE	0	1	2	3	4	5	6	7	8	SL	SR
UNDE	99,5%	0,0%	0,0%	0,0%	0,0%	0,0%	0,0%	0,1%	0,1%	0,1%	0,1%	0,1%
0	2,3%	95,1%	0,0%	0,0%	0,0%	0,0%	0,0%	0,0%	0,0%	0,0%	2,6%	0,0%
1	0,2%	0,0%	99,4%	0,0%	0,0%	0,0%	0,0%	0,0%	0,0%	0,0%	0,5%	0,0%
2	1,7%	0,0%	0,0%	97,7%	0,0%	0,0%	0,0%	0,0%	0,0%	0,0%	0,6%	0,0%
3	0,5%	0,0%	0,0%	0,0%	99,5%	0,0%	0,0%	0,0%	0,0%	0,0%	0,0%	0,0%
4	0,9%	0,0%	0,0%	0,0%	0,0%	99,1%	0,0%	0,0%	0,0%	0,0%	0,0%	0,0%
5	0,6%	0,0%	0,0%	0,0%	0,0%	0,0%	99,4%	0,0%	0,0%	0,0%	0,0%	0,0%
6	0,3%	0,0%	0,0%	0,0%	0,0%	0,0%	0,0%	97,3%	0,0%	0,0%	0,0%	2,4%
7	3,6%	0,0%	0,0%	0,0%	0,0%	0,0%	0,0%	0,0%	96,4%	0,0%	0,0%	0,0%
8	1,2%	0,0%	0,0%	0,0%	0,0%	0,0%	0,0%	0,0%	0,0%	98,8%	0,0%	0,0%
SL	2,4%	0,0%	1,3%	0,8%	0,2%	0,0%	0,0%	0,0%	0,0%	0,0%	95,4%	0,0%
SR	5,1%	0,0%	0,0%	0,0%	0,0%	2,5%	0,9%	4,1%	0,0%	0,0%	0,0%	87,4%

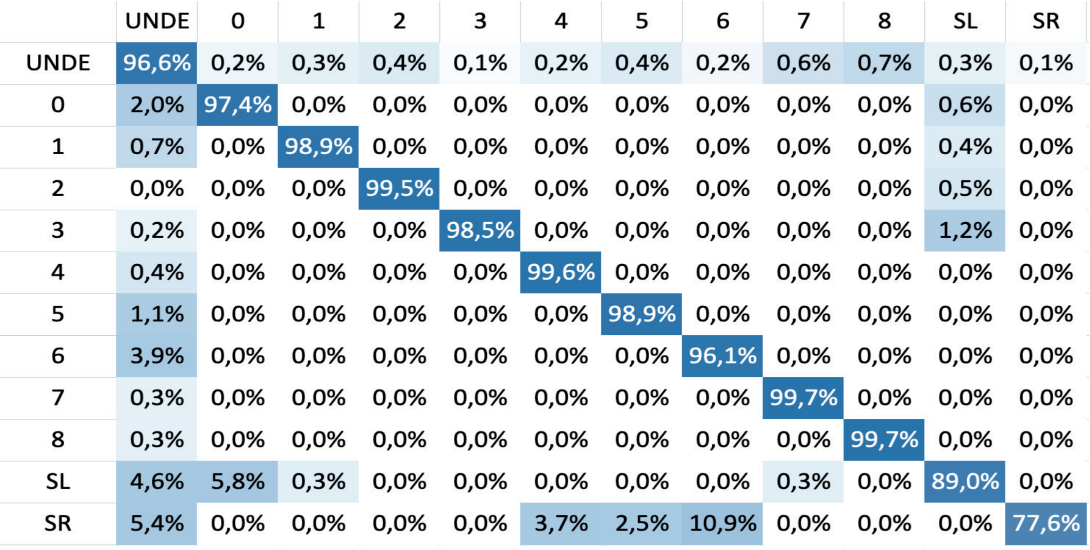
FIGURE 4.10: Confusion matrix for the best LSTM network design.

of the gains in performance are achieved by increasing the parameter size, especially in the LSTM network. But in other networks like Vanilla and GRU, important gains are achieved by changing the values of some hyperparameters such as the batch size and the learning rate.

It must be noted that both the architecture and training parameters are jointly optimized. This explains the apparently contradictory observation that in some steps decreasing the size results in less error, and in others, an increase in the number of parameters produces an increase in the error. The obtained Vanilla does not meet our minimum requirements and has a much lower performance than GRU and LSTM. In the case of both LSTM and GRU, we can observe a similar performance but with a different number of parameters. In the case of the LSTM, the model weighs more than twice that of the GRU model. In the case of the maximum RAM required for the model, the LSTM requires more than three times the memory of the GRU cell. The confusion matrices for both networks can be shown in Figure 4.10 and 4.11.

It is difficult to compare with other related systems found in the literature because of the difference in the technology of the sensors, their dimensions, or their supported gesture vocabulary. However, to give an idea of the achieved performance, we will compare it with the accuracy of the swipe gesture recognition which is common in various systems.

MonoTouch [468] (which is a special surface device) achieves an 85–93% accuracy for swipes. Flex Sensors [469] achieves a comparable 74–87% accuracy,



	UNDE	0	1	2	3	4	5	6	7	8	SL	SR
UNDE	96,6%	0,2%	0,3%	0,4%	0,1%	0,2%	0,4%	0,2%	0,6%	0,7%	0,3%	0,1%
0	2,0%	97,4%	0,0%	0,0%	0,0%	0,0%	0,0%	0,0%	0,0%	0,0%	0,6%	0,0%
1	0,7%	0,0%	98,9%	0,0%	0,0%	0,0%	0,0%	0,0%	0,0%	0,0%	0,4%	0,0%
2	0,0%	0,0%	0,0%	99,5%	0,0%	0,0%	0,0%	0,0%	0,0%	0,0%	0,5%	0,0%
3	0,2%	0,0%	0,0%	0,0%	98,5%	0,0%	0,0%	0,0%	0,0%	0,0%	1,2%	0,0%
4	0,4%	0,0%	0,0%	0,0%	0,0%	99,6%	0,0%	0,0%	0,0%	0,0%	0,0%	0,0%
5	1,1%	0,0%	0,0%	0,0%	0,0%	0,0%	98,9%	0,0%	0,0%	0,0%	0,0%	0,0%
6	3,9%	0,0%	0,0%	0,0%	0,0%	0,0%	0,0%	96,1%	0,0%	0,0%	0,0%	0,0%
7	0,3%	0,0%	0,0%	0,0%	0,0%	0,0%	0,0%	0,0%	99,7%	0,0%	0,0%	0,0%
8	0,3%	0,0%	0,0%	0,0%	0,0%	0,0%	0,0%	0,0%	0,0%	99,7%	0,0%	0,0%
SL	4,6%	5,8%	0,3%	0,0%	0,0%	0,0%	0,0%	0,0%	0,3%	0,0%	89,0%	0,0%
SR	5,4%	0,0%	0,0%	0,0%	0,0%	3,7%	2,5%	10,9%	0,0%	0,0%	0,0%	77,6%

FIGURE 4.11: Confusion matrix for the best GRU network design.

but in this case, the comparison is harder because it is based on a sequence of finger poses. TouchRing [470] (which is not based on the touch panel, but a smart ring) achieves a 78% accuracy on the swipes. Knuckletouch [454], which follows a more similar approach to our work, achieves a 88–89% accuracy for swipes. In our case, we achieve an 87–95% accuracy, which is comparable to or better than the former works.

In any case, both networks have been validated in a final industrial product with positive results. Although the GRU network has lower performance, its very low memory consumption allows using a lower-cost microcontroller to achieve additional savings.

Figure 4.12 depicts the parts of the final industrial product prototype based on the presented techniques. A low-cost standard microcontroller running the RNN-based gesture recognizer is mounted on a PCB which is connected to the sensor foil. The device is placed under the mechanical enclosure of the car dashboard.

4.3 SpArSe Extension

In the second section of this chapter, the topic of NAS, which was illustrated in Section 2.2, is researched in conjunction with the other two corpus topics: gesture recognition and MCUs.

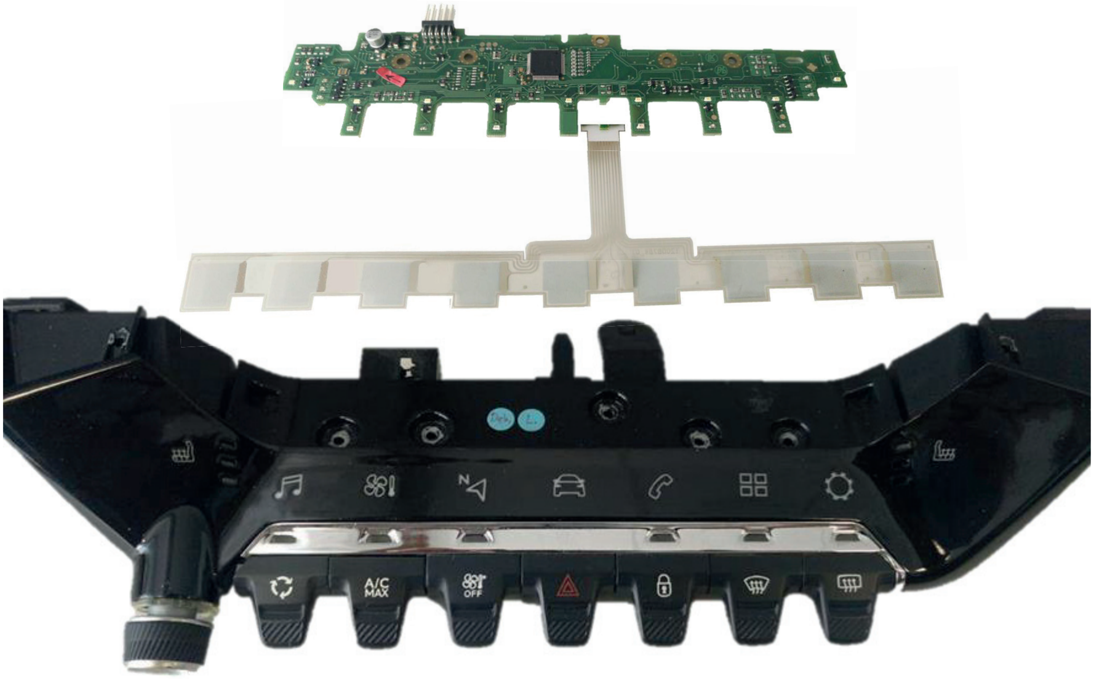


FIGURE 4.12: Industrial design of the final product. From top to bottom (1) microcontroller PCB, (2) capacitive sensor foil, and (3) mechanical enclosure.

The first objective is to implement, modify, and improve SpArSe [188]: an automatic CNN builder for image classification. The second objective is to extend it to account for latency as a new objective and incorporate recurrent cells as part of its search space. The last objective is to test the framework in a gesture recognition problem, Corpus of Social Touch [435] (CoST).

We also succinctly use pruning and quantization, which were described in Sections 2.3 and 2.1, respectively, as means of further decreasing the requirements of the developed models.

4.3.1 Experimental Design

SpArSe Modification and Implementation

SpArSe is built upon two major components: the search space, Ω, and the search strategy. Each point in the search space, Ω_i, is a specification of the neural network in terms of architecture and training parameters. The search strategy consists of a multi-objective Bayesian optimization procedure, improved with morphisms to reduce training time. Each of the configuration points sampled during the search, Ω_i, is evaluated with regards to a performance constituted as a three objective target: accuracy, Model Size (MS), and

Working Memory (WM). Each of the three metrics is defined as

$$Error(\Omega_i) = 1 - Accuracy_{Val}(\Omega_i) \tag{4.19}$$

$$ModelSize(\Omega_i) = \sum_i ||w_i||_0 \tag{4.20}$$

$$WorkingMemory(\Omega_i) = \max_l(||x_l||_0 + ||w_l||_0 + ||y_l||_0) \tag{4.21}$$

where l indicates layer l, w the weight matrix, x the input, and y the output of layer l. In the original paper, *WorkingMemory* (WM) only includes input and output or input and weight.

The search space, which is detailed in the original work, consists of sequential convolutional layers organized in blocks, altogether with the possibility of branching through fully connected (FC) layers, and finally, the last output layer. This search space comprises a wide range of choices encountered when building a network of such characteristics: number of layers, type, and number of filters for each layer, as well as other training and pruning parameters.

An important note is that, as the search space is composed of conditional variables, an implementation of the Arc Kernel [276] is used, and is especially suited for conditional spaces. By decomposing the space using and using a cylindrical embedding, it is able to assess the conditionality among variables. The embedding g, $g_i : \mathcal{X}_i \longrightarrow \mathbb{R}^2$, where $\mathcal{X}_i$ is the original dimension i space, can be used with any distance and covariance method:

$$g_i(\mathbf{x}) = \begin{cases} [0,0]^T & if \delta_i(\mathbf{x} = False \\ w_i \left[\sin\left(\pi\rho_i \frac{x_i}{u_i - l_i}\right), \cos\left(\pi\rho_i \frac{x_i}{u_i - l_i}\right)\right] & otherwise \end{cases}$$

where $w_i \in \mathbb{R}^+$ and $\rho \in [0,1]$ are the radius and angle factor of the cylindrical space, $u_i - l_i$ establishes the length scale of the dimension i, and δ_i is a delta function establishing the existence of coupling among dimensions x. The implementation, developed by the author, can be found in the GPyTorch documentation for the Arc Kernel.

1) Search Space Changes: In our implementation, the changes applied to the search space are the deletion of branching through FC layers and the simplification of the pruning strategy. The main reason to avoid branching is the increase in weight and complexity. As explored in other studies in greater detail [161], the increased complexity could help to discover new structures inside the network. Regarding pruning, the original paper uses structured and unstructured pruning, whereas we have used only unstructured pruning.

2) Search Procedure Changes: The search procedure in the original implementation consists of three stages: in the first, new networks are sampled randomly or morphed from a previous one. In the second stage, the morphs are restricted to pruning. In the third stage, the reference configurations are restricted to optimal Pareto Points. The architecture sampling procedure is performed through Thompson Sampling (TS).

In general, we conserve the three-stage procedure, but with changes with regards to the AcqF and the network training procedure. In the first stage, we follow a low fidelity scheme, as detailed in [187], consisting of training for only one epoch and sample models following a random procedure. In this case, there are no morphisms, since we want to explore the space and obtain architectural performance information. In the second stage, we continue without morphisms but the sampling procedure is led by a GPs with Monte Carlo-based Expected Improvement (EI) as AcqF. Thus, we rely on the AF for the proposal of new points, while in the original paper this was relegated to morphisms or TS. The reason is the evidence of greedy exploration by TS [471, 472], while EI, among others, balances better the exploration/exploitation trade-off. In the second stage, we also increase the epochs through which the models are trained. In the final stage, we base new sampling points only on morphisms directed through the AcqF. The training epochs are extended to a normal training procedure.

In addition, we have changed the method by which the maximum working memory is calculated, as detailed in Equation 4.21. In the present implementation, the maximum feature map size or WM is the maximum sum of parameters of input, output, and weight along the network. An increase in WM is, therefore, expected. In the case of an RNN cell, the present approach has been to perform the WM computation for a cell as the unit of measure: we include in the computation the input, the output, and all the weights and states of the cell. We base this choice on the fact that in most implementations RNN cells are built as a whole function and hence all these components consume memory resources.

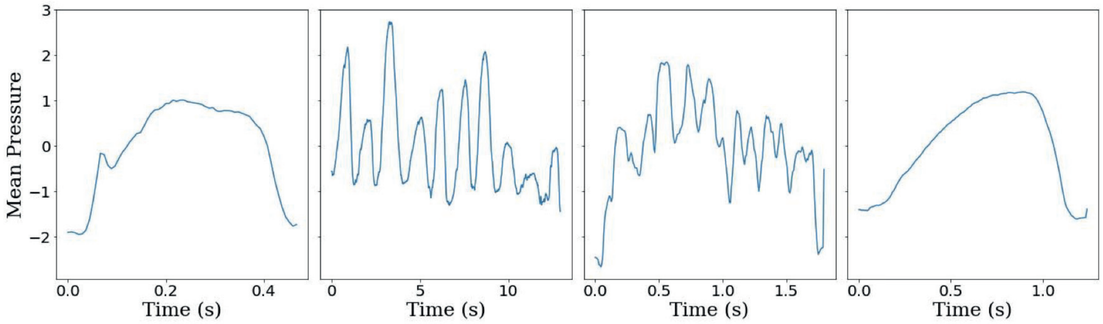


FIGURE 4.13: Example of CoST samples. First, from the left, grab the gesture. Next, tickle gesture. Third from the left, scratch and, finally, squeeze. Values of pressure are standardized by mean subtraction and standard deviation division.

Search Space Extension and CoST Dataset

Next, we extend SpArSe to gesture recognition, specifically, to sequence classification. The gestures that will be classified consist, as detailed next, of multivariate sequences. As the gestures must be classified within a certain time span to ensure an adequate user experience [473], latency is also included as another objective. To specify the latency of the network, the number of floating operations (FLOP) used to predict a sample input is needed. Then, we let the user specify the frequency or speed of its platform to finally obtain a latency measure by comparing both measures.

$$Latency(\Omega_i) = \frac{FLOPS}{PlatformFrequency} \tag{4.22}$$

To be able to capture temporal dependencies, RNNs are also incorporated into the search space.

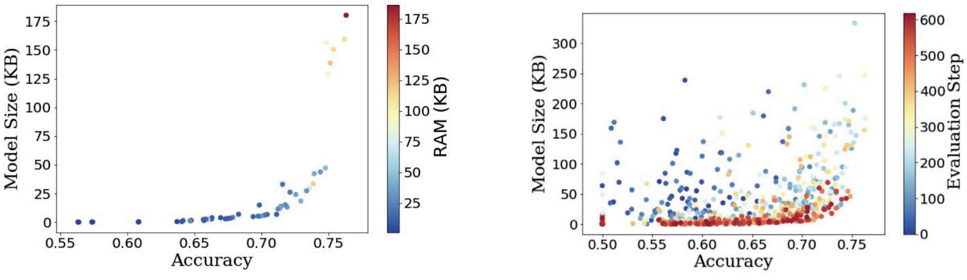
In order to test out the RNN and latency inclusion, and the overall suitability for gesture classification, the next task is to trial our implementation and extension of SpArSe with a suitable dataset: the Corpus of Social Touch (CoST). It consists of sequences coming from an arm-placed device with 64 pressure sensors arranged in an 8x8 grid. The sensor is placed in the user's arm to detect the impulses for different types of actions. Each of the values of sensors ranges from 0 to 1023 and the sampling frequency is 135 Hz. Each sequence is arranged as a tensor of dimensions Tx8x8, where T is the length of the sequence. The lengths of the sequences vary from 10 to 1747 samples. The dataset was recorded with the participation of 31 subjects and comprises 14 classes: grab, hit, massage, pat, pinch, poke, press, rub, scratch, slap, stroke, squeeze, tap, and, tickle. In Figure 4.13, samples for different classes from the dataset averaged over the 64 channels, are presented.

In the present study, the CoST dataset has been divided into two different manners. For the NAS procedure, the dataset has been split randomly between 20 subjects for training, 5 for validation, and 6 for testing. After finding the best performing network, we apply leave-one-subject-out testing to obtain the test results. Such division is in accordance of previous studies [27, 28, 474].

As stated in the previous section, the lengths of the sequences are quite large. This induces problems both for RNN and CNN-based architectures. For CNN's, saving and using whole sequences would be a problem for embedded deployment since we would be storing a really large array. For RNNS, there is a dichotomy in the inference phase: if the RNN cell prediction is faster than the sampling rate there is no problem with real-time prediction. However, if it is slower you are forced to store all the data in RAM, which can be impossible for most microcontroller capabilities (a sequence of, for example, 500 samples, equals 125 KB, in 32-bit float numbers). Hence, we devise two strategies: first, partition the sequences and predict the sub-sequences, as done in previous studies [27, 475], and second, pick samples from inside the sequence (fixing the beginning and the end). With the first method, the sequence is not processed as a whole but rather separately, while in the second, although intermediate information is lost, the whole sequence is perceived. This is important since for some sequences the gesture is recorded at the end [474]. Both the number of chops and the in-sampling number are included in the search procedure. In the case of CNN's, the sequences are arranged in the original 8x8 grid, obtaining an [T, 8, 8] tensor, where T is the sequence length. To adapt the CNN to different T sizes the model is padded to the length chosen for the sequence, as detailed previously. The only preprocessing applied to the dataset is standardization by mean subtraction and standard deviation division for each channel.

4.3.2 Results

The network builder and trainer have been developed using PyTorch, as well as the pruning and quantization procedures. Regarding quantization, post-training static quantization is applied to CNNs and dynamic quantization to RNNs, both cases with 8-bit types. The search algorithm is mainly built with



(A)

(B)

FIGURE 4.14: (A) Pareto frontier for our implementation of Sparse and CIFAR 10 Binary. Notice the different possibilities to choose from regarding RAM, size, and performance. Each point corresponds to a specific configuration of the search space. (B) Full search procedure for CIFAR-10 Binary with the modified SpArSe. Color degradation shows the progress of the search.

Framework	CIFAR-10 Binary				MNIST			
	Accuracy (%)	**MS** (KB)	**WM** (KB)	**GPUD**	**Accuracy** (%)	**MS** (KB)	**WM** (KB)	**GPUD**
SpArSe [188]	73.66	1.13	3.95	25	97.03	1.38	15	1
Ours	71.15	6.78	11.77	1	98.16	3.88	7.80	3

TABLE 4.4: Results for the original version of SpArSe and our implementation. Accuracy is in % and Size corresponds to the model weight in KB taking into account only weights and not code. The maximum RAM is also in KB and corresponds, in our case, to the computation specified in Section III-A. Our results are run in a single RTX 2070 Ti, while the original paper uses four RTX 2080. GPUD corresponds to Graphic Processing Unit days.

Ax [476]. Each of the models and search procedures has been carried out with a single RTX 2070 Ti.

SpArSe Implementation Results

We have compared the original implementation with our implementation after modifications. The comparison has been made with the MNIST [79] and CIFAR 10 Binary [477] image classification datasets, with the same preprocessing and splits as in the original paper.

In Table 4.4, results for the modified version can be compared with the original implementation. In both cases, MNIST and CIFAR10 Binary, the size of the model are bigger than in the original implementation. In the case of WM, it is bigger for the original SpArSe in CIFAR10 Binary while it is lower in the MNIST case. Regarding the accuracy, we obtain a 2 percentage point worse result in the case of CIFAR10 Binary, but in the MNIST case, we obtain a result better by a 1 percentage point. In general and at first sight, our models seem to weigh more and have similar performance. However, the authors consider the results inconclusive with regards to which approach is better. The reason is the stochastic nature of the search strategy and the oscillations that can be found in the results after multiple runs. For this matter, a broad comparison would be needed in order to achieve a fairer comparison.

Nevertheless, that does not imply that the increase in MS could not be due to the architecture change. From the argument previously mentioned in Section 4.3.1, it could be followed that reduced search space and the impossibility of the pruning strategy to prune further compared to the original search space, would difficult the task of finding better and smaller networks. That is, the bigger the search space the greater the probability of finding a well-performing network with a smaller size and feature maps.

Finally, a point that was not noted in the original paper is the utility of the Pareto frontier as the final result. With the frontier, we are able to choose among a range of different combinations suiting our direct hardware constraints. For example, and as illustrated in Figure 4.14a, we could choose a heavier network in the CIFAR10 Binary, with 74.79% accuracy, 47.12 KB in size, and 46.54 KB in WM. Or, if we were constrained for those resources, we could choose an example of the other extreme with 60.8% accuracy, 0.13 KB in size, and 1.536 KB in RAM. Hence, as stated in the previous paragraph, it would be necessary to determine a fairer comparison between methods, by, for example, comparing the Pareto frontier.

An important point is also the search reduction time. In the CIFAR-10 Binary case, the original implementation lasts 25 days searching for the final configuration, while in our case the search only lasts one day. In the MNIST case, our search endures 3 days compared to 1 day of the original implementation. However, we have only used 1 RTX 2070 and the original implementation employs 4 RTX 2080 Ti, clearly distorting the comparison.

CoST

In the case of the CoST dataset, we have first compared the results between splitting or in-sampling the data for a pure LSTM or GRU model, as established in Section 4.3.1. Results are illustrated in Table 4.5. As seen, cutting the signal delivers better results than sampling points from it, probably due

Model	**Model Type**	**Accuracy** (%)	**MS** (KB)	**WM** (KB)	**Latency** (ms)
Albawi et al. [39]	CNN	63.71	3556.28*	2105.34∗	-
Ta et al. [41]	Random Forest	60.8	-	-	-
Hughes et al. [42]	3DCNN + RNN	52.86∗∗	71.34	175.95	-
Ours	RNN Cut	65.12	15.77	13.86	199.62
Ours	RNN Insample	58.38	32.20	31.89	155.43
Ours	2DCNN	63.54	77.21	45.36	134.36
Ours	2DCNN + RNN	61.64	74.77	51.66	267.12

TABLE 4.5: Results for the CoST dataset with hold-one-subject-out testing, as in [27] and [28]. * indicates that it has been estimated from the network details in the original paper. ** In this case, authors did not use leave-one-subject-out testing and hence only MS and WM are directly comparable.

Model Type	**Accuracy** (%)	**MS** (KB)	**WM** (KB)	**Latency** (ms)
CNN	63.54	77.21	45.36	134.36
	61.44	27.74	42.64	100.60
	52.93	13.58	18.54	10.49
CNN + RNN	62.07	112.52	72.34	221.14
	61.64	74.77	51.66	267.12
	57.69	28.22	34.04	56.70

TABLE 4.6: Results for the CoST dataset with hold-one-subject-out testing for CNN and CNN+RNN search spaces. The first three rows are for results with CNN and the last three with CNN+RNN. Each row corresponds to the test result of different configuration points in the Pareto front.

to the loss of signal information when sampling. Next, we tested the framework with CNN and CNN plus RNN-based search spaces and compared all of them to previous implementations. In general, the RNN with signal cutting provides state-of-the-art results for all metrics except latency, which is provided by a CNN.

As stated before, the results presented correspond to certain configurations of the Pareto front. Other results, although totally useful, are usually neglected. For this reason, a selection of other points of the Pareto front corresponding to CNN and CNN+RNN cases can be found in Table 4.6. As can be seen, although results regarding accuracy could be similar, there is a range of possibilities to choose among regarding different sizes and latency values. All in all, our implementation solutions have proven to be directly usable in resource-constrained environments while obtaining the maximum performance for the defined search space.

4.4 2D Gesture Recognition

In the fourth section of this chapter, we explore the application and research of an unsupervised method for 2D gesture recognition with a capacitive matrix: NNs as information encoder, plus a K-Means algorithm for clustering the encoded vectors. Moreover, we employ the software developed in the previous section (Section 4.3) to find suitable CNN and RNN networks in an automatic manner. Finally, we develop an automatic quantizer and conversor [219] for porting our models from PyTorch to code in CMSIS-NN directly usable in Cortex-M or A MCUs.

4.4.1 Sensing method and data

Capacitive Sensing Platform

As depicted in Figure 4.15, the sensing platform consists of a surface touch module with three elements: the plastic cover, the capacitive foil, and the embedded board. The touch interface is the upper part of the plastic cover, which is immediately after the capacitive foil and glued to it. The embedded board is in the lower part and connected to the capacitive foil. The sensing component, which is the capacitive foil, consists of a central sensitive surface and 24 electrodes that run through it: 9 horizontally and 13 vertically, plus a global shield electrode.

The sensing mechanism is based on the transference of charge between two capacitors: integration and measurement condensers, as detailed in Section 4.2 or in the work [2].

The final result of a measurement is a set of voltages sensed at the measurement capacitor, one for each electrode. Each voltage is then converted through a 10-bit ADC to obtain a final raw value for each electrode at a predefined sampling rate (in the present case, 500 Hz).

Collection and Preparation

The user enters the gestures by touching the sensing surface and, without withdrawing the finger, draws the gesture on it. We detect the touch by the method described in [478], and then we begin to collect the raw values of the electrodes as described in the previous section. By examining which electrodes, horizontally and vertically, have the highest value we can determine in which spatial coordinates the finger is placed. When the user withdraws the

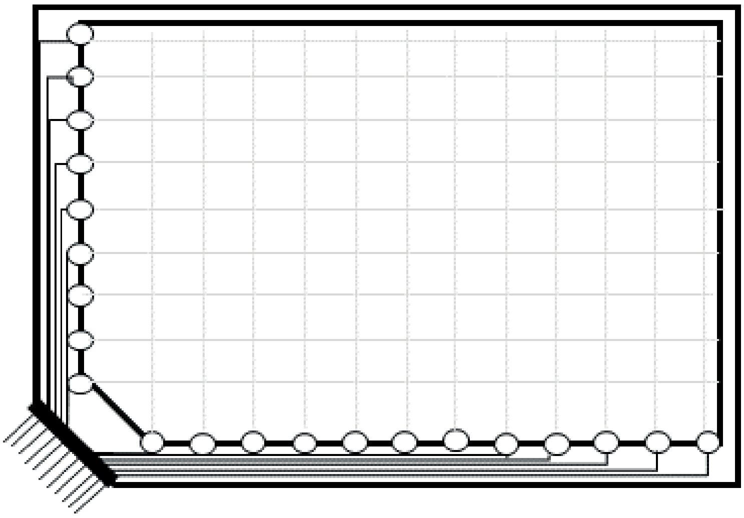


FIGURE 4.15: Illustration of the capacitive foil. Each of the dots represents an electrode, emitting each one a distinctive measurement of its row/column on the surface. The foil is pasted underneath the plastic cover and connected to the microcontroller through the lower-left pins.

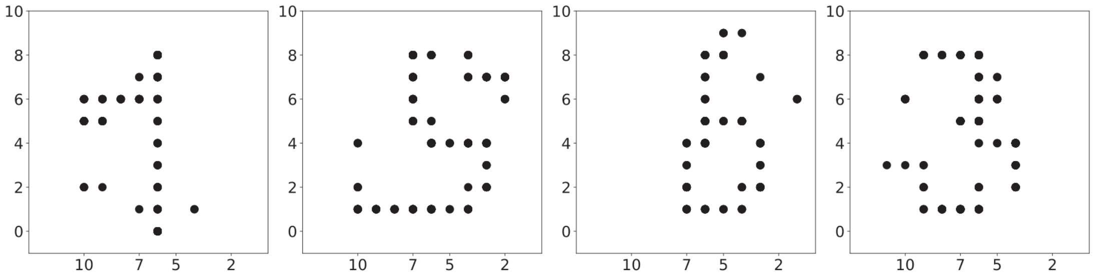


FIGURE 4.16: Examples of numbers drawn at the sensitive surface. From left to right: a one, a five, a six, and a three. The black points correspond to the electrode pairs through which the finger has passed. In an image configuration they correspond to 1s while the background is set to 0.

finger, we stop collecting data. It has to be noted, that the gestures described do not contain segmentation and thus we do not assess that problem. That is, we consider a gesture all the signals from a touch to a withdrawal, and force users to perform the whole gesture in such a manner.

By integrating all the coordinates through which the finger has passed we construct a greyscale image of 9×13 pixels of the figure drawn. That is, we mark the points through which the user has passed with a 1, while the rest of the untouched points remain as background with a value of 0. Hence, the orientation of the strokes is deleted and only the final form of the gesture drawn is recorded. That is we produce a static version of the gesture. To ease the deployment in a microcontroller and due to deployment framework limitations, we end up squaring the image to 13x13 by padding it. An example of such a method can be seen in Fig 4.16, where different numbers drawn on the sensitive surface can be visualized as the points through which the finger has passed.

To collect the dataset, numbers from 1 to 9 were drawn a total of 100 times per number on the sensing surface. The dataset is divided in a stratified manner in training, validation, and test, with 720, 90, and 90 samples respectively.

4.4.2 Models and NAS framework

Models

We consider two auto-encoder (AE) model types, that is, networks that are trained to replicate the input. The first a Convolutional Auto-encoder (CAE), and the second is a Dense Auto-encoder (DAE), made only of fully connected layers.

In general, the architecture consists of two differentiated parts: the first, the encoder, is in charge of extracting the information and compressing it into a central vector, and the second, reconstructing the input. In the CAE case, compression is performed by a succession of convolutional or pooling layers followed by ReLU activations, while the input reconstruction is performed by deconvolution layers and final upsampling. In the DAE case, all the layers are fully connected followed by ReLU activations, except for the final layer which reconstructs the input by reordering the output of the network. In both cases, there is a central fully connected layer that produces the latent feature vector. An important note is that to reconstruct the input, as it is made of 0s and 1s, we have used a Binary Cross-entropy Loss for training the AEs.

To provide unsupervised classification, a KM algorithm is in charge of grouping the compressed feature vectors into clusters. The choice of KM is central since it needs the number of clusters to be specified. In our case, this is precisely the number of classes.

Training and Model Optimization

To train the whole unsupervised method we divide the training into three separate steps. First, we train the AE to replicate the input. The loss used is an L2 norm-based reconstruction loss. The networks are trained for a maximum of 50 epochs and the model with the best validation loss is used as the final model; the number of epochs is chosen based upon observation of previous training. Second, once it has stopped training, we forward the entire training dataset through it and collect the latent feature vector for each input. The latent vector corresponds to the central chosen vector, which can be seen in Figure 4.17. In the CAE case, it has 12 elements, while in the DAE it has 117. Hence, the feature extraction through the autoencoder acts as a form of data compression algorithm with an effective compression ratio close to 10.

Third, we train the KM algorithm with all the collected feature vectors. That is, we have a feature vector per image and each of those vectors correspond to the central vector obtained by forwarding the corresponding image through the trained autoencoder. For example, in the CAE case, we have 720 vectors of size 12. As we are classifying 9 different numbers (from digit 1 to 9, excluding 0), we establish 9 clusters for the K-Means algorithms. K-Means algorithm is initialized following the k-means++ strategy [479] of similarly distanced cluster centers and the distance used is the L-2 norm2. Finally, to obtain test or validation results, we forward the test or validation sets through the AE, obtain the feature vectors, and predict to which cluster they belong.

Regarding the performance measure, we have selected a clustering supervised measure, V-Measure [480], for checking the actual classification capabilities of the framework. Choosing an actual unsupervised clustering metric, such as Silhouette Score would not provide direct insight into the accuracy of the predictions. The training, however, remains fully unsupervised.

As detailed as the first step, we have to build and train the network. However, building and tuning neural networks to be well-performing but also to comply with hardware requirements is a difficult task. Hence, we employ an extended implementation of a NAS framework [188] oriented towards optimizing accuracy, model size, maximum feature map (working memory), and latency. The details of the NAS and its extension are defined in Section 4.3.1.

The framework works, in each architecture case, CAE or DAE, with a configuration space composed of both training hyperparameters and architectural parameters, including post-training linear static quantization and L1-Norm-based pruning. Once the search space has been defined, the framework sequentially searches for solutions that minimize the previously mentioned and defined objectives (Equations 4.19-4.22).

Once we have the best model found and trained by our NAS framework, we

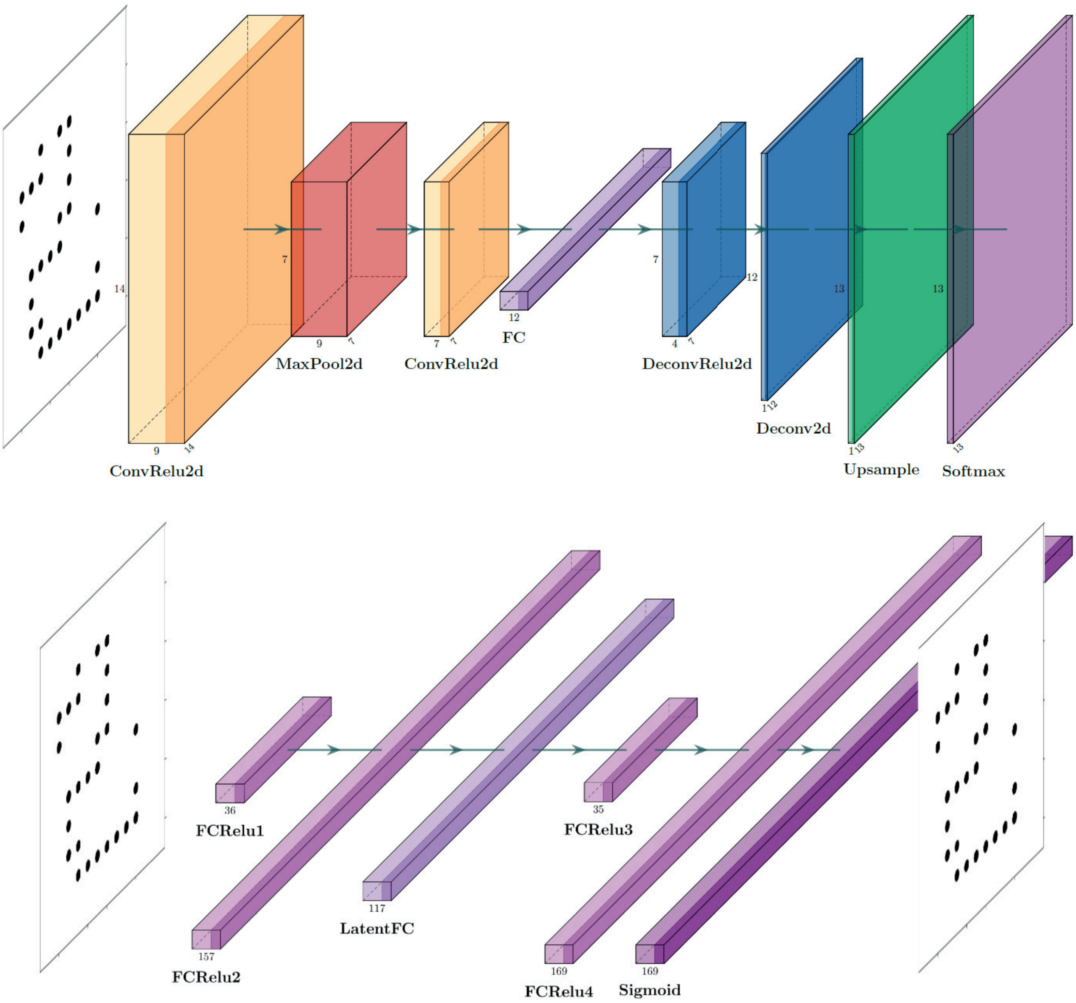


FIGURE 4.17: Best architectures resulting from the optimization procedures for both the CNN and FC autoencoders. The elements in the image represent the feature maps. In the convolution case, the size of the feature map is indicated by the vertical and depth numbers, while the horizontal number indicates the number of feature maps. In the FC case, the number indicates the number of neurons in each step, and the images are flattened at the beginning and reshaped at the end.

can proceed with the next steps of training the KM model and obtaining test results.

4.4.3 Results

The models established, CAE and DAE, are both developed in PyTorch and the NAS procedure is developed under Ax [476] and BoTorch [481]. To produce the embedded code and transform the models, we use Torch2CMSIS

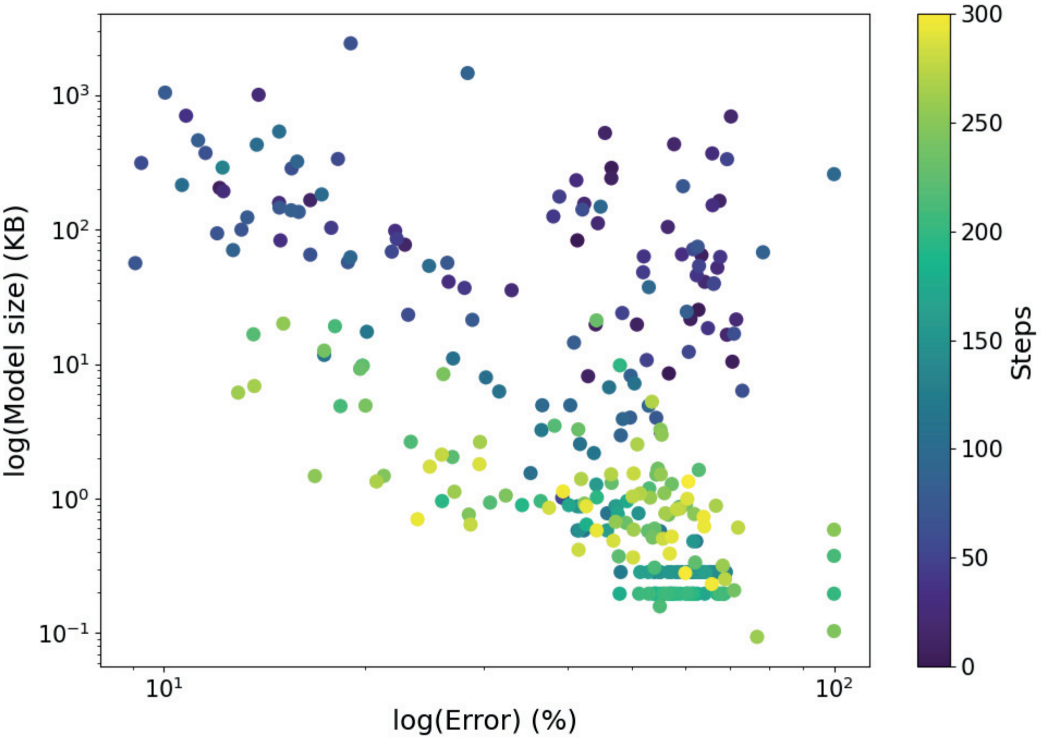


FIGURE 4.18: Optimization procedure for CAE. The y-axis represents the Model size and the x-axis the error (per pixel classification). Both axes are on a logarithmic scale. The color illustrates the evolution procedure as each of the steps of the Bayesian optimization procedure.

Model	**MF** (kB)	**MS** (kB)	**L** (ms)	**V-M** (%)
CNN	2.69	6.13	5.53	87.18
FC	18.32	39.67	4.08	58.63

TABLE 4.7: Results for the optimization procedure with SpArSeMod. MF corresponds to maximum feature map size, MS to model size, L to latency, and V-M to the V-Measure.

[219], my library for model conversion and quantization between PyTorch and the CMSIS-NN suite.

First, we illustrate the optimization procedure for the CAE with SpArseMod in Figure 4.18. As seen, while the optimization procedure advances there is a decrease in the model size. Also, one can observe the trade-off between the model size and the error: while we continue to obtain smaller architectures the error worsens, obtaining in some cases a compromise.

The final result is an election of the best points of the search: the Pareto frontier. With the best-chosen model, we train and test the K-Means unsupervised clustering and classification. We present the optimization results for the AEs,

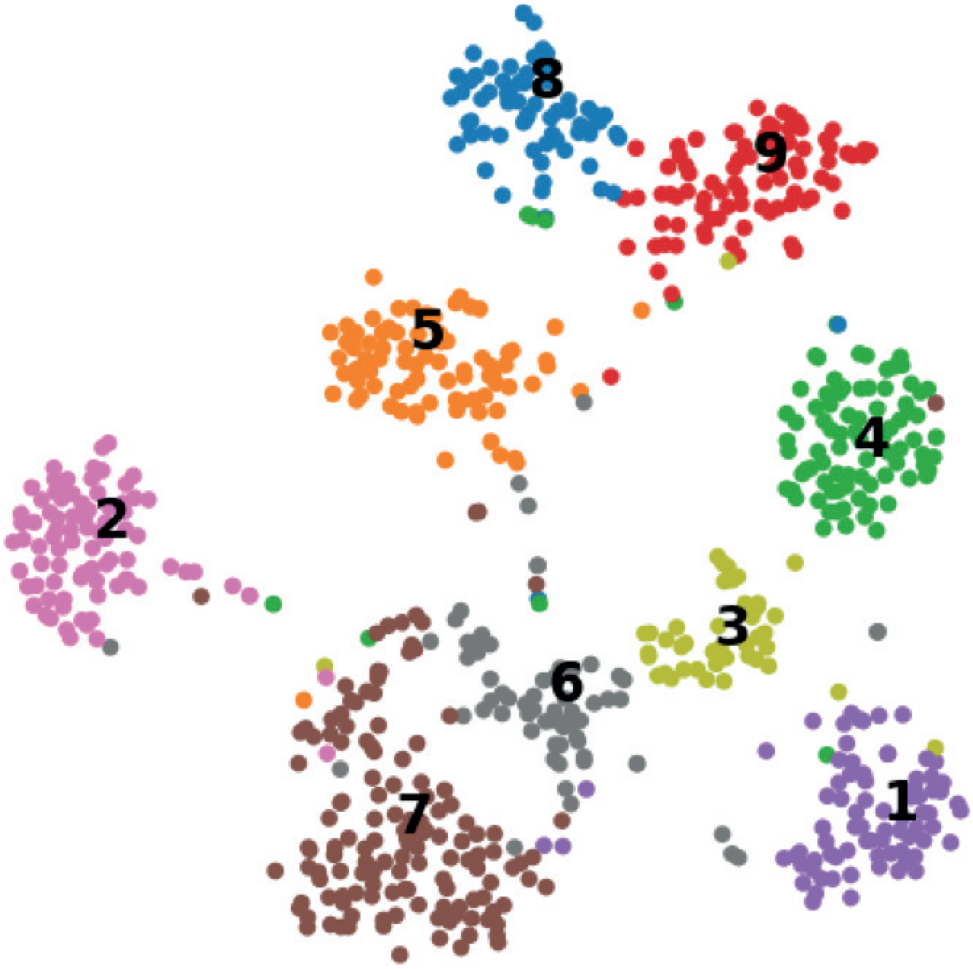


FIGURE 4.19: Cluster visualization of the different gesture numbers through t-SNE.

which correspond to the results for model size, maximum feature map, and latency, and are detailed in Table 4.7, as well as the test results for the K-Means unsupervised classification. As seen, the CAE outperforms the DAE in almost all metrics, except for latency where DAE is a little bit faster. The results for the model size and maximum feature map sizes are low enough to be able to embed the model in a low-resource microcontroller. After the optimization result and after training the KM model with the feature maps obtained, we obtain the test results for the unsupervised classification. In this case, the CAE model performs much better than the DAE, achieving a good result regarding the V-Measure. This indicates the feasibility of this method to provide unsupervised classification.

As the final step of our development, we further embed the best model with

CMSIS-NN in an NXP-S32K142 microcontroller to perform inference. To perform inference and obtain test results in the embedded implementation, we only need the encoder part of the CAE model and the centroids of KM. We proceed in the following manner: first and for each input, we obtain the feature vector by forwarding the input through the encoder, and second, we measure the distance to all the centroids and choose the nearest one to assign a class. It is important to note that, in CMSIS-NN legacy API, the quantization used is the power of two based and in the case of PyTorch is linear, enforcing thus a loss of precision. This changes the accuracy of the model but not the size since in both cases we use 8-bit integers. We obtain a V-Measure of 84.08%, hence resulting in a loss of around 3.104% in the V-measure, but also validating our embedded deployment.

Finally, to have a visual representation of the clusters, we embed with t-SNE [482] all the encoded features from the training set and also the centroids. As can be seen in Figure 4.19, the clusters are well separated identifying each one as a group of gestures corresponding to a specific number.

4.5 Conclusions

As this chapter has been organized separately in three projects, the conclusions drawn are separated as well.

First, we have shown that tiny RNNs can be used in embedded systems to process the data from low-cost capacitive sensors to recognize simple gestures such as touches and slides in various scenarios including the use of gloves and bare hands.

The use of execution platform constraints in the Bayesian Optimization process has been shown to be effective to reduce the optimization time. Thus, the multidimensional design space can be quickly analyzed until a valid design is found that meets the minimum requirements. Although further optimization is possible, the investment would be justified if it is either able to increase the accuracy of the system or reduce the platform requirements so that cheaper hardware platforms can be used providing additional savings.

Second, we have modified and extended the procedure and architecture introduced in [188], obtaining similar results regarding accuracy and size performance, but reducing significantly the search time. These results act as a validation of the general NAS procedure since the modifications were only based on the addition of lower fidelities and modification of sampling methodologies. By adapting the search space, including combinations of CNNs and RNNs, we have been able to develop state-of-the-art methods for a Gesture Recognition task. Further improvements could include an unconstrained or more flexible search space and a tolerance-stopping method.

As contributions stand out the implementation and opensourcing of the NAS system [218], as well as the implementation [436] of the Arc Kernel [276] for the library GPyTorch [437].

Finally, we have shown a full pipeline of work for unsupervised classification of gestures. We have employed a CAE plus KM as a model, and a multi-objective NAS to find a proper embeddable model. Finally, we have been able to embed that model into an embedded platform with CMSIS-NN, obtaining good performance results, and thus validating our approach. There are two next important steps to be able to improve the gesture recognition system. First, to change the quantization to linear instead of the power of two based to reduce the drop in performance and also to improve. Second, to improve the clustering by adding a Kullback-Leibler divergence component in the loss, hence separating more the feature vectors corresponding to different numbers [483].

As contribution stands out the development and open-sourcing of a library for automatic conversion and quantization of PyTorch models to CMSIS-NN format [219].

Chapter 5

Inverse Tone Mapping on Smartphones

In this chapter, we present the second application, inverse tone mapping (ITM), and its development for a specific hardware platform: mobile phones. ITM consists in extending, transforming, and adapting the contents of a standard dynamic range (SDR) image to a high dynamic range (HDR) environment. Such a process is complicated and computationally demanding due to the need for the hallucination of saturated values in the original SDR image.

SoA NN methods devoted to ITM have solved the problem by pursuing improvements in quality, yielding computing-intensive models only actionable in GPU or with long processing times; efficiency in inference or the possibility for deployment in resource-constrained environments have been mostly neglected. The current study presents a development for making NN-based ITM more efficient and bringing its deployment to mobile phones. To pursue such an objective we make use of different quantization schemes (Section 2.1), efficient operations (Section 2.4), and attention functions, proving how these mechanisms help make NNs more efficient and move them closer to resource-constrained devices.

The chapter is organized as follows. Section 5.1 introduces the topic of ITM and its context for NN and resource-constrained devices. Section 5.2 explores and presents studies and works of relevance. Next, Section 5.3, presents the proposed solution and all its components. Section 5.4 defines the experimental conditions, datasets and results obtained. Finally, Section 5.5 finishes the section by offering concluding remarks and observations, as well as providing future lines of work.

5.1 Introduction

HDR imaging enables capturing, storing, and displaying images and videos that cover the whole range of illuminance values present in natural scenes [484]. In contrast, low dynamic range (LDR) imaging technologies only work with a reduced range of values limited by their channel bit depth, thus producing results of inferior perceived quality [485, 486].

In the last two decades, HDR imaging methods have been applied in different fields and sectors, such as digital games [487, 488] and photography [489, 490], among others. Due to the lack of appropriate HDR displays, HDR content was converted to LDR. This operation, called tone mapping (TM) [491, 492, 493], pursues reproduction of images by reducing the tonal values within the images, which leads to loss of details and inevitable appearance changes in the reproduced images [485, 494]. Recently, there has been a substantial increase in production and commercialization of HDR displays. Sustained by new standards, such as HDR10, the number of HDR TVs shipped worldwide has leaped by more than a 10x factor between the years 2016-2019 [487]. In contrast, most content to be reproduced is still LDR, thus not attaining the reproduction capabilities of HDR displays.

This situation shows the need for the conversion of LDR images into HDR content. As mentioned previously, this process is named ITM and involves the recreation of the missing information, and expansion and adaptation of the available information to a higher bit depth. Handcrafted ITM algorithms [495, 496, 497, 498, 499] focused on range expansion to accommodate the LDR content to the new bit depth, as well as the application of tone operators to linearize the content and adjust the missing information. However, such methods did not provide sufficiently appealing results that could match originally produced HDR content [500].

Deep neural networks (DNNs) have been applied to ITM tasks [501, 20, 502] providing state-of-the-art results and products in industrial applications [503]. Nevertheless, current deep-learning-based ITM methods incur high computational costs, both in terms of memory and latency, due to the use of costly operations [504, 21], large models [22, 505] or feedback loops [29, 502], among other factors. Such burdens impede their deployment and usage in resource-constrained environments, such as edge devices and mobile phones.

To address this issue, we propose a novel Mixed Quantization Network (MQN) for computationally efficient mobile ITM by integrating different quantization

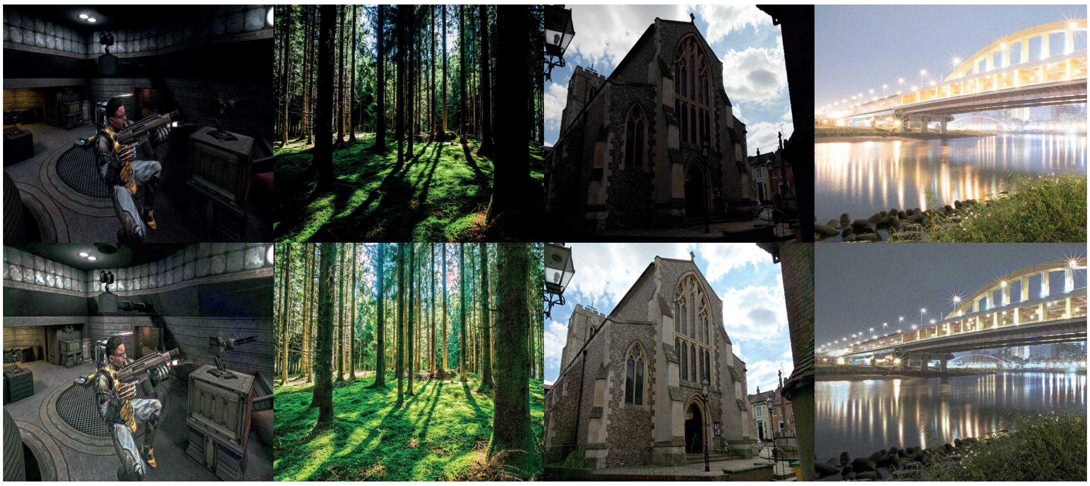


FIGURE 5.1: **Mobile ITM from single LDR images using mixed quantization network (MQN).** Employment of mixed quantization methods and efficient blocks in MQN help reduce the computational complexity of HDR image reconstruction (shown at the bottom row) from single LDR images (shown at the top row), and enable its deployment to mobile platforms, achieving a latency of ≈21ms with a Samsung Note 20 Exynos 990 MPSocC.

schemes applied to models, deploying efficient convolutions into models, and training models using input data. The proposed MQN enables attaining similar accuracy compared to state-of-the-art methods on a reference ITM dataset and can be deployed to a mobile platform to perform more computationally efficient inference. Further, it can be utilized by different hardware platforms, such as CPU or GPU, due to the flexibility of its components and structure. More precisely, our models can perform 10x to 100x times faster inference than competing methods (sample results are given in Figure 5.1). The present work, as far as the author is aware, is the first one devoted to constructing computationally efficient DNN-based ITM methods for mobile ITM tasks.

5.2 Background

Next, related work and concepts with regard to ITM, NN-based ITM methods, and NN efficient methods are reviewed.

Vanilla ITM Methods. Vanilla ITM methods have been implemented using tone operators [495], expansion algorithms [495, 506, 499], such as gamma curve expansion and over-exposed region enhancement [496], among other techniques [485, 498]. A great benefit, in general, of such methods, is their low computing power requirement. However, they struggle in the generation of high-quality image content on over- and under-exposed image regions [504].

DNN-based ITM methods. Recently, DNNs have been employed on ITM tasks. Although most of these methods blatantly differ regarding network architectures and model training components, they can be categorized into three main groups.

The methods in the first group [507, 22, 29, 21, 508, 509, 510, 20, 511, 512, 513, 514, 515, 516], which recreate a HdR image from a single SDR one, suffer from the problem of not accurately processing over- and under-exposed regions, but provide more compact systems. Methods in the second group use a group of bracketed over- and under-exposed images as input to directly learn to generate HDR output [517, 518, 519, 520, 521, 522, 523, 517, 524], as similarly utilized in photographic HDR generation. Methods belonging to these two groups differ mainly according to their network components, such as non-local blocks [520] and attention mechanisms [518]. Methods in the third group train models to generate a set of over- and under-exposed images from an LDR input, and then merge them to obtain an HDR image [502, 505, 525, 504]. Their main distinction from the other groups lies in the methods used to generate different exposures, such as deconvolution [504], sequential generation [505, 525], or recurrency [502].

Single Image HDR Reconstruction. In this work, we focus on single image HDR reconstruction. The task is substantially more challenging than multi-input HDR imaging. Among the first works on this task, [20] proposes using a U-Net type DNN to learn only representations of the over-exposed regions, while the rest of the image is only linearized through a default function. [21] use a three-branch network with different dilation ratios and sizes. Recently, [22] achieved the state-of-the-art by developing a model that reverses and unravels the camera pipeline to reproduce the final HDR, using a different DNN for each step, thus resulting in a computationally heavy system.

Computationally Efficient ITM Methods. Most DNN-based methods use heuristics that make them unsuitable for deployment in resource-constrained platforms. A common trick to enhance the training that burdens inference is using feedback loops [29, 502, 525], which are commonly used to generate differently exposed bracketed images. Another case is when different DNNs are stacked sequentially and/or in parallel [505, 22, 521]. There are also studies that make use of custom network blocks, such as non-local blocks [520] or 3D convolutions [504], which would hinder their deployment to mobile platforms [526].

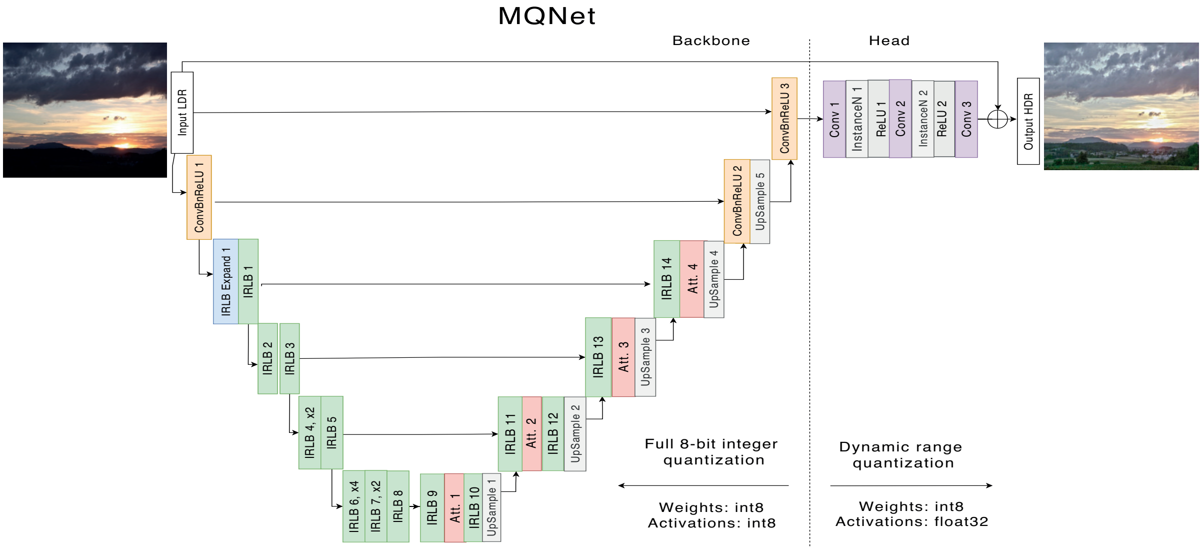


FIGURE 5.2: Illustration of the base backbone and high precision head that comprise the MQN. IRLB is used for fast inference and gated attention mechanisms [19] for improvement of feature representation learning accuracy. The dotted line indicates the separation between the fully quantized architecture and the dynamically quantized head. Input is added to the output of the head to produce the overall output.

5.3 A Mixed Quantization Network for Mobile ITM

The purpose of this work is to develop a learning-based ITM system with fast inference, especially devoted to the deployment of platforms with limited computational power. To this end, we propose a Mixed Quantization Network (MQN) by designing its architecture and components with state-of-the-art fast inference techniques such as quantization and efficient convolutions. To be able to accelerate inference while maintaining the required high precision output needed for ITM, we implement a mixed quantization (MQ) scheme as depicted in Figure 5.2. Moreover, we train models to learn multi-scale feature representations of HDR content over the input using a backbone network. The proposed MQN has two components: (1) a feature learning backbone network, endowed with full integer post-training quantization, and (2) a smaller network equipped with a head utilizing dynamic quantization to obtain a target precision for ITM.

5.3.1 Backbone and High Precision Head of MQN

Backbone

In order to learn multi-scale feature representations, we implement the backbone using a U-Net architecture with skip connections, which is also utilized in other single image ITM methods [22, 20, 504] due to its strong accuracy to speed trade-off compared to single-scale (resolution) networks [21]. As the encoder of the backbone, we use a MobileNetV2 (MBV2) [16] to obtain fast inference. We select a width factor $\alpha = 0.35$ and apply skip connections at the activations of layers $(1, 3, 6, 13)$ and output of batch normalization at layer 16. We extensively use IRLB blocks [15], which have a reduced computation cost[1] compared to vanilla convolutions, and can be used in different hardware platforms (GPU [258, 527], TPU [527], CPU [15, 16], NPU [528, 529]) with improvements in latency. To implement the decoder of the backbone, we favor upsampling in contrast to transposed convolutions, since the former is faster and does not produce artifacts [22].

To construct the decoder, after each upsampling we concatenate the upsampled feature maps and selected intermediate outputs of the encoder with skip connections. Following the concatenation, we add several IRLB blocks operating on the same resolution. Instead of IRLB blocks, the last two blocks of the decoder are composed of pointwise convolutions followed by batch normalization and ReLU activations. We design the whole network to have a reduced number of filters and convolutions compared to other U-Net structures (U-Net [530] has 7.76M parameters, U-Net++ [531] has 9.04M parameters and our model has about 1M parameters), thus reducing latency and memory consumption.

Attention mechanisms

We employ a gated attention mechanism (depicted by Att. in Figure 5.2) after IRLB blocks to improve performance. In the analyses, we explore three methods to implement attention: spatial attention (SA), channel spatial attention (CSA), and channel block attention (CA).

First, at the end of the first IRLB block in the decoder, we add Spatial Attention (SA) [376] gated blocks. We define SA blocks by

---

[1] The computational cost reduction from a vanilla convolution to a depthwise separable is of $\frac{1}{N} + \frac{1}{D_k^2}$, where N is the number of output channels and D_k is the size of the convolution kernel.

$$\mathbf{O}_f = ((\sigma_1 \circ C_1)(I_f)) \odot I_f \tag{5.1}$$

where O_f and I_f are the input and output respectively with f channels. C_1 denotes a convolution with a filter and kernel of size 1, σ_1 is a sigmoid activation, and $\circ$ indicates function composition.

Second, we add channel information through a depthwise convolution in parallel to the SA mechanism, which defines the channel spatial attention (CSA) mechanism at a given layer by

$$O_f = ((\sigma \circ D_f)(I_f)) \odot (((\sigma \circ C_1)(I_f)) \odot I_f) \tag{5.2}$$

where D_f is a depthwise convolution with f filters.

Finally, although with a higher computational cost due to pooling mechanisms, we also test channel attention (CA) blocks [19]. This operation, inspired by both residual layers and gated attention is defined by

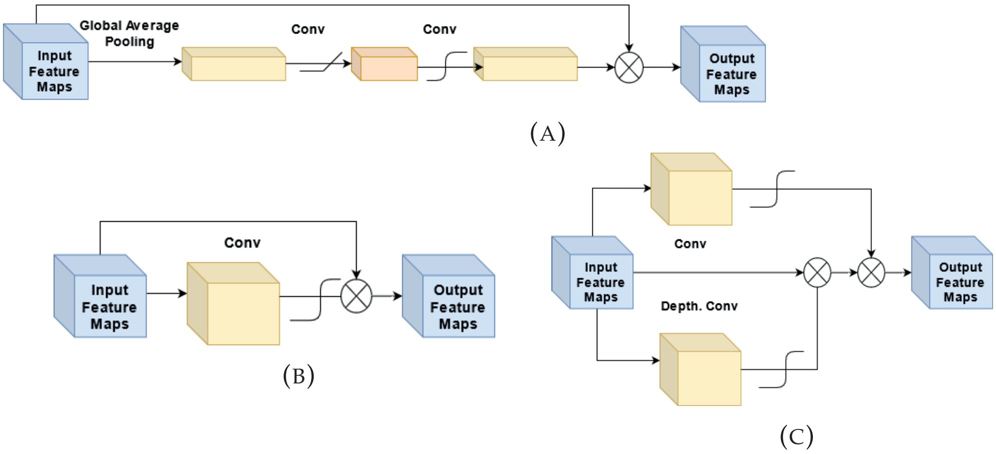
$$\mathbf{O}_f = ((\sigma_1 \circ C_f \circ \sigma_2 \circ C_{f'} \circ GP)(\mathbf{I}_f)) \odot \mathbf{I}_f \tag{5.3}$$

where C_f denotes convolution with f filters and kernels of size 1, where $f' = f \cdot r$ and r is the reduction ratio of the attention mechanism. Finally, σ_2 is the ReLU function and GP denotes global pooling.

In the analyses (Section 5.4.3), CA provides higher accuracy compared to SA and CSA. All such attention mechanisms can be seen as reduced one-head attention models without dense connections, thus being faster but also less powerful than those employed in, for example, transformer networks [112]. All three attention mechanisms are depicted in Figure 5.3.

High Precision Head

The head is in charge of recovering both the required detail and style of the input LDR image in the output HDR image. For this reason, the head is composed of three layers (convolution, instance normalization (IN) [532] and ReLU) and a residual connection with the original input of the model. Then, the head produces the final HDR prediction by $\hat{H} = \sigma(I + \phi(O))$, where ϕ denotes a hyperbolic tangent activation function, I is the input LDR image and O is the output HDR image of the system. We use ϕ to learn the nonlinear



(A)

(B)

(C)

FIGURE 5.3: Depiction of (a) Channel Attention (CA) block, (b) Spatial Attention (SA) and Channel Spatial Attention (SCA) block. *Conv* refers to a standard convolution, a soft sigmoid form denotes the sigmoid activation, the rectilinear symbol denotes ReLU activation and the $\otimes$ denotes element-wise product.

transformation between pixel values of the LDR and HDR images, and the purpose of using σ is to map to relative illuminance values, i.e. [0, 1] interval [21]. The entire MQN architecture is illustrated in Figure 5.2.

5.3.2 Mixed Quantization and Fusion

Full *8-bit* integer quantization [150] cannot be used directly in ITM, since a higher precision output (HDR image) is required. Hence, direct ITM methods [507, 22, 29, 21, 508, 509, 510, 20] cannot use full *8-bit* quantization and benefit from size and latency reduction on devices supporting only integer-valued operations. To address this problem, we define a mixed quantization scheme. The backbone of MQN is quantized to full *8-bit* integer quantization of both weights and activations, to obtain the most acceleration at inference time, thus opening the door to deployment in integer-only hardware accelerators, such as NPUs, but without restricting the application in other platforms, such as FPGAs [533] or GPUs [534]. Meanwhile, the remaining part of the network, the head, which produces output with the equivalent resolution to that of the input, is quantized by dynamic range post-training quantization [150]. That is, *8-bit* integer quantization is applied to the weights and 32-bit float activations are used in order to obtain the required high precision output for the mobile ITM tasks.

5.3.3 Loss functions

We train MQN models using the following loss functions of ground-truth and predicted HDR images H and $\hat{H}$:

- ℓ_1 loss function $\mathcal{L}_1(H,\hat{H}) = ||\hat{H} - H||_1$, where $\|\cdot\|_1$ is the ℓ_1 norm.
- ℓ_2 loss function $\mathcal{L}_2(H,\hat{H}) = ||\hat{H} - H||_2$, where $\|\cdot\|_2$ is the ℓ_2 norm.
- Cosine loss function $\mathcal{L}_{CS}(H,\hat{H}) = 1 - \frac{1}{K}\sum_{k=1}^{K}\frac{\langle \hat{\mathbf{h}}_k, \mathbf{h}_k\rangle}{||\hat{\mathbf{h}}_k||_2||\mathbf{h}_k||_2}$, where $\langle\cdot,\cdot\rangle$ denotes the inner product, and $\hat{\mathbf{h}}_k \in \mathbb{R}^3$ and $\mathbf{h}_k \in \mathbb{R}^3$ is the k-th pixel vector of the image $\hat{H}$ and H.
- As part of the problem is content generation, we include a perceptual loss in the form of a variant of the feature reconstruction (FR) loss function as in [535] to force the network to match the feature traits of the original HDR images. In this case, we use a VGG16 to produce the necessary feature maps to compute the FR loss by $\mathcal{L}_{FR}(\hat{H},H) = \sum_{i=1}^{3}\frac{1}{K}\sum_{k=0}^{K}|F_i(h_k) - F_i(\hat{h}_k)|$, where F_i denotes the output feature map obtained from the i^{th} pooling block of the VGG16 and $|\cdot|$ is the absolute value function.

Thus, the overall loss function used for training the MQN model is defined by

$$\mathcal{L}_{\mathcal{MQN}}(\hat{H},H) = \lambda_1\mathcal{L}_1(\hat{H},H) + \lambda_2\mathcal{L}_2(\hat{H},H) + \lambda_3\mathcal{L}_{CS}(\hat{H},H) + \lambda_4\mathcal{L}_{FR}(\hat{H},H), \tag{5.4}$$

where $\lambda_i > 0, i = 1,2,3,4$ are parameters used to balance range of loss functions.

5.4 Experimental Analyses

First, the experimental setup and evaluation methodologies are described, and implementation details such as further information on datasets, training, and evaluation procedures are given. Next, results from the architecture alternatives defined in Section 5.3 are presented. Finally, the proposed model is compared with state-of-the-art methods with extensive analyses regarding the accuracy, latency, and other measures.

5.4.1 Datasets

We build our training data from a collection of HDR image datasets [536, 537, 538, 539], consisting of 3768 HDR images, split into a training set of 3580 images and a validation set of 188 images. Most of these datasets do not contain unprocessed LDR images which can be used as input. We opt then for creating the LDR images through TM [21], that is, we apply a tone mapping operator (TMO) [492, 491, 493] to the original HDR images to produce LDR images. For testing and comparing with state-of-the-art methods, we use publicly available datasets, HDR-Eye [540], HDR-Real [22], and RAISE-1K [541]. These datasets contain LDR and HDR images, enabling a fair evaluation between methods.

Training Datasets

We employ 4 different datasets for training: Ward, Funt, PFSTools, and HDR-Plus. Most of them only contain high bit-depth images, and thus input LDR images have to be recreated using TM operations. Specifically, the four tone mapping operators (TMOs) used are: Drago [492], Mantiuk [491], Reinhard [493] and Exposure obtained from the OpenCV library [542]. For data augmentation, we use a batch of ground-truth HDR images. For each image, a TMO is chosen randomly and the TMO is applied to the image with random parameters.

Ward and PFSTools The Ward dataset [537] is a collection of 33 HDR images originally intended to compare different HDR formats (OpenEXR, Radiance RGBE, and XYZE, 24-bit and 32-bit LogLuv TIFF, and others). The PFSTools [538] is a collection of 8 HDR images of both outside and interior scenes.

Funt The Funt collection [536] is a set of 105 HDR images built by bracketing 9 differently exposed LDR images. The LDR images have a difference of 1 exposure value (EV) between them, a rate of capture of 5 seconds, and the f-stop is not fixed. The final HDR images are created from raw LDR bursts by alignment and filtering.

HDRPlus HDR+ [539] is a content enhancement pipeline dataset consisting on 3640 bursts (made up of 28461 images in total) resulting from the Google HDR+ system. The dataset also contains an intermediate DNG burst merge image and the 8-bit image resulting from the pipeline. We use the merged burst image in DNG format as our HDR output and do not use the 8-bit images defined as a result of the pipeline or the raw input images.

Test Datasets

For testing our method, we choose three datasets that, contrary to our training data, have LDR-HDR paired images, thus enabling a fair evaluation.

HDR Eye HDR Eye [540] consists of 46 LDR-HDR pairs taken with the Sony DSC-RX100 II, Sony NEX-5N, and Sony 6000 cameras. The HDR images are generated by combining LDR images with exposures (-2.7, -2, -1.3, -0,7, 0, 0,7, 1,3, 2, 2.7). Evaluation is performed using images with size 256x256 as suggested in [525].

HDR Real HDR Real [22] is a photographic dataset specially designed for extreme HDR contexts. It consists of 1838 LDR-HDR pairs taken by amateur photographers, employing 42 different cameras, using different exposures, and covering the whole range of lighting conditions: from near pitch-black to extremely saturated images. We perform evaluation using 256x256 images, following [525]. Instead of preprocessing the datasets as employed in [22], we employ the dataset directly without preprocessing.

RAISE 1K The RAISE-1K [541] consists of a subset of 1000 RAW-TIFF image pairs selected from the original RAISE dataset. Originally intended for digital forensics, it contains high-resolution images captured in diverse scenarios: indoor, outdoor, man-made, and natural. Following [22], we consider using unprocessed RAW images (with 12- or 14-bit depth) as ground truth HDR images, and the TIFF images as 8-bit LDR input images. We convert the RAW images to *.hdr* format and the TIFF images to JPEG. For evaluation we downsize images to a quarter of their original size respecting proportion ratios, resulting in images that are approximately 720p.

5.4.2 Experimental Setup

Accuracy measures

We measure the accuracy of methods using Peak Signal Noise Ratio (PSNR), Structural Similarity (SSIM) and HDR-VDP-2 [543].

For the evaluation of methods using Peak Signal Noise Ration (PSNR) and Structural Similarity (SSIM), we use tone-mapped images and predictions. Inspired by [22], and unlike our training procedure where we use OpenCV tone mapping operators, we tone map images using four tone mapping operators from the Photomatix suite [23] for evaluation: (1) detailed, (2) balanced, (3) realistic and (4) photographic.

Regarding HDR-VDP metric evaluation, we use version 2.2.2 for evaluation. Our predictions take values from [0, 1] (relative luminance). We rescale them to a display range of $1000\,cd/m^2$ and align the 0.01 and 0.99 percentiles of both prediction and ground truth. For a fair comparison, we use the same parameters as utilized in [505, 525] to obtain the *pixel-per-degree* parameter, which are 24-inch display, 0.5 distance and 1080p display resolution.

With regards to latency measurements, we test models on both a desktop GPU, NVIDIA GTX 1080 Ti, and a mobile platform, Samsung Note 20 Exynos 990. The latency calculations are performed on the desktop platform taking into account the process between the reading of the LDR image and the output of the final HDR image, that is to say, the reading and writing computational costs are not considered. So, for those methods, such as [20] or ours, that use the input mixed with the output to create the final HDR image, the combination procedure is also included in the latency computation. In the mobile platform, latency is tested through the native benchmark application for the arch64 architecture offered by Tensorflow [544], always running on a CPU with 4 threads, using images of size 256x256, and results averaged over 300 runs.

Training parameters

The entire training takes approximately 5 days on a computer with an Intel Core i7-6850K and an Nvidia GTX 1080 Ti. We use the Adam optimizer with an initial learning rate of 5×10^{-5}, a decreasing learning schedule with a decay factor of 0.99 applied at every 4 epochs, and a batch size of 4. In the analyses, the best results are obtained using $\lambda_1 = 1$, $\lambda_2 = 1$, $\lambda_3 = 0.1$ and $\lambda_4 = 0.05$ for integrating loss functions.

5.4.3 Ablation Studies

In this section, we study ablations with regard to various attention mechanisms, quantization schemes, loss functions, and deployment of MQN to hardware platforms for efficient mobile ITM.

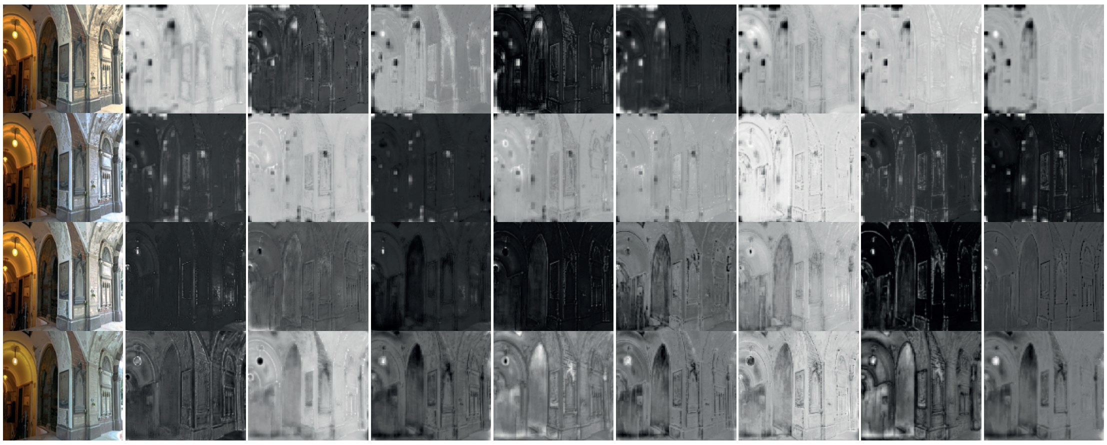


FIGURE 5.4: Illustration of feature maps learned at the Att. 4 layer of the MQN depicted in Figure 5.2. The rows show in order results obtained without using an attention mechanism, followed by using SA, CSA, and CA. The first column shows the predicted HDR image $\hat{H}$ and the rest shows the feature maps learned at different channels of the Att. 4 layer.

TABLE 5.1: Analyses of accuracy and latency measurements (on the SN20E990) for different attention mechanisms. None indicates that no attention mechanism is used and B indicates backbone. Blue and red indicate the best and the second-best accuracy, respectively.

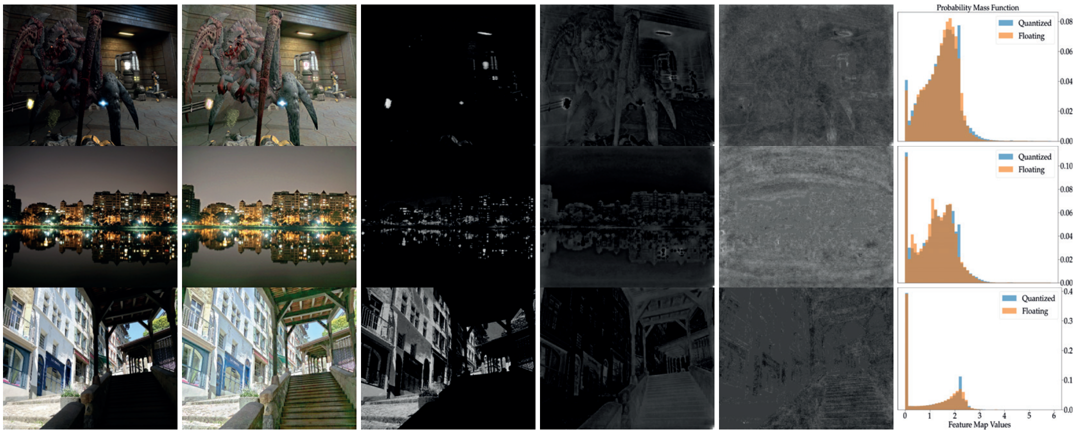
Attention	**Latency B (ms)**	***PSNR-TM***	***SSIM-TM***
None	11.55 ± 0.27	20.76 ± 3.21	0.8440 ± 0.0741
SA	11.68 ± 0.49	21.12 ± 3.03	0.8333 ± 0.0800
CSA	12.17 ± 0.25	20.75 ± 3.06	0.8559 ± 0.0597
CAB	12.40 ± 0.32	21.25 ± 3.11	0.8782 ± 0.0520

TABLE 5.2: Results obtained using different quantization schemes. Latency (L.) is measured on the deployment platform (SN20E990).

Backbone	**Head**	**L. Backbone** (ms)	**L. Head** (ms)	***PSNR-T***	***SSIM-T***
Quant.	Dynamic	11.52 ± 0.82	9.63 ± 0.12	21.25 ± 3.11	0.8782 ± 0.05
Quant.	Float32	11.52 ± 0.82	20.95 ± 0.99	21.34 ± 3.08	0.8793 ± 0.05
Float32	Float32	21.13 ± 0.42	20.95 ± 0.99	21.58 ± 3.14	0.8727 ± 0.05
Int 16	-	857.67 ± 18.2	-	17.23 ± 3.87	0.7612 ± 0.1016

Attention Mechanisms

We explore different attention mechanisms specified in Section 5.3.1. Results given in Table 5.1 show that accuracy increases when moving from SA to CAB, which provides the best accuracy with an increase of 3% on SSIM, albeit with



a) I. b) $\hat{H}$. c) f_1. d) f_2. e) Δf_2. f) PMFs.

FIGURE 5.5: Analyses of the quantized $Q(\mathbf{f})$ and floating point $F(\mathbf{f})$ features $\mathbf{f}$ learned by the MQN at the ConvBnReLU3 layer at Figure 5.2 using a) three sample input images with b) predictions $\hat{H}$. Visualization of (c) the first channel f_1 and (d) the second channel f_2 of $\mathbf{f}$, (e) the difference map $\Delta f_2 = \|Q(f_2) - F(f_2)\|_1$. We show the probability mass function (PMF) of $Q(f_2)$ and $F(f_2)$ in (f).

an increase in latency of ≈ 1 ms. We examine the features learned using different attention mechanisms in Figure 5.4. We observe that better feature representations of edges and surfaces are learned when models are trained using CSA and CA. For instance, in both cases, the lamp is well captured with large feature activation values, enabling the dimming effect in the prediction.

Quantization Schemes

Next, we study the behavior of features $\mathbf{f}$ learned at the interface between the backbone and the head (i.e., at the ConvBnReLU 3 layer depicted in Figure 5.2), as well as how quantization affects the interface and the head. In Table 5.2, we analyze how the performance and latency of models change for different quantization methods. The results show that the proposed quantization scheme enables us to obtain similar PSNR/SSIM accuracy whilst showing improvements in latency.

In order to analyze the effect of quantization on statistical properties of features, we compute histograms approximating probability mass functions (PMFs) of features and their quantized versions. The results given in Figure 5.5.e show that distributions of quantized features $Q(\mathbf{f})$ and features with floating point values $F(\mathbf{f})$ have similar distributions. This result suggests that the quantization scheme preserves statistical information on features.

TABLE 5.3: Analyses of accuracy for different loss combinations.. Blue and red indicate the best and the second-best accuracy, respectively.

Loss	***PSNR**-TM*	***SSIM**-TM*
(i) $\mathcal{L}1$	20.00 ± 3.10	0.8295 ± 0.0646
(ii) $\mathcal{L}_1, \mathcal{L}_2$	19.96 ± 3.05	0.8281 ± 0.0656
(iii) $\mathcal{L}_1, \mathcal{L}_2, \mathcal{L}_{CS}$	20.22 ± 3.02	0.8268 ± 0.0625
(iv) $\mathcal{L}_1, \mathcal{L}_2, \mathcal{L}_{CS}, \mathcal{L}_{FR}$	21.25 ± 3.11	0.8782 ± 0.0520
(v) $\mathcal{L}_1, \mathcal{L}_{FR}$	21.19 ± 2.85	0.8261 ± 0.0832
(vi) $\mathcal{L}_2, \mathcal{L}_{FR}$	21.53 ± 2.92	0.8343 ± 0.0675
(vii) $\mathcal{L}_1, \mathcal{L}_2, \mathcal{L}_{FR}$	21.54 ± 2.85	0.8538 ± 0.0597

We also study what has been learned in the interface as well as the effect of the network on a computer graphics image. In Figure 5.5 (c and d), we present two feature maps f_1 and f_2 corresponding to two channels of $\mathbf{f}$. The maps show that two very different representations are learned in these channels: in (c) light sources are identified, striking their relevancy for the ITM task, while in (d) general edge structures are learned. Moreover, we can see in the first row, as well as in Figure 5.1, that the network can be applied to computer graphics images without having special artifacts or distortions, opening the door for employment of MQN in computer graphics.

An alternative solution to the MQ scheme, already available in some network optimization suits [545], is quantizing network parameters to 8-bit integers and activations to 16-bit integers. We compare both schemes and present the results in Table 5.2 (first and last, rows) using the same network with the exception of the IN blocks which have been substituted by Batch Normalization blocks due to incompatibilities in the inference framework. The results show that there is a substantial loss in accuracy when the *int*-16 scheme is used. In the case of latency, we observe a substantial increase in latency, probably due to an issue of a lack of suitability of such quantization schemes with the deployment platform or the lack of suitable implementations in the deployment suite. All in all, aside from latency measurements, analyses of the accuracy of models show the benefits of our MQN in contrast to the aforementioned scheme which has 8-bit integer parameters and 16-bit integer activations.

Loss functions

We analyze the effect of using different loss functions defined in Section 5.3.3 in Table 5.3. The results show that employing $\mathcal{L}_{FR}$ with $\mathcal{L}_{CS}$ and $\mathcal{L}_2$ increases

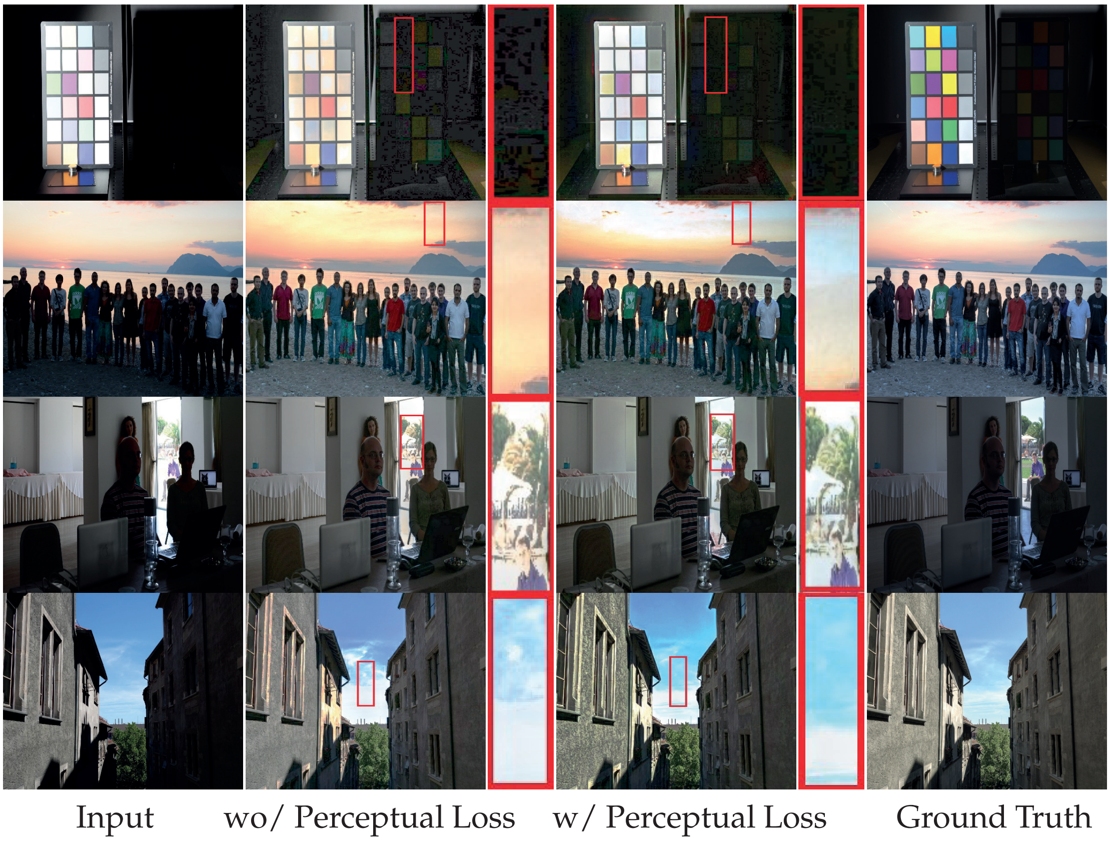


FIGURE 5.6: Visual analyses of the effect of training models with and without using $\mathcal{L}_{FR}$ on predictions $\hat{H}$. It helps models gain structural coherency and improve color details.

accuracy substantially. Meanwhile, $\mathcal{L}_{CS}$ and $\mathcal{L}_2$ seem to provide a slightly negative effect on their own.

Moreover, we analyze qualitatively the effect of perceptual loss in the network results, as illustrated in Figure 5.6. As the results show, training models using perceptual loss helps learn feature representations of color coherency as well as color details, further enhancing the quality of the image. Examples are the structural coherency in the color-checker (row 1) or the sky color coherency (row 4).

To provide additional information about the training process, in Figure 5.7 we show the training and validation loss evolution of all combinations of losses found in Table 5.3.

Deployment Platforms

In the experimental analyses, we used CPUs as our main deployment hardware platform. However, as our objective has been to develop a well-performing

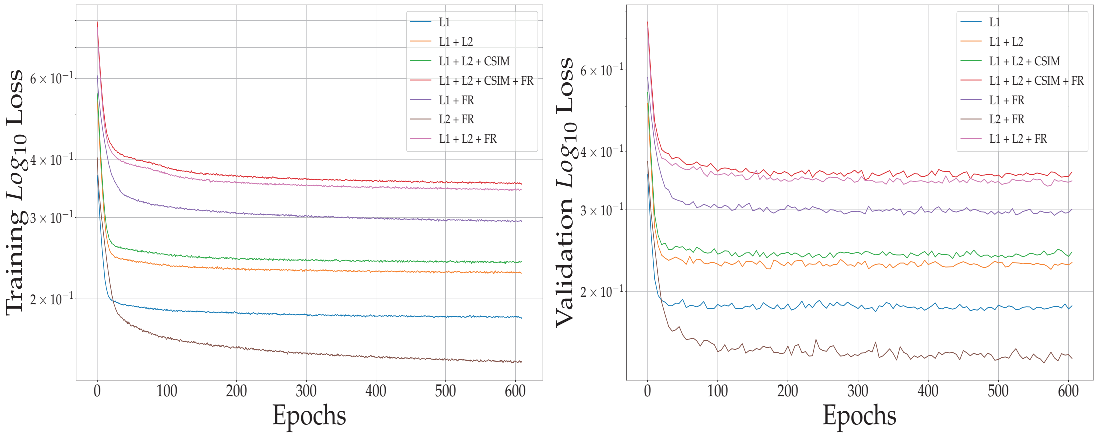


FIGURE 5.7: Training (left) and validation (right) loss plots for each loss combination established in Table 5.3.

TABLE 5.4: Comparison with other state-of-the-art single image HDR reconstruction methods. Performance metrics and latency values were reproduced with the same evaluation criteria and original codes. Blue and red indicate the best and second-best accuracy. P. indicates the number of parameters, L. M. indicates latency for mobile CPU, M. RAM the maximum RAM memory consumed by the model, and O. the number of operations in multiply-accumulate units. Performance values are given in HDRVDP-Q score. *FHDR[29] uses recurrence: the present value is computed taking into account two iterations.

Model	P. (M)	L. GPU (ms)	L. M. (ms)	O. (GMAC)	M. RAM (MB)	HDR-Eye	Raise-1K	HDR-Real
HDRCNN [20]	29.44	247	-	30.35	-	51.16 ± 4.43	51.89 ± 2.77	45.56 ± 8.18
SingleHDR [22]	29.01	976	-	112.75	-	53.05 ± 5.08	51.69 ± 2.56	48.72 ± 4.03
FHDR [29]	0.571	54	4970 ± 434	72.34*	832.40	51.41 ± 6.72	53.13 ± 1.71	45.82 ± 8.67
HDRUnet [546]	1.651	17	808 ± 22	23.42	353.46	50.32 ± 4.07	51.42 ± 3.35	44.60 ± 7.30
ExpandNet [21]	0.45	21	474 ± 7	13.66	262.77	50.52 ± 3.94	51.83 ± 1.68	44.86 ± 8.21
DeepHDR [547]	51.545	17	238 ± 4	18.94	251.17	51.11 ± 4.45	51.66 ± 2.79	45.81 ± 8.34
TwoStage [513]	1.088	32	3338 ± 456	54.91	397.41	49.68 ± 3.7	52.95 ± 2.36	43.46 ± 7.55
Ours (best)	0.928	11	21 ± 1	0.5	9.45	51.59 ± 4.61	51.81 ± 1.56	45.15 ± 8.15

but efficient model employing mixed quantization, attention, and efficient operations, our model can also be extended to other hardware platforms. For this reason, we also deploy our model to the GPU (Arm $Mali^{TM}$-G77 MP11) of SN20E990, elucidating the flexibility of our model with regards to the deployment of MQN models on platforms with different hardware configurations. We obtained a latency of 10.5 ms for our backbone and 9.3 ms for our high precision head, improving our results even further and showcasing the hardware flexibility for the implementation of our MQN.

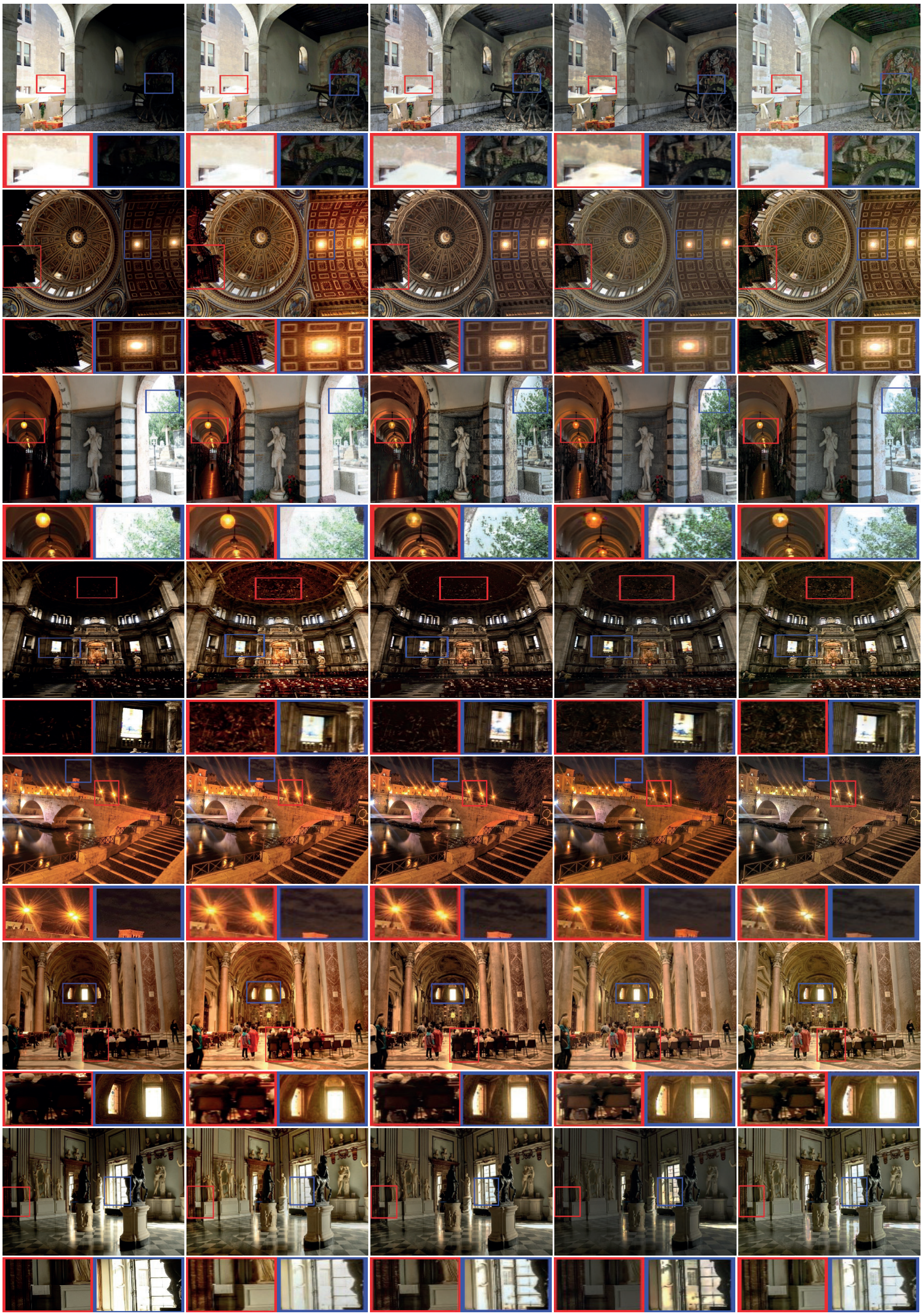


FIGURE 5.8: Visual comparison of results obtained using, from left to right; input LDR images, HDRCNN [20], ExpandNet [21], SingleHDR [22] and our proposed MQN. All images are produced with Balanced TMO from the suite Photomatix [23], similarly to [22].

5.4.4 Comparison with State-of-the-art Methods

In Table 5.4, we compare the accuracy, latency, and size of models trained using MQN and state-of-the-art methods. We selected competing methods according to two criteria: (1) We considered methods that promised a reasonable accuracy-latency trade-off [546, 547] due to their simplicity, to evaluate the efficiency gain introduced with our method. (2) To estimate the cost of adding efficiency as a factor in network design, we compared MQN to state-of-the-art or baseline methods [22, 20] that did not take accuracy-efficiency trade-off into account. The results show that MQN models provide accuracy (HDRVDP-Q score) on par with the larger state-of-the-art models and with a notable reduction in latency and RAM consumption.

In terms of latency, our MQN provides the fastest model, both in experiments conducted on GPU and mobile deployment platforms. Comparison of the latency of the models on the GPU platform is not fair, since our MQN model uses *8-bit* integer quantization and employs depthwise convolutions which are not suited for GPU platforms [16]. To obtain a fair comparison on the mobile deployment platform, we adapt the four methods HDRUNet [546], ExpandNet [21], DeepHDR [547] and TwoStage [513] that perform the fastest inference on the GPU. The results show that the difference between the latency of our MQN models and these four models increases further on the mobile platform. More precisely, our MQN model is running in real-time (21ms) while others run from a quarter of a second (DeepHDR with 238ms) to more than 3 seconds (TwoStage with 3338ms). The same trend is observed with memory consumption (maximum RAM MB) where our model stands as the most efficient with a reduction of a factor of x26 or more. The main factor for such differences is the employment of our MQ scheme with computationally efficient components, such as IRLB blocks. These facts, along with learning representations of HDR content over the input, enable us to obtain a faster model with similar accuracy. However, other models use methods that hinder efficient deployment, such as by utilising images with same resolution [21, 513], inefficient network composition [22], convolution operations and special blocks [546, 547].

In Figure 5.8, we compare the ITM models visually. In the analyses, our MQN model performs well with good quality in under-exposed regions. For instance, our MQN model recovers details better than others as seen in the detail of the image in row one. In the case of over-exposed regions, our model can recover details better than HDRCNN [20] and similarly to ExpandNet [21]

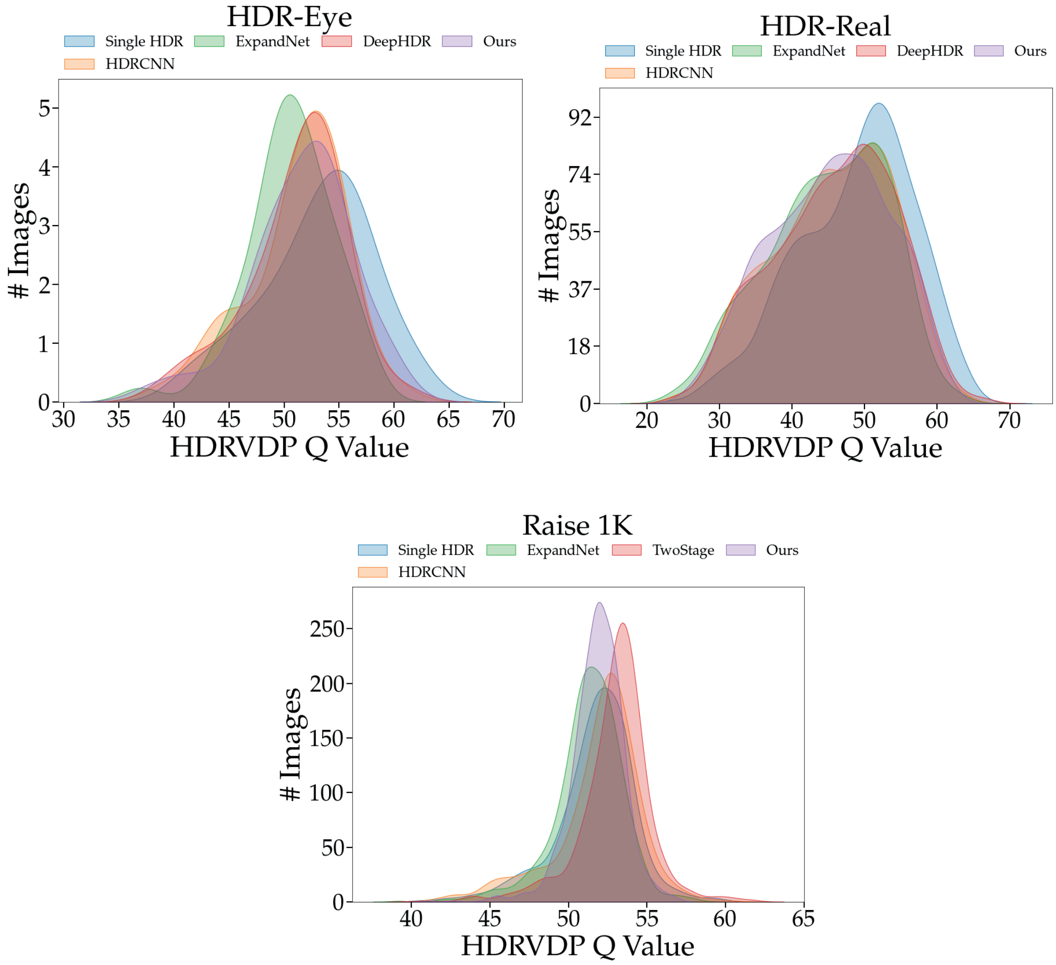


FIGURE 5.9: Distribution of HDRVDP Q score values for three test datasets and 5 best competing methods in each case.

as seen in the lamp, tree, and tent details. Although SingleHDR [22] performs slightly better on over-exposed regions, the SingleHDR model has 29 M parameters and takes almost a second to perform inference on a desktop GPU, while our model is 29x times smaller and almost 100x faster.

Extended State-of-the-art Comparison

Histogram of HDRVDP-Q values In Figure 5.9, we plot the distribution of the Q value scores obtained from HDRVDP for HDRCNN [20], SingleHDR [22], ExpandNet [21], and our method for all three test sets: HDR-Eye, HDR-Real and Raise 1K. As the results show, our method performs similarly or better than HDRCNN and ExpandNet, and slightly worse than SingleHDR, even though our method is purposed for fast inference and not for performance quality, as other competing methods are.

TABLE 5.5: Comparison with state-of-the-art single image HDR reconstruction methods for PSNR and SSIM performance metrics. Values reproduced with the same evaluation criteria and original code. Blue indicates the best and red indicates the second best accuracy.

	HDR Eye		HDR Real		Raise 1K	
Model	**PSNR**$-T$	**SSIM**$-T$	**PSNR**$-T$	**SSIM**$-T$	**PSNR**$-T$	**SSIM**$-T$
HDRCNN [20]	18.82 ± 3.49	0.7754 ± 0.1044	16.53 ± 5.65	0.6378 ± 0.2303	17.25 ± 2.81	0.5950 ± 0.1213
FHDR [29]	20.30 ± 5.40	0.7794 ± 0.1897	16.47 ± 5.83	0.6436 ± 0.2305	17.68 ± 3.67	0.5653 ± 0.1315
ExpandNet [21]	19.85 ± 2.96	0.7854 ± 0.0954	16.24 ± 5.99	0.6175 ± 0.2564	16.58 ± 2.77	0.5264 ± 0.1306
SingleHDR [22]	21.70 ± 4.50	0.8259 ± 0.1244	21.16 ± 5.33	0.7409 ± 0.2150	15.12 ± 3.44	0.5688 ± 0.1253
DeepHDR [547]	19.38 ± 3.38	0.7723 ± 0.1131	16.49 ± 6.15	0.6252 ± 0.2591	17.22 ± 2.86	0.5861 ± 0.1243
TwoStage [513]	17.97 ± 4.16	0.7519 ± 0.1057	14.57 ± 4.52	0.5660 ± 0.2235	19.12 ± 2.81	0.6162 ± 0.1257
HDRUnet [546]	18.58 ± 4.11	0.7724 ± 0.1139	14.59 ± 4.73	0.5778 ± 0.2478	17.70 ± 3.07	0.5989 ± 0.1263
Ours Best	19.97 ± 4.21	0.7990 ± 0.1160	15.95 ± 5.76	0.6162 ± 0.2436	17.71 ± 2.73	0.5776 ± 0.1150

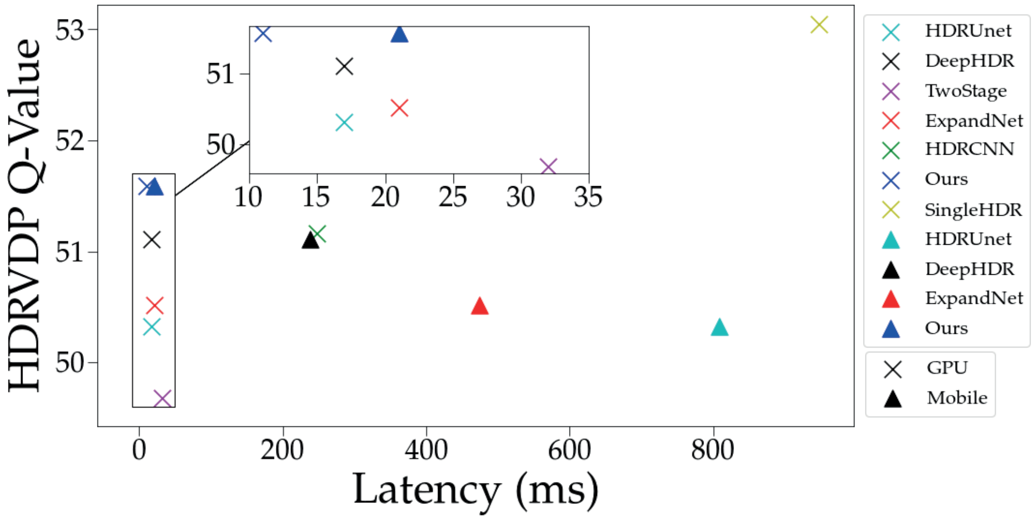


FIGURE 5.10: Latency and accuracy trade-off comparison with the HDR Eye dataset between competing methods and our proposed solution. Accuracy is measured by the HDR-VDP Q value score. Latency is computed both in the mobile (CPU) and GPU platforms.

Quantitative Evaluation on Tone Mapped Images We also evaluate our method and compete for state-of-the-art methods using PSNR and SSIM. To provide HDR images for these metrics, we tone map the images with the use of the Photomatix suite and four tone mapping operators: Detailed, Balanced, Realistic, and Photographic. Results are shown in Table 5.5. The results show that our method performs better than ExpandNet and HDRCNN in HDR Eye and Raise, and always after Single HDR. However, note that our method is 100x faster than SingleHDR both on mobile and GPU, and almost 10x faster than ExpandNet on mobile.

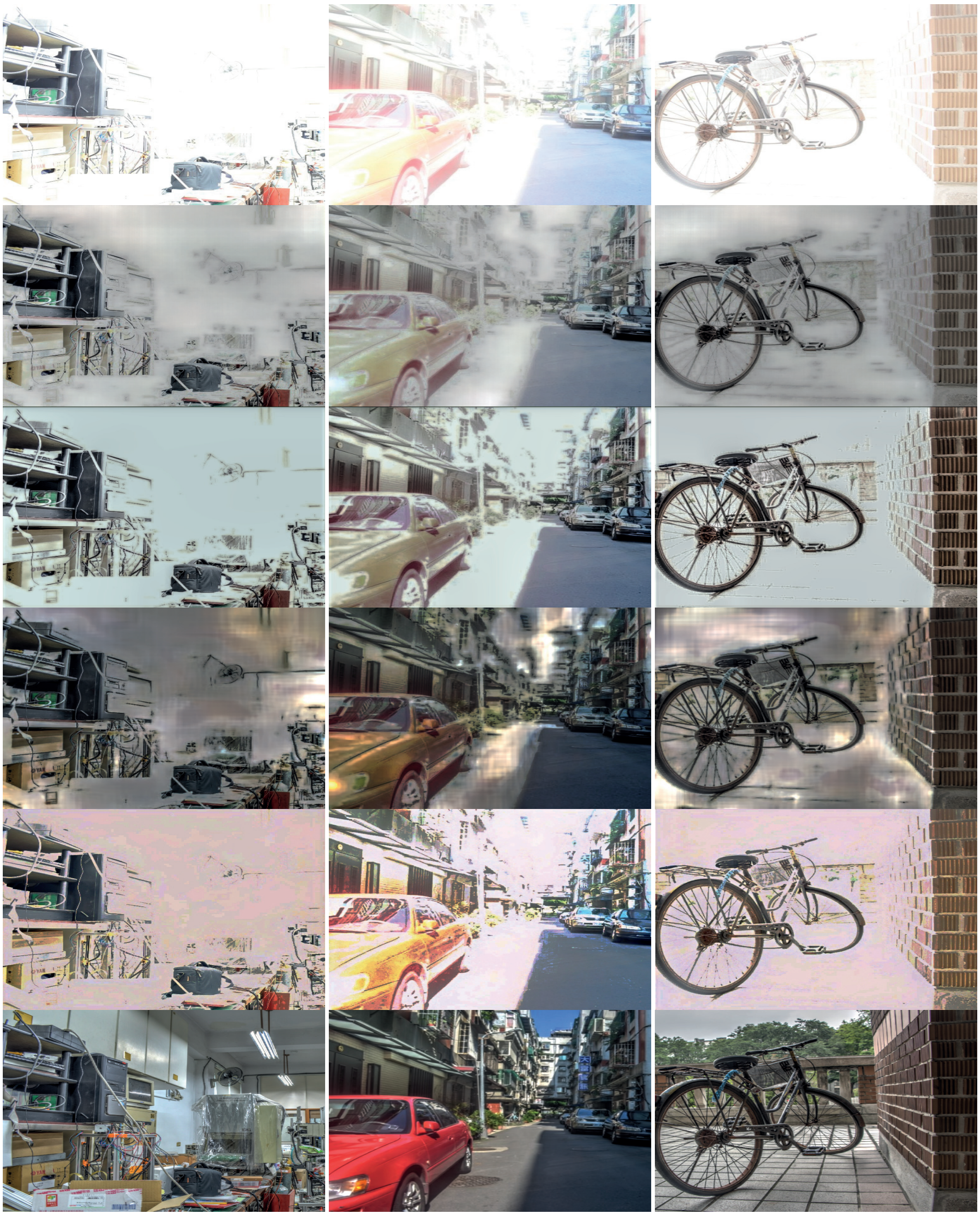


FIGURE 5.11: Three examples of extreme cases of overexposure in HDR Real dataset. Order from top to bottom is: input, HDR-CNN [20], ExpandNet [21], SingleHDR [22], our method and ground truth.

Comparative Analysis of Latency and Accuracy Our method is intended for fast inference, while all competing methods focus on accuracy. We compare the latency and accuracy trade-off in Figure 5.10. We can see that our model is almost 100x faster than SingleHDR while providing only 3% accuracy loss, and while being 10x faster than ExpandNet it achieves a 1 Q point more on accuracy.

Analyses for Extreme Over-exposed Cases In Figure 5.11, we illustrate the difficulty of ITM for some of the samples belonging to the HDR Real with three extremely over-exposed inputs. As seen, all networks struggle to recreate the ground truth, failing in most of the content.

5.5 Conclusion

In this work, we proposed a novel DNN-based method called Mixed Quantization Network (MQN), for computationally efficient ITM. The proposed MQN has produced competitive accuracy with better computational efficiency compared to the state-of-the-art, being the first DNN-based single image ITM method targeting computationally efficient mobile ITM.

We have proven how with the use of quantization (Section 2.1), specifically a mixed quantization scheme, and efficient operations (Section 2.4), IRLB blocks, we can reduce the computational requirements of models. Moreover, we also have proven, by deploying our framework to CPU and GPU mobile platforms, that such optimizations can move the models closer to efficient inference on hardware platforms that *a priori* could not handle such models.

Nevertheless, although we have attained similar performance to SoA models while being more efficient, there are still more steps to push efficient inference further. Examples could be using distillation, structured pruning, or NAS in order to improve latency and accuracy trade-off for mobile ITM.

Chapter 6

Bronchoscopy Navigation

6.1 Introduction

Early detection is fundamental for lung cancer mortality reduction [548, 549]. After a suspicious pulmonary lesion (PL) has been detected through a computed tomography (CT) scan, a decisive diagnosis can only be achieved through a biopsy. Recent advances in sensors and imaging have improved the sensitivity yield of navigational bronchoscopy (NB) [550, 551], establishing it as a solid alternative to percutaneous approaches, which have a higher degree of medical complications [552, 553].

Among the different methods for NB, vision-based NB (VNB) stands out for its low cost, accessible configuration, and reliability. In such a method, a virtual model of the patient pulmonary system is built from CT scans [554, 555, 556] and the optimal path to the PL is defined. The physician should replicate this path during the interventional bronchoscopy. Therefore, during navigation, VNB models require the registration of the virtual lung model with the frames from the video bronchoscopy to provide effective guidance during the biopsy. The registration can be achieved by, either tracking the position and orientation of the bronchoscopy camera [557, 558, 559, 560], or by calibrating its deviation from the pose (position and orientation) simulated in the virtual lung model.

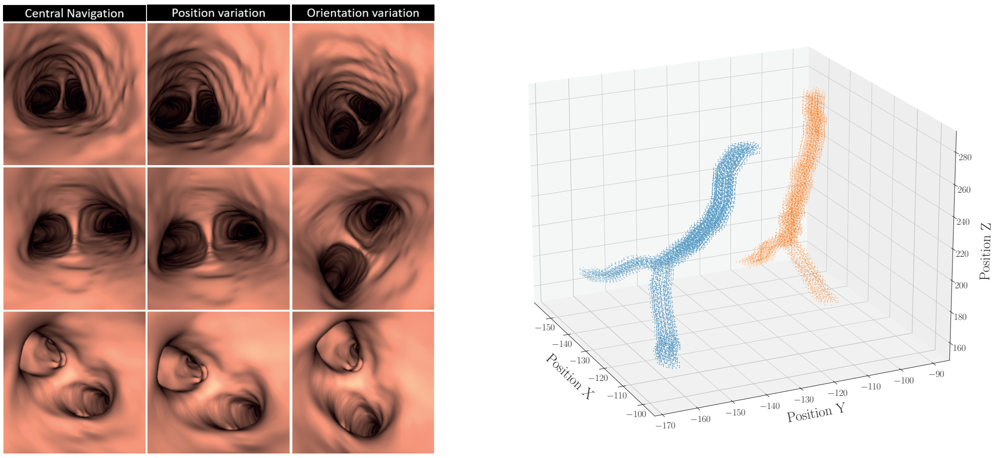
Traditionally, the problem of image-based tracking in VNB has been solved through geometric and hand-crafted methods. Feature generation [557, 558, 580, 560] and similarity measures [571, 572, 569, 30] were commonly used for such purpose, accounting, however, with tracking errors and large execution times.

Method	Year	Image Size	Tracking Type	PE (mm)	AE (°)	CTF (%)
[561]	1998	100x100	Local/Global	2	5	-
[562]	2001	-	Local	-	-	79
[563]	2002	-	Local	-	-	-
[564]	2002	410x410	Local	-	-	73.37
[565]	2004	454x487	Local	3 ± 2.26	2.18 ± 1.63	-
[566]	2004	-	Local	-	-	77.79
[567]	2006	30x30	Local	-	-	76.4
[568]	2009	-	Local	-	-	-
[30]	2010	-	Global	-	-	89
[559]	2011	362x370	Local	-	-	83.2
[569]	2011	30x30	Local	-	-	70.2
[570]	2012	362x370	Local	3.72	10.2	-
[571]	2014	256x263	Local	4.5	12.3	-
[572]	2015	487x487	Local	8.48 ± 6.29	-	-
[573]	2016	-	Global	-	-	-
[574]	2017	50x50	Local	1.5	-	-
[575]	2017	-	Local	5-15[1]	-	-
[24]	2019	-	Local	2.4	3.4	90.2
[576]	2019	307 × 313	Local	3.18 ± 2.34	-	-
[577]	2020	256x256	Local	1.17	9.71	-
[578]	2020	440x440	Local	3.02	-	78.1
[579]	2021		Local	6.2 ± 2.9	-	-

TABLE 6.1: Comparison among bronchoscopic tracking studies with regards to data and evaluation characteristics. Notably, none of the methods share a dataset (currently there is no publicly available dataset for this task) or publish their code. Moreover, metrics, although aiming to measure the same quantities, are different or lacking in some cases. Metrics shown are umbrella terms for measuring the position error (PE), angle error (AE), and the number of correctly tracked frames. Tracking type [30] refers to the type of information provided by the tracking: global type positions the bronchoscope in macro terms, e.g. 3rd bifurcation, while local, give information with regards to position and angle of the bronchoscope.

Recently, supervised data-intensive learning methods, such as neural networks (NNs) [581, 574, 577, 576], have been used for localization and tracking in bronchoscopies, providing better results than previous methods. Moreover, temporal learning techniques have recently been applied to other endoscopic modalities [582] but have not been appropriately tested in bronchoscopy. Additionally, depth information has lately been extensively used to improve tracking [583, 579, 576, 584], mixing it with generative neural networks [577, 576, 584, 579].

Despite these advances, there is a common obstacle among studies: results



(A)

(B)

FIGURE 6.1: Details of the synthetic dataset for bronchoscopy tracking and calibration. 6.1a, an example of synthetic frames from a trajectory from patient P18's lower left lobe. 6.1b, positions visited by the trajectories corresponding to two different patients: P18 and P20. Both cases only show points pertaining to the lower right and left lobe trajectories.

are affected by a lack of fair comparability due to the absence of public bronchoscopy datasets and the usage of appropriate metrics as a gold standard, Additionally, learning methods depend on high data availability, often hindering their application in data-scarce environments. Table 6.1 summarizes such a situation from bronchoscopic literature: none of the selected studies details public code or data as to allow for a fair comparison. With regards to metrics, position and angle metrics differ or lack in most studies, making further comparison difficult.

In such a situation, with great progress but also a lack of comparability, is of outstanding importance to establish ground points for enabling advances. In the case of a system for pose estimation in intervention guiding, it should address 3 key points: 1) definition of the most appropriate metric and comparison protocol for the evaluation of the estimated pose; 2) determination of the most accurate strategy for the processing of temporal information and 3) the highest generalization level (single or across subject) of models.

Hence, the goal of this paper is to analyze the performance of different deep learning approaches for image-based bronchoscopy tracking using a standardized and fair comparison framework. In particular, we contribute to the following aspects:

- **Synthetic Dataset.** We present a bronchoscopy navigation synthetic dataset based on real anatomies to enable fair comparison among methods with a cross-subject setting analysis, as well as, address the data requirements of learning methods.
- **Evaluation Protocols.** A study and comparison of rotation and position losses and metrics (including a novel one) for bronchoscopy navigation, which helps to establish better grounds for training and evaluation.
- **Processing of Temporal Information.** We investigate different solutions for neural network temporal learning. Models such as recurrent NNs (RNNs) [77, 585], pseudo-3D convolutions [586] or 3D convolutions [587], could exploit temporal information in bronchoscopic videos, currently not explored, and provide new results.
- **Population Modelling.** We analyze the different options with regards to data usage for industrial applications: first, with a patient setting, focused on the development of a general model, and second, with an intra-patient setting, by providing a specialized model.

The following sections are organized as follows. Section 6.2.1 describes the data generation and its characteristics. Section 6.2.2 presents the different metrics and losses for evaluation. Then, Section 6.2.3, defines and describes important concepts and composition of the proposed bronchoscopic system and its elements, altogether with a description of the temporal learning architectures and proposed population modeling. Next, Section 6.3 presents the experiments, their setup, and the main results obtained for the system while exploring their significance. Finally, Section 6.4 concludes this chapter with the main key points and important takeaways.

6.2 Methodology

6.2.1 Dataset Generation

Virtual lung models are built from an own database of computed tomography scans [2] [588] using [556] to segment the airways. Virtual airways models are simulated using their own platform developed in C++ and VTK, BronchoX.

From the virtual models, bronchoscope trajectories are simulated from the trachea entrance up until the 4-6 level and cover the upper-right, lower-right,

---

[2] CVC CPAP Study Database

upper-left, and lower-left lobes. Trajectories are generated from the central navigation path through the luminal central line traversed using the arch-length parameter. Different increments in this parameter allow the simulation of varying velocities across the path.

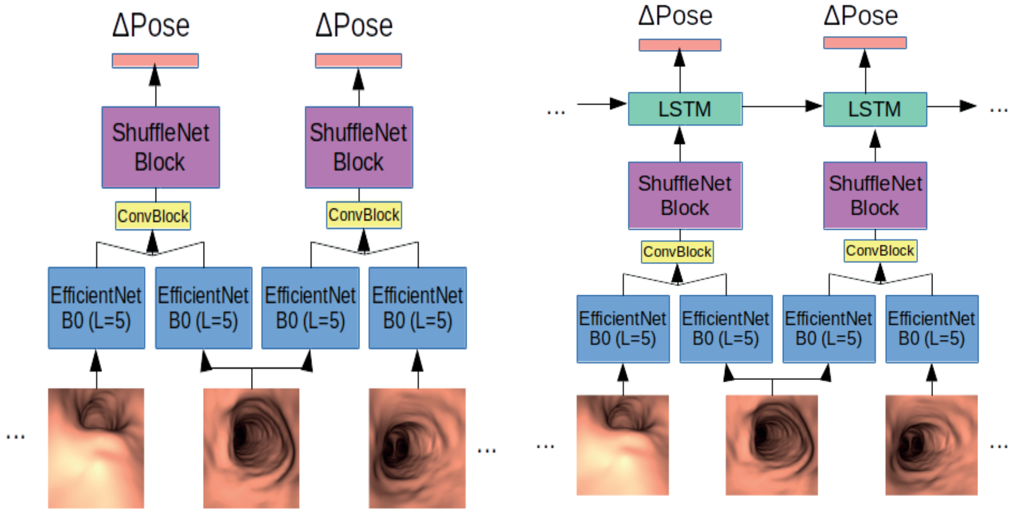
For each central path, different variations, both, in position (between $[-2:1:2]$ voxels in each axis) and camera orientation (in the range $[-45:15:45]$ degrees of rotations around the navigation vector) are generated. The variation in camera position implicitly also modifies the camera point of view, since it is given by its position and a point in the central path at a distance Δd from the current point. The rotation around this navigation vector introduces a variation in the orientation of the image plane. This way, we simulate a full change in the camera central pose.

Finally, paths with neighboring variations are randomly combined along the navigation arc-length parameter in order to simulate realistic trajectories. In total, our dataset has 876 trajectories per patient and lobe, amounting to a total of 842712 frames.

The dataset has as input values the synthetic frames from the camera view during the trajectory and as ground truth source values the associated pose inside the VTK airway model coordinate system. The position is in voxel units, and camera view angles are presented in Euler angles.

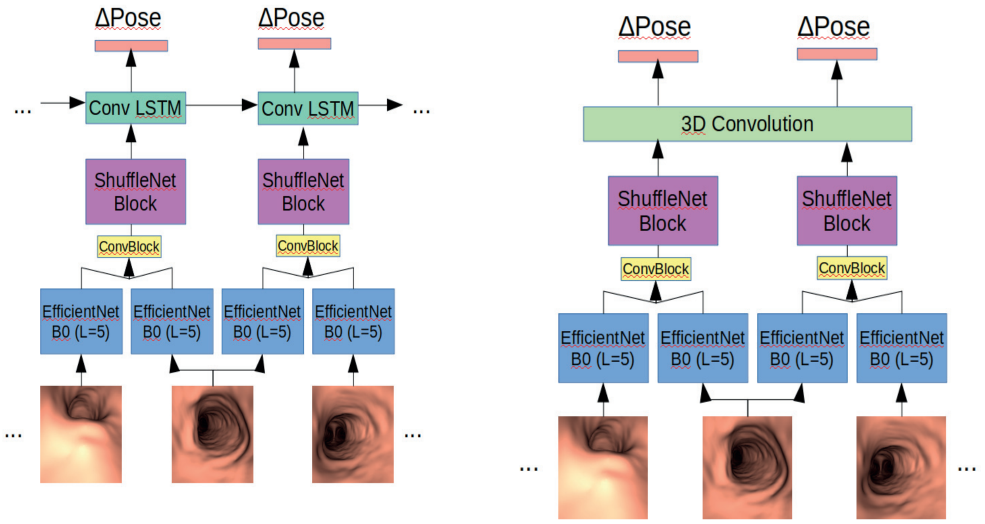
Figure 6.1a shows some dataset examples. Each row has different carinas and each column represents variations in position and orientation from the central navigation. Figure 6.1b shows examples of the lower left and right lobes for two different patients. Dataset will be made publicly available.

Additionally, the data is processed before training to prepare the inputs and ground truths. For every two pairs of images in a sequence, the difference in position and rotation between them is computed. Both components, the difference in position $\Delta p = (\Delta x, \Delta y, \Delta z) \in \mathbb{R}^3$, and the difference in orientation $\Delta o = (\Delta \alpha, \Delta \beta, \Delta \gamma) \in \mathbb{R}^3$, define the difference in pose, $\Delta \mathbf{P} = (\Delta p, \Delta o) \in \mathbb{R}^6$, which constitutes the ground truth of the system. That is, we predict the difference in position and orientation between two images, thus allowing both tracking and calibration. No standardization is applied to the ground truths, while images are standardized through mean subtraction and standard deviation division.



(A) Base network

(B) Base network plus recurrence



(C) Base network plus convolutional recurrence

(D) Base network plus 3d convolutions

FIGURE 6.2: Architectures for bronchoscopy calibration and tracking. (a) is the baseline network without temporal information management, as in [24]. (b, c, d) include different mechanisms to manage temporal information across predictions.

6.2.2 Metrics

In order to train and validate a system for pose estimation, we need metrics for assessing the error of the predicted rotation and position. As those are two separate components, we can have different metrics for each of them.

The most common choice is either the mean squared error (MSE), when the metric is a loss function, or its equivalent, euclidean norm (L2), when it is an evaluation metric. Although they naturally fit the euclidean space of positions, these functions do not necessarily suit the rotation space. An alternative used in the literature is the *direction error* (DE) [589],

$$DE = \cos^{-1}(\mathbf{v}_E \cdot \mathbf{v}_{GT}) \tag{6.1}$$

where $\mathbf{v}_E$ and $\mathbf{v}_{GT}$ are, respectively, the estimated and ground truth direction vectors. Such direction vectors are computed from the rotation matrix, R, and a unitary direction vector, $\mathbf{u}$, as:

$$\mathbf{v}_r = R \cdot \mathbf{u} = \begin{bmatrix} c_\beta c_\gamma & -c_\beta s_\gamma & s_\beta \\ s_\alpha s_\beta c_\gamma + c_\alpha s_\gamma & -s_\alpha s_\beta s_\gamma + c_\alpha c_\gamma & -s_\alpha c_\beta \\ -c_\alpha s_\beta c_\gamma + s_\alpha s_\gamma & c_\alpha s_\beta s_\gamma + s_\alpha s_\gamma & c_\alpha c_\beta \end{bmatrix} \begin{bmatrix} u_x \\ u_y \\ u_z \end{bmatrix}$$

where, c_i and s_i are, respectively, the cosinus and sinus of angle i. A common choice for $\mathbf{u}$ is the look-at vector of the virtual camera, usually given by the x-axis, so that:

$$\mathbf{v}_r = R \cdot \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} = \begin{bmatrix} c_\beta c_\gamma \\ s_\alpha s_\beta c_\gamma + c_\alpha s_\gamma \\ -c_\alpha s_\beta c_\gamma + s_\alpha s_\gamma \end{bmatrix}$$

The main issue with such a function is that the choice of $\mathbf{u}$ affects the perceived rotations. That is, by selecting a specific $\mathbf{u}$, the system is oblivious to the rotations through that direction.

To remedy the such problem, we present an alternative metric, the *cosinus error* (CE):

$$\begin{aligned} CE = \frac{1}{3}((1 - \cos(\Delta\alpha_E - \Delta\alpha_{GT}))+ \\ (1 - \cos(\Delta\beta_E - \Delta\beta_{GT}))+ \\ (1 - \cos(\Delta\gamma_E - \Delta\gamma_{GT}))) \end{aligned}$$

where $(\alpha_E, \beta_E, \gamma_E)$, $(\alpha_{GT}, \beta_{GT}, \gamma_{GT})$ are, respectively the estimated and ground truth Euler angles.

Importantly, all metrics can be used both as loss for training, as well as metrics for evaluation of performance.

6.2.3 Relative Pose Estimation

Given a sequence of L image pairs at time t, $(I_1^t, I_2^t) \in \mathcal{D}_1 \times \mathcal{D}_2$, for $\mathcal{D}_1$, $\mathcal{D}_2$ a source and target domains, a pose estimation system can be formulated as a function, $f_{\mathcal{P}}$, predicting the difference in pose between the image pairs:

$$f_{\mathcal{P}} : (I_1^t, I_2^t)_{t=0}^{t=L} \longrightarrow (\Delta \mathbf{P}^t)_{t=0}^{t=L}, \quad I_i^t \in \mathcal{D}_i, \Delta P^t \in \mathbb{R}^6 \tag{6.2}$$

where each domain $\mathcal{D}_i = \mathbb{R}^{C_i \times H_i \times W_i}$ represents RGB images ($C_i = 3$) of size $H_i \times W_i$, and the difference in pose for each time t is a vector $\Delta P^t = (\Delta p^t, \Delta o^t) = (\Delta x^t, \Delta y^t, \Delta z^t, \Delta \alpha^t, \Delta \beta^t, \Delta \gamma^t)$ representing the difference in (x, y, z) position coordinates and (α, β, γ) Euler angles.

In case I_1^t, I_2^t are consecutive frames of the same path, $f_{\mathcal{P}}$ is modeling a tracker, and the change in a pose through the sequence would be given by accumulating the differences estimated across the video:

$$\mathbf{P}_L = \mathbf{P}^0 + \sum_{t=0}^{L} \Delta \mathbf{P}^t \tag{6.3}$$

where $\mathbf{P}^0$ corresponds to the initial pose vector, and $\mathbf{P}^L$ to the pose vector after the accumulated changes of the L pose differences.

If I_1^t, I_2^t correspond to frames of different paths, $f_{\mathcal{P}}$ would be modeling a pose calibration.

The network loss is given by the addition of the position and rotation metrics:

$$\mathfrak{L} = \mathfrak{L}_{\mathfrak{p}} + \mathfrak{L}_{\mathfrak{o}} \tag{6.4}$$

where p refers to the position and o to orientation. The position loss $\mathfrak{L}_{\mathfrak{p}}$ is given by the MSE error, while for the orientation loss $\mathfrak{L}_{\mathfrak{o}}$ we used the three metrics (MSE, DE, and the proposed CE) described in the previous section.

In any case (tracking or calibration), if the input sequence has more than two image pairs ($L > 0$), there are several ways of processing such temporal information in order to improve the difference in pose estimation. All architectures follow Equation 6.3 and their scheme can be found at Figure 6.2. The different architectures used are a base network working only between two images, and 3 different ways of incorporating temporal information across frames. Next, configuration details for each architecture are presented, altogether with the population modeling approach.

Base Network (Baseline)

The baseline network is a static estimation of pose differences from a single image pair ($L = 0$ in Equation 6.2). Each image is passed through a convolution backbone, specifically an EfficientNet-B0 [183]. Then both feature maps are concatenated and passed through a convolutional block and a ShuffleNet block [17], to obtain a suitable performance/latency trade-off. Finally, the resulting feature map is flattened and passed through a fully connected layer to obtain the different pose predictions between the two images. An illustration of the overall components of the base network can be found in Figure 6.2a.

Recurrence (Baseline + LSTM)

The previous network does not include temporal management. To be able to include such information, the first modification to the base network consists in the addition of a recurrent LSTM [379] (Long Short Term Memory) module after the ShuffleNet block. Such convolutional plus recurrent network type can well exploit temporal information, and thus it has been successfully applied to video tasks, such as object tracking [590], action recognition [591] or video captioning [592].

For every pair of images in the sequence, we obtain a group of feature maps. Each one is flattened and a vector, $\mathbf{v}$, of dimension (L, F), is built, where L is the number of image pairs and F is the size of the flattened feature maps. Such vector, $\mathbf{v}$, is the input for the LSTM block. At each step, l from the sequence of L image pairs, the output vector from the LSTM cell is forwarded to a fully connected layer, which finally delivers the difference pose vector. An illustration of the mentioned module is found in Figure 6.2b.

Convolutional recurrence (Baseline + ConvRNN)

In the previous architecture recurrence required flattening the feature maps so they could be fed to the LSTM. Such flattening destroys visual relations present in the feature maps, thus losing information. To avoid such loss, a possibility is to use a convolutional LSTM [593], where vectorial operations are substituted by convolution ones. In such a way, we are able to maintain the visual structure during recurrence. Flattening, however, is still needed to produce the pose prediction through a fully connected layer after Conv-LSTM. In Figure 6.2c, an illustration of the overall structure is presented.

3D Convolution (Baseline + 3D)

An alternative to recurrence for managing temporal information is 3D convolutions [594, 595]. Once all the feature representations from all image pairs in the sequence are generated, a 4D tensor of size (L, C, H, W) can be built.

Such tensor is fed to a block of two (Conv3D, BatchNorm, ReLU) layers. Working at once with the feature maps coming from all image pairs allows learning relations among them, producing improvements in angle and position estimation. After the 3D convolution block, the result is flattened and fed to a fully connected layer to predict the final difference pose prediction. In Figure 6.2d an illustration of the overall network can be seen.

Approaches for Population

Once the data has been prepared as specified in Section 6.2.1, we define two different approaches to prepare our training and validation scheme. The purpose is to establish the attainable degree of generalization of the models developed in two different industrial settings.

First, is the population or cross-subject setting, in which a patient is selected as validation and the rest of the patients are used for training. This case, in an industrial environment, would correspond to building a general model with different patients and expect enough generalization of the model to apply it directly to an unseen patient.

And second, is the personalized setting, where validation is performed over the same patients as training, but with different sequences. Such a procedure implies that every time we include a patient, models should be retrained. Hence, although the procedure would be more cumbersome, less generalization effort is expected from models.

Pruning

The presented networks, although stemming from a highly efficient network, EfficientNetV1-B0, have a fixed set of computing requirements. To reduce the computational burden and allow for a better adaptation to resource-constrained environments, we apply L_2 structured pruning [167]. Moreover, to avoid the drop in performance, we apply pruning progressively during training: first, we train the network until epoch, e, afterward and until epoch, e', we prune the network the same amount each epoch in an accumulated manner, thus avoiding a sudden change in network training. With such a procedure, we achieve decreasing network requirements while maintaining or even improving network accuracy.

6.3 Experiments and results

In the present section, we present the experimental details and results for the experiments and configurations stated in Section 6.2.3.

Experimental setting . With regards to data and for conducting the following experiments we have selected different groups for each experiment. Moreover, at each experiment, we have selected 15 trajectories per patient and lobe for training, reserving a 3 for validation. Each sequence has been divided into a set of pairs of images. In the case of temporal information experiments, we have selected the same number of sequences, but sequences have been split into chunks of 10 pairs of images.

Networks have been trained using two NVIDIA RTX 2080ti, using Adam optimizer with a learning rate of $1e^{-4}$, and early stopping based on validation loss evolution. Dropout has been added to the ShuffleNet blocks to avoid overfitting. In the case of temporal learning, networks have been trained using truncated backpropagation through time (T-BPTT). A batch size of 2048 has been used in all the experiments, with the exception of the architectures, with a batch size of 512 to offer comparability among trials since the 3D architecture did not fit into memory.

6.3.1 Results

Losses

Table 6.2 shows results (mean ± standard deviation) in the validation set for the different loss combinations evaluated with the different metrics. The best performers are highlighted in boldface. For all evaluation metrics, with the exception of L_2 loss for the L_2 metric, the network trained with the proposed rotation metric CE achieves the best results. Models trained with metrics

	Position Error	**Rotation Error**		
Loss	L_2	L_2	DE	CE
$\mathfrak{L}_{\mathfrak{p}MSE} + \mathfrak{L}_{\mathfrak{o}MSE}$	0.549 ± 0.573	$\mathbf{1.236 \pm 6.762}$	0.724 ± 0.498	0.173 ± 0.194
$\mathfrak{L}_{\mathfrak{p}MSE} + \mathfrak{L}_{\mathfrak{o}DE}$	0.368 ± 0.191	2.441 ± 8.491	0.713 ± 0.533	0.434 ± 0.385
$\mathfrak{L}_{\mathfrak{p}MSE} + \mathfrak{L}_{\mathfrak{o}CE}$	$\mathbf{0.264 \pm 0.169}$	1.554 ± 8.572	$\mathbf{0.542 \pm 0.503}$	$\mathbf{0.116 \pm 0.181}$

TABLE 6.2: Results for the different loss combinations evaluated with the different metrics. Values show mean and standard deviation.

adapted to the rotational domain improve also the results for the position error, not only in average but also in variability. A reduction in the latter indicates a more stable behavior of the predictions and, thus, higher generalization or transfer capability.

Patient Settings

We implement the two patient settings described in Section 6.2.3, outer and intra settings, altogether with the loss configuration, $\mathfrak{L}_{\mathfrak{p}MSE} + \mathfrak{L}_{\mathfrak{o}CE}$. In Figure 6.3, we compare validation performance with regard to the position and angle errors of both options. Clearly, the intra-patient setting obtains notably better results, both for position and angle results, indicating the benefits of retraining the model each time a new patient enters the pool, albeit its associated costs.

Moreover, we also examine the effect of increasing the number of training and validation sequences. As seen in Figure 6.3, increasing data helps reduce the error in the intra-patient setting for both the position and angle terms. However, in the case of the outer-patient setting, it only helps reduce the angle error, while the position error is only reduced slightly. The improvements in the intra-patient setting when increasing data show a positive association with our synthetic dataset since its data production is cheap and unlimited with regard to quantity.

Architectures

Table 6.3 shows results for the validation set and for the different architectures presented in Section 6.2.3. For this case, sequences of 10 pairs of images are used for training, while full sequences are used for validating the methods. We employ $\mathfrak{L}_{\mathfrak{p}MSE} + \mathfrak{L}_{\mathfrak{o}CE}$ as combined loss, as well as the intra patient setting.

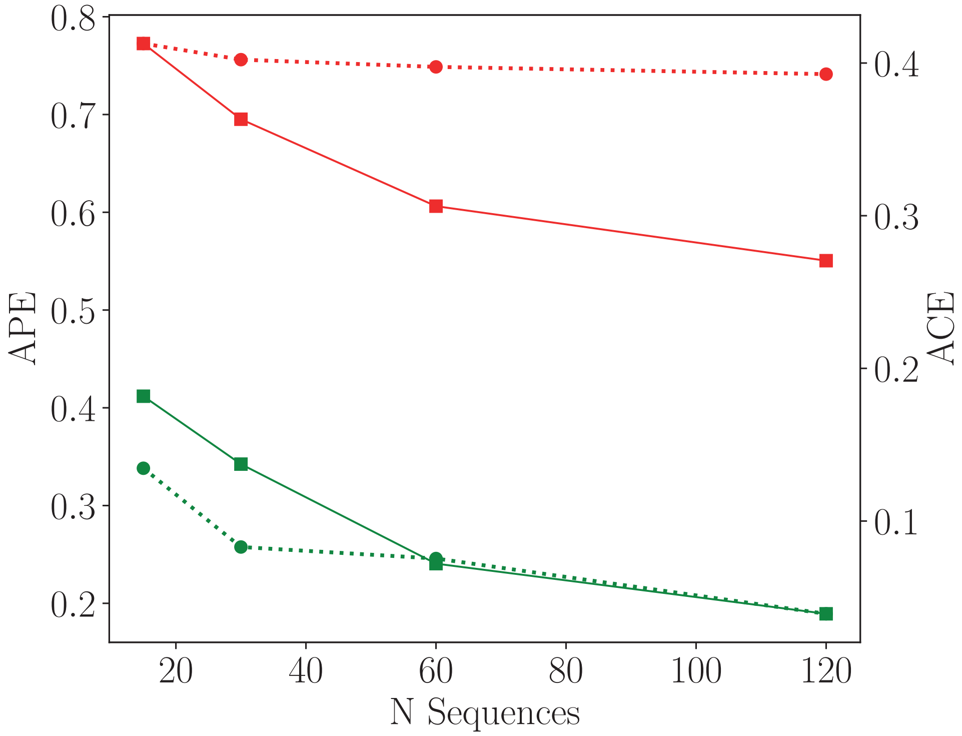


FIGURE 6.3: Effect of data increase in an outer-patient (red) and in an intra-patient setting (green). Average Position Error is represented by a dashed line and Average Cosinus Error is represented by a solid line. The X-axis represents the number of sequences per lobe and patient available.

Config.	L_2 (voxels)	CE
Baseline	0.395 ± 0.268	0.188 ± 0.237
Baseline + LSTM	0.371 ± 0.240	0.213 ± 0.250
Baseline + 3D	0.397 ± 0.252	$\mathbf{0.167} \pm \mathbf{0.221}$
Baseline + ConvRNN	$\mathbf{0.355} \pm \mathbf{0.252}$	0.195 ± 0.242

TABLE 6.3: Ablation results for the different architectures proposed in Section 6.2.3 with regards to performance metrics L_2 and CE.

As observed, recurrence improves position tracking. The results, in both cases, favor the convolutional recurrent solution compared to the plain recurrent, showcasing the benefit of maintaining the 2D visual structure inside the recurrent cell. It is interesting to note the slight worsening of rotation tracking, being worse in the plain recurrent solution.

In the case of the 3D convolution fusion, it can be observed the improvement in rotation tracking, albeit the lack of improvement in position tracking.

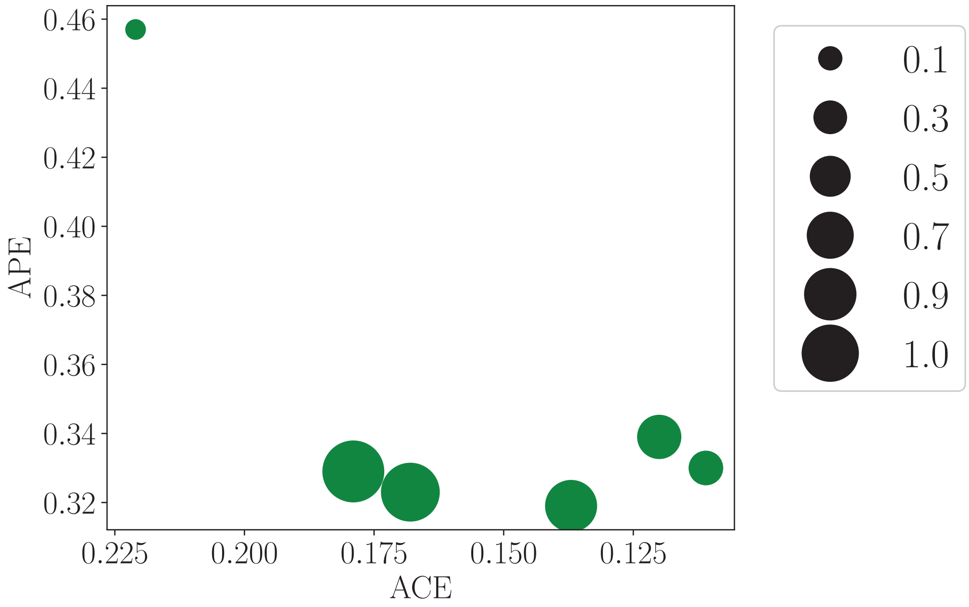


FIGURE 6.4: Results for the pruning experiment. Size represents the relative size of the pruned networks with regard to the original one. Note the improvement in rotation prediction of the networks pruned to a 90%, 70%, 50%, and 30%.

Config.	Param. (M)	GMAC	Latency FP32 (ms)	Latency FP16 (ms)	Latency INT8 (ms)
Baseline	13.967	1.086	157.915	93.9702	70.2556
Baseline + LSTM	14.164	1.087	308.4	97.1851	66.7389
Baseline + 3D	13.983	1.223	163.881	91.6489	62.6699
Baseline + ConvRNN	14.142	1 .089	251.524	95.8629	65.7964

TABLE 6.4: Ablation results for the different architectures proposed in Section 6.2.3. Latency is computed with an NVIDIA Jetson AGX Xavier embedded GPU board, using *trtexec* benchmark, and TensorRT 8.4 as conversion and deployment environment. Sample input has dimensions (1, 10, 3, 256, 256), being (batch size, sequence length, channels, height, width).

With regard to latency, we measure the latency in FP32, FP16, and INT8 for all different architectures. Results are shown in Table 6.4. In FP32, and in both cases, LSTM or ConvRNN. the addition of recurrence increases the number of parameters, the number of operations, and latency, especially in the latter case, probably due to the lack of specific implementations. In the case of 3D, there is only a slight increase in latency for FP32. However, as we move to smaller data types, such as FP16 or INT8 we notice two things, first, the notable decrease in latency for both cases, and second, the reduction of the initial differences of latency, obtaining homogeneous results for all different architectures.

Network	Param. (M)	L_2 (voxel)	CE
OffsetNet [596]	43.6	0.419 ± 0.276	0.197 ± 0.228
BronchoTrack	**9.8**	$\mathbf{0.361 \pm 0.270}$	$\mathbf{0.180 \pm 0.235}$

TABLE 6.5: Comparison with the state-of-the-art method for bronchoscopy tracking and calibration in terms of position error (L_2), angle error, and the number of parameters.

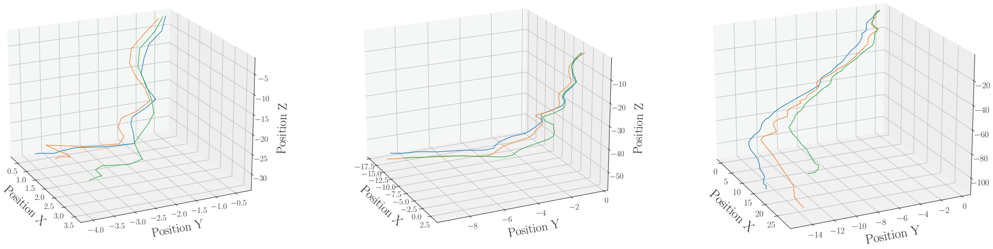


FIGURE 6.5: Position trajectory comparison with the two state-of-the-art methods, OffsetNet (green), our method (orange), and the ground truth (blue).

Network	GPU Board	FP32 Latency (ms)	FP16 Latency (ms)	INT8 Latency (ms)
OffsetNet [596]	GTX 1080 Ti	28.8154	-	23.1551
BronchoTrack (baseline + ConvRNN)	GTX 1080 Ti	40.7378	-	32.8537
OffsetNet [596]	Jetson Xavier	186.705	69.3725	44.1123
BronchoTrack (baseline + ConvRNN)	Jetson Xavier	251.524	95.8629	65.7964

TABLE 6.6: Comparison with the state-of-the-art method for bronchoscopy tracking and calibration in terms of latency in two GPU boards: GTX 1080 Ti and an embedded GPU board, NVIDIA Jetson Xavier. Results for input with dimensions (1, 10, 3, 256, 256), where each dimension corresponds to (batch size, length of sequence, channels, height, width). TensorRT 8.4 is used as a deployment framework and *trtexec* as a benchmark utility. The latency benchmark is performed with unpruned networks since sparsity is only available for Ampere-based architecture boards.

Pruning

In Figure 6.4 results for the pruning scheme defined in Section 6.2.3 are presented. We select as the loss combination, $\mathfrak{L}_{\mathfrak{p}MSE} + \mathfrak{L}_{\mathfrak{o}CE}$, the intra-patient setting, and the convolutional recurrent solution. Taking into account the base network, the latter is pruned up to different degrees, producing a set of pruned networks. Specifically, we prune the base network to a (90%, 70%, 50%, 30%, and 10%) of its original size.

As seen in the figure, pruning up to 70% improves the rotation accuracy while slightly improving the position results. Pruning up to 30% improves further the rotation accuracy but slightly worsens position results. Finally, pruning up to the 10% of its original size critically harms the accuracy of both the rotation and position tracking. All in all, the improvements obtained through pruning show how it can help remove redundancy and adapt the network to the data complexity in case.

Comparison to State-of-the-art

Finally, we compare our model with the current state of the art for bronchoscopy tracking, OffsetNet [581], implementing it as described in the original publication. For such comparison, we build on our best results in previous sections and choose as loss the combination $\mathfrak{L}_{\mathfrak{p}_{MSE}} + \mathfrak{L}_{\mathfrak{o}_{CE}}$, as the architecture of the convolutional recurrent network while pruning it to a 70% of its original size. We call this network BronchoTrack. Both networks are trained and tested in an intra-patient setting, using sequences of 10 steps long for training, and full sequences for testing.

In Table 6.5 we show results for the comparison in the test set. As shown our network is better for both position and orientation tracking. Moreover, it has a notably reduced computational burden in terms of parameters, it weighs more than x4 less. However, with regards to latency results shown in Table 6.6, our network is noticeably slower in FP32, while it has a reduced difference in INT8. We explain these results due to the special operation included in our network (ConvRNN, ShuffleNet blocks), which might not have specialized operations kernels in the deployment framework and for the embedded board, making OffsetNet, which is simpler in terms of operations, much faster.

In Figure 6.5, we further compare both models by showing the accumulated position tracking for example trajectories in the test set. As seen our model is able to closely match the ground truth sequence while OffsetNet deviates more and struggles to follow the ground truth path.

6.4 Conclusions

In the present study, we have presented several contributions to bronchoscopy tracking and calibration. We have built a synthetic dataset to allow for a fair comparison between methods and be able to train data-hungry models. Two different patient settings for training learning methods have been analyzed, concluding with the benefits of an intra-patient setting. We also have experimented with the configuration of a neural network model with regard to the loss function, temporal information management, and resource consumption reduction. Finally, when compared to a state-of-the-art method, better results have been obtained while consuming fewer resources in terms of memory. All in all, there are still important next steps to take to improve current results. Such steps could involve the study of transfer learning to real bronchoscopy videos, for example with the use of generative adversarial networks, or the adaptation to the specific conditions of a medical setting in terms of hardware and resources.

Chapter 7

Conclusions and Future Directions

7.1 Conclusions

Last decade advances in deep learning have supposed a great leap in state-of-the-art results with regard to tasks such as image classification, language translation, and many others. However, with such success, there has been a related increase in model complexity and size, which has incremented the hardware requirements both for training and inference (both generally and initially limited to GPUs). Moreover, the hardware capabilities (OPS performance, memory, throughput, latency, energy) have supposed an initial limitation to deploying applications in resource-constrained platforms and applications, such as mobile or embedded platforms.

There have been many initiatives to reduce training time, and energy costs, and improve data efficiency during the development phase. Equally, there has also been profound research to optimize deep learning models with a focus on inference and deployment: decrease model complexity, size, latency, and memory consumption. In such direction, there are five optimization methods that have stood out: pruning, quantization, neural architecture search, efficient operations, and distillation.

Such techniques can be grouped into two sets: first, those that set off from a network and apply the optimization (pruning, quantization, and distillation), and second, those that create a network (NAS and efficient operations). The first group has benefits limited by the initial network and generally but not always their improvements are traded for a decrease in accuracy. Meanwhile, the second group can offer improvements with the possibility of improving the accuracy by exploring automatically or manually the network configuration space. Thus, as a conclusion, if a great change in latency, or any other performance-related metric, is desired without a limitation in improvements, NAS and efficient operations should be targeted as optimizations.

Nevertheless, the first group of optimizations is usually the fastest to apply since it does not require exploration or manual network building. Moreover, NAS can be extremely expensive in computing time. Hence, in conclusion, it can be stated that quantization, pruning, and distillation are generally cheaper to apply than NAS, in the sense that requires either fewer human or hardware hours to implement and obtain results.

In parallel, in order to enable inference deployment in specialized hardware platforms, new frameworks have appeared. The number of frameworks has grown greatly in the past years targeting many platforms. Examples are CMSIS-NN, uTensor, TF Lite Micro, and others for MCUS, and TF Lite and Core ML for mobile platforms, among many frameworks and platforms. Those frameworks include several features for the deployment of models, but most importantly, the crucial point is if they support the specific model operations and optimizations. This last point is critical since it can affect severely the deployment process. Thus, as conclusion, before training it is of utmost importance to check the full pipeline to deployment, including running inference, ensuring that deployment is feasible.

As stated in the introduction, this thesis has been based on the two previous concepts, optimization methods, and deployment frameworks. Altogether with an industrial research scope. I have worked from researching, improving, and implementing optimization methods, passing through developing deploying tools, to prototyping research projects. Specifically, I have worked in three different industrial and research environments, where I have researched, developed, and applied different optimization methods in order to bring deep learning models to the targeted deployment platforms while using or developing the corresponding deployment frameworks. In all of them, I reached to provide scientific advances with respect to the state of the art.

In the first environment, on MCUs for low-cost automotive applications, I experimented with quantization, NAS, small RNNs, and unsupervised methods. I have shown that tiny small RNNs can be used for gesture recognition on MCUs with simple gestures. Moreover, I have used NAS to find small, fast but well-performing RNNs for gesture recognition by extending and adapting an already existing NAS framework [188] to include RNNs and latency in a multi-objective setting, proving its usefulness. Additionally, I have contributed by open-sourcing an implementation of the arc-kernel [276, 436] necessary for improved multi-objective search with Gaussian processes, and also a converter, quantizer, and automatic deployer for deploying NNs from PyTorch to CMSIS-NN in MCUs [219].

In the second environment, ITM on mobile platforms, I have experimented with quantization and efficient operations. I have used a mixed quantization scheme to, altogether with efficient operations (IRLB) and attention mechanisms, accelerate a network and reduce its computational requirements, so as to deploy it to a mobile platform. Moreover, all optimizations applied enabled the network to run in most of the hardware platforms (CPU and GPU) of the device, proving their versatility and portability.

In the final environment, bronchoscopy navigation on embedded boards, I experimented with pruning and efficient operations. I have used efficient operations (ShuffleNet blocks) and an efficient network (EfficientNet-B0) to produce a network that has a reduced model size and performs better than state-of-the-art models. Moreover, pruning has helped us to reduce the model size further.

All in all, the present thesis, has shown how model optimization methods help reduce computing requirements of NNs during inference, improving their deployment in resource-constrained hardware platforms. These optimizations can serve to improve latency, model size, or memory consumption among many performance metrics. However, each of them has different characteristics, effects, and resource consumption, making them suitable for different applications. Moreover, we have also shown the importance of deployment frameworks, where the support for specific operations and optimization methods is of utmost relevance.

7.2 Future directions

There are still many research directions to improve inference for NNs and ease their employment in resource-constrained platforms.

First of all, we have only tackled a subset of models, mostly CNNs and RNNs. However, newer models, such as transformers, are even heavier, and there is still much room to elucidate how optimizations can help to bring them to resource-constrained devices. There is already active research on the topic [597, 598, 599].

In the same sense, we have explored three platforms: MCUs, mobile, and embedded GPU boards. However, there are other platforms that are experimenting with growth, albeit more limited, on the capability to deploy NN models, such as FPGAs or ASICs. Research trying to ease deployment in FPGAs, for example, could be useful due to their higher energy efficiency characteristics. However, the adaptation of both NN operations and especially, optimization methods, is not a straightforward task and is a topic under active current development [600, 601].

The integration of deployment frameworks and optimizations is a work in progress by companies and researchers. Examples are the integration of sparse kernels in TensorRT or in XNNPACK. Currently, design has to specifically take into account every step from the development of the network to the platform and deployment framework to succeed in deployment. Advances in integrating model operations, optimizations, and deployment frameworks will relax the constraints and help spread AI to currently inaccessible applications and situations.

Finally, an also important research focus is the explainability of the improvements provided by methods such as NAS, pruning, or efficient operations, which would result in better comprehension of NNs and their improvement. In parallel, the combination of all of them is another topic that could be further explored.

Chapter 8

Bibliography

[1] J. Borrego-Carazo, D. Castells-Rufas, E. Biempica, and J. Carrabina, "Resource-Constrained Machine Learning for ADAS: A Systematic Review," *IEEE Access*, vol. 8, pp. 40573–40598, 2020.

[2] D. Castells-Rufas, J. Borrego-Carazo, J. Carrabina, J. Naqui, and E. Biempica, "Continuous touch gesture recognition based on RNNs for capacitive proximity sensors," *Personal and Ubiquitous Computing*, Nov. 2020.

[3] J. Borrego-Carazo, D. Castells-Rufas, J. Carrabina, and E. Biempica, "Capacitive-sensing module with dynamic gesture recognition for automotive applications," in *2020 23rd International Symposium on Design and Diagnostics of Electronic Circuits & Systems (DDECS)*, pp. 1–6, IEEE, 2020.

[4] J. Borrego-Carazo, D. Castells-Rufas, J. Carrabina, and E. Biempica, "Extending SpArSe: Automatic Gesture Recognition Architectures for Embedded Devices," in *2020 19th IEEE International Conference on Machine Learning and Applications (ICMLA)*, pp. 7–12, IEEE, 2020.

[5] J. Borrego-Carazo, D. Castells-Rufas, E. Biempica, and J. Carrabina, "Unsupervised Embedded Gesture Recognition based on multi-objective NAS and capacitive sensing," vol. 249, pp. 9–16, IFSA Publishing, Nov. 2020.

[6] J. Borrego-Carazo, M. Ozay, F. Laboyrie, and P. Wisbey, "A Mixed Quantization Network for Computationally Efficient Mobile Inverse Tone Mapping," British Machine Vision Association, Nov. 2021.

[7] R. Schwartz, J. Dodge, N. A. Smith, and O. Etzioni, "Green AI," *Communications of the ACM*, vol. 63, pp. 54–63, Nov. 2020.

[8] S. Bianco, R. Cadene, L. Celona, and P. Napoletano, "Benchmark analysis of representative deep neural network architectures," *IEEE Access*, vol. 6, pp. 64270–64277, 2018.

[9] R. Snieder and K. Larner, *The art of being a scientist: A guide for graduate students and their mentors*. Cambridge University Press, 2009.

[10] R. Krishnamoorthi, "Quantizing deep convolutional networks for efficient inference: A whitepaper," *arXiv:1806.08342 [cs, stat]*, June 2018. arXiv: 1806.08342.

[11] W. Wen, C. Wu, Y. Wang, Y. Chen, and H. Li, "Learning structured sparsity in deep neural networks," *Advances in neural information processing systems*, vol. 29, pp. 2074–2082, 2016.

[12] T. Liang, J. Glossner, L. Wang, S. Shi, and X. Zhang, "Pruning and quantization for deep neural network acceleration: A survey," *Neurocomputing*, vol. 461, pp. 370–403, 2021.

[13] H. Wang, X. Hu, Q. Zhang, Y. Wang, L. Yu, and H. Hu, "Structured pruning for efficient convolutional neural networks via incremental regularization," *IEEE Journal of Selected Topics in Signal Processing*, vol. 14, no. 4, pp. 775–788, 2019.

[14] K. He, X. Zhang, S. Ren, and J. Sun, "Deep Residual Learning for Image Recognition," *arXiv:1512.03385 [cs]*, Dec. 2015. arXiv: 1512.03385 version: 1.

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

[16] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L.-C. Chen, "Mobilenetv2: Inverted residuals and linear bottlenecks," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 4510–4520, 2018.

[17] N. Ma, X. Zhang, H.-T. Zheng, and J. Sun, "Shufflenet v2: Practical guidelines for efficient cnn architecture design," in *Proceedings of the European conference on computer vision (ECCV)*, pp. 116–131, July 2018.

[18] A. Reuther, P. Michaleas, M. Jones, V. Gadepally, S. Samsi, and J. Kepner, "Survey and benchmarking of machine learning accelerators," in

2019 IEEE high performance extreme computing conference (HPEC), pp. 1–9, IEEE, 2019.

[19] Y. Zhang, K. Li, K. Li, L. Wang, B. Zhong, and Y. Fu, "Image Super-Resolution Using Very Deep Residual Channel Attention Networks," in *Computer Vision – ECCV 2018* (V. Ferrari, M. Hebert, C. Sminchisescu, and Y. Weiss, eds.), Lecture Notes in Computer Science, (Cham), pp. 294–310, Springer International Publishing, 2018.

[20] G. Eilertsen, J. Kronander, G. Denes, R. K. Mantiuk, and J. Unger, "HDR image reconstruction from a single exposure using deep CNNs," *ACM transactions on graphics (TOG)*, vol. 36, no. 6, pp. 1–15, 2017.

[21] D. Marnerides, T. Bashford-Rogers, J. Hatchett, and K. Debattista, "Expandnet: A deep convolutional neural network for high dynamic range expansion from low dynamic range content," in *Computer Graphics Forum*, vol. 37, pp. 37–49, Wiley Online Library, 2018.

[22] Y.-L. Liu, W.-S. Lai, Y.-S. Chen, Y.-L. Kao, M.-H. Yang, Y.-Y. Chuang, and J.-B. Huang, "Single-image HDR reconstruction by learning to reverse the camera pipeline," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 1651–1660, 2020.

[23] H. L. , "Photo Editing Software for HDR & Real Estate Photography \textbar Photomatix," 2017.

[24] J. Sganga, D. Eng, C. Graetzel, and D. Camarillo, "Offsetnet: Deep learning for localization in the lung using rendered images," in *2019 International Conference on Robotics and Automation (ICRA)*, pp. 5046–5052, IEEE, 2019.

[25] Y. Wang, J. Wang, W. Zhang, Y. Zhan, S. Guo, Q. Zheng, and X. Wang, "A survey on deploying mobile deep learning applications: A systemic and technical perspective," *Digital Communications and Networks*, vol. 8, no. 1, pp. 1–17, 2022. Publisher: Elsevier.

[26] M. Xu, J. Liu, Y. Liu, F. X. Lin, Y. Liu, and X. Liu, "A First Look at Deep Learning Apps on Smartphones," Tech. Rep. arXiv:1812.05448, arXiv, Jan. 2021. arXiv:1812.05448 [cs] type: article.

[27] S. Albawi, O. Bayat, S. Al-Azawi, and O. N. Ucan, "Social touch gesture recognition using convolutional neural network," *Computational Intelligence and Neuroscience*, vol. 2018, 2018. Publisher: Hindawi.

[28] M. M. Jung, X. L. Cang, M. Poel, and K. E. MacLean, “Touch challenge’15: Recognizing social touch gestures,” in *Proceedings of the 2015 ACM on International Conference on Multimodal Interaction*, pp. 387–390, 2015.

[29] Z. Khan, M. Khanna, and S. Raman, “Fhdr: Hdr image reconstruction from a single ldr image using feedback network,” in *2019 IEEE Global Conference on Signal and Information Processing (GlobalSIP)*, pp. 1–5, IEEE, 2019.

[30] R. Khare and W. E. Higgins, “Toward image-based global registration for bronchoscopy guidance,” in *Medical Imaging 2010: Visualization, Image-Guided Procedures, and Modeling*, vol. 7625, p. 762510, International Society for Optics and Photonics, 2010.

[31] S. Russell and P. Norvig, “Artificial Intelligence: A Modern Approach, Global Edition 4th,” *Foundations*, vol. 19, p. 23, 2021.

[32] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, Nov. 2016. Google-Books-ID: omivDQAAQBAJ.

[33] D. Amodei and D. Hernandez, “AI and Compute,” May 2018.

[34] E. Strubell, A. Ganesh, and A. McCallum, “Energy and policy considerations for deep learning in NLP,” *arXiv preprint arXiv:1906.02243*, 2019.

[35] H. Cai, C. Gan, T. Wang, Z. Zhang, and S. Han, “Once-for-all: Train one network and specialize it for efficient deployment,” *arXiv preprint arXiv:1908.09791*, 2019.

[36] T. Hobbes and M. Missner, *Thomas Hobbes: Leviathan (Longman Library of Primary Sources in Philosophy)*. Routledge, 2016.

[37] M. Shelley, *Frankenstein 1818*. Neri Pozza Editore, 2018.

[38] J. Fuegi and J. Francis, “Lovelace & Babbage and the creation of the 1843’notes’,” *IEEE Annals of the History of Computing*, vol. 25, no. 4, pp. 16–26, 2003.

[39] C. Babbage, “On the economy of machinery and manufactures,” 1832.

[40] A. M. Turing, “Computing machinery and intelligence,” in *Parsing the turing test*, pp. 23–65, Springer, 2009.

[41] P. Broca, "Remarks on the seat of the faculty of articulated language, following an observation of aphemia (loss of speech)," *Bulletin de la Société Anatomique*, vol. 6, pp. 330–57, 1861.

[42] C. Golgi, "The neuron doctrine: theory and facts," *Nobel lecture*, vol. 1921, pp. 190–217, 1906.

[43] S. R. y Cajal, *Die Retina der Wirbelthiere*. Bergmann, 1894.

[44] F. Crick, "The impact of molecular biology on neuroscience," *Philosophical Transactions of the Royal Society of London. Series B: Biological Sciences*, vol. 354, no. 1392, pp. 2021–2025, 1999.

[45] B. V. Zemelman, G. A. Lee, M. Ng, and G. Miesenböck, "Selective photostimulation of genetically chARGed neurons," *Neuron*, vol. 33, no. 1, pp. 15–22, 2002.

[46] W. S. McCulloch and W. Pitts, "A logical calculus of the ideas immanent in nervous activity," *The bulletin of mathematical biophysics*, vol. 5, pp. 115–133, Dec. 1943.

[47] D. Hebb, "The Organization of Behavior: A Neuropsychological Theory," 1949.

[48] A. Newell and H. Simon, "The logic theory machine–A complex information processing system," *IRE Transactions on information theory*, vol. 2, no. 3, pp. 61–79, 1956.

[49] H. L. Gelernter, "Realization of a geometry theorem proving machine.," in *IFIP congress*, pp. 273–281, 1959.

[50] J. McCarthy, *Programs with common sense*. RLE and MIT computation center, 1960.

[51] B. Widrow and M. E. Hoff, "Adaptive switching circuits," tech. rep., Stanford Univ Ca Stanford Electronics Labs, 1960.

[52] B. Widrow, "Generalization and information storage in network of adaline 'neurons'," *undefined*, 1962.

[53] F. Rosenbaltt, "The perceptron–a perciving and recognizing automation," *Cornell Aeronautical Laboratory*, 1957.

[54] F. Rosenblatt, "The perceptron: A probabilistic model for information storage and organization in the brain," *Psychological Review*, vol. 65, no. 6, pp. 386–408, 1958.

[55] F. Rosenblatt, “On the convergence of reinforcement procedures in simple perceptrons,” *Cornell Aeronautical Laboratory Report VG-1196-G-4, Buffalo, NY*, p. 72, 1960.

[56] F. Rosenblatt, “PRINCIPLES OF NEURODYNAMICS. PERCEPTRONS AND THE THEORY OF BRAIN MECHANISMS,” tech. rep., CORNELL AERONAUTICAL LAB INC BUFFALO NY, Mar. 1961.

[57] H. D. Block, B. W. Knight, and F. Rosenblatt, “Analysis of a Four-Layer Series-Coupled Perceptron. II,” *Reviews of Modern Physics*, vol. 34, pp. 135–142, Jan. 1962.

[58] S. Winograd and J. Cowan, *Reliable Computation in the Presence of Noise*. Cambridge, MA, USA: MIT Press, Dec. 1963.

[59] J. Lighthill, “Artificial intelligence: a general survey. Science Research Council,” 1973.

[60] M. Marvin and A. P. Seymour, “Perceptrons,” 1969.

[61] J. L. McClelland, D. E. Rumelhart, and P. R. Group, *Parallel distributed processing*, vol. 2. MIT press Cambridge, MA, 1986.

[62] J. L. McClelland, “Parallel distributed processing: Implications for cognition and development,” tech. rep., CARNEGIE-MELLON UNIV PITTSBURGH PA ARTIFICIAL INTELLIGENCE AND PSYCHOLOGY . . . , 1988.

[63] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, “Learning representations by back-propagating errors,” *nature*, vol. 323, no. 6088, pp. 533–536, 1986.

[64] S. E. Dreyfus, “Artificial neural networks, back propagation, and the Kelley-Bryson gradient procedure,” *Journal of guidance, control, and dynamics*, vol. 13, no. 5, pp. 926–928, 1990.

[65] P. Werbos, “Beyond regression:" new tools for prediction and analysis in the behavioral sciences,” *Ph. D. dissertation, Harvard University*, 1974.

[66] D. B. Parker, “Learnins logic.,” *Technical Report*, 1985.

[67] H. J. Kelley, “Gradient theory of optimal flight paths,” *Ars Journal*, vol. 30, no. 10, pp. 947–954, 1960.

[68] A. E. Bryson, “A gradient method for optimizing multi-stage allocation processes,” in *Proc. Harvard Univ. Symposium on digital computers and their applications*, vol. 72, p. 22, 1961.

[69] G. Cybenko, “Approximation by superpositions of a sigmoidal function,” *Mathematics of control, signals and systems*, vol. 2, no. 4, pp. 303–314, 1989.

[70] B. Boser, E. Sackinger, J. Bromley, Y. LeCun, R. Howard, and L. Jackel, “An analog neural network processor and its application to high-speed character recognition,” in *IJCNN-91-Seattle International Joint Conference on Neural Networks*, vol. i, pp. 415–420 vol.1, July 1991.

[71] H. Graf and D. Henderson, “A reconfigurable CMOS neural network,” in *1990 37th IEEE International Conference on Solid-State Circuits*, pp. 144–145, Feb. 1990.

[72] J. Carrabina, F. Lisa, V. Gaitan, L. Garrido, and E. Valderrama, “Hardware implementation of a neural network for high energy physics application,” in *International Workshop on Artificial Neural Networks*, pp. 426–431, Springer, 1993.

[73] “ Capítulo XI: Chips Neuronales .”

[74] F. Lisa, J. Carrabina, C. Perez-Vicente, N. Avellana, and E. Valderrama, “Two-bit weights are enough to solve vehicle license number recognition problem,” in *IEEE international conference on neural networks*, pp. 1242–1246, IEEE, 1993.

[75] Y. Bengio, P. Simard, and P. Frasconi, “Learning long-term dependencies with gradient descent is difficult,” *IEEE transactions on neural networks*, vol. 5, no. 2, pp. 157–166, 1994.

[76] S. Hochreiter, “Untersuchungen zu dynamischen neuronalen Netzen,” *Diploma, Technische Universität München*, vol. 91, no. 1, 1991.

[77] S. Hochreiter and J. Schmidhuber, “Long Short-Term Memory,” *Neural Computation*, vol. 9, pp. 1735–1780, Nov. 1997.

[78] K. Fukushima and S. Miyake, “Neocognitron: A self-organizing neural network model for a mechanism of visual pattern recognition,” in *Competition and cooperation in neural nets*, pp. 267–285, Springer, 1982.

[79] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, “Gradient-based learning applied to document recognition,” *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998. Publisher: Ieee.

[80] Y. Bengio, R. Ducharme, and P. Vincent, “A neural probabilistic language model,” in *Advances in Neural Information Processing Systems*, pp. 932–938, 2001.

[81] G. E. Hinton, S. Osindero, and Y.-W. Teh, “A Fast Learning Algorithm for Deep Belief Nets,” *Neural Computation*, vol. 18, pp. 1527–1554, July 2006.

[82] Y. Bengio, P. Lamblin, D. Popovici, and H. Larochelle, “Greedy layer-wise training of deep networks,” in *Advances in neural information processing systems*, pp. 153–160, 2007.

[83] Y. Bengio and Y. LeCun, “Scaling learning algorithms towards AI,” *Large-scale kernel machines*, vol. 34, no. 5, pp. 1–41, 2007.

[84] O. Delalleau and Y. Bengio, “Shallow vs. deep sum-product networks,” *Advances in neural information processing systems*, vol. 24, pp. 666–674, 2011.

[85] A. Krizhevsky, I. Sutskever, and G. E. Hinton, “Imagenet classification with deep convolutional neural networks,” *Advances in neural information processing systems*, vol. 25, pp. 1097–1105, 2012.

[86] A. Hannun, C. Case, J. Casper, B. Catanzaro, G. Diamos, E. Elsen, R. Prenger, S. Satheesh, S. Sengupta, and A. Coates, “Deep speech: Scaling up end-to-end speech recognition,” *arXiv preprint arXiv:1412.5567*, 2014.

[87] O. Abdel-Hamid, A.-r. Mohamed, H. Jiang, L. Deng, G. Penn, and D. Yu, “Convolutional Neural Networks for Speech Recognition,” *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, vol. 22, pp. 1533–1545, Oct. 2014.

[88] A. Graves, A.-r. Mohamed, and G. Hinton, “Speech recognition with deep recurrent neural networks,” in *2013 IEEE International Conference on Acoustics, Speech and Signal Processing*, pp. 6645–6649, May 2013. ISSN: 2379-190X.

[89] B. Hu, Z. Lu, H. Li, and Q. Chen, "Convolutional neural network architectures for matching natural language sentences," in *Advances in neural information processing systems*, pp. 2042–2050, 2014.

[90] G. Mesnil, X. He, L. Deng, and Y. Bengio, "Investigation of recurrent-neural-network architectures and learning methods for spoken language understanding.," in *Interspeech*, pp. 3771–3775, 2013.

[91] T. Mikolov and G. Zweig, "Context dependent recurrent neural network language model," in *2012 IEEE Spoken Language Technology Workshop (SLT)*, pp. 234–239, Dec. 2012.

[92] J. Bobadilla, F. Ortega, A. Hernando, and A. Gutiérrez, "Recommender systems survey," *Knowledge-Based Systems*, vol. 46, pp. 109–132, July 2013.

[93] H. Wang, N. Wang, and D.-Y. Yeung, "Collaborative Deep Learning for Recommender Systems," in *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pp. 1235–1244, New York, NY, USA: Association for Computing Machinery, Aug. 2015.

[94] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller, "Playing atari with deep reinforcement learning," *arXiv preprint arXiv:1312.5602*, 2013.

[95] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra, "Continuous control with deep reinforcement learning," *arXiv preprint arXiv:1509.02971*, 2015.

[96] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: a simple way to prevent neural networks from overfitting," *The journal of machine learning research*, vol. 15, no. 1, pp. 1929–1958, 2014. Publisher: JMLR. org.

[97] G. E. Dahl, T. N. Sainath, and G. E. Hinton, "Improving deep neural networks for LVCSR using rectified linear units and dropout," in *2013 IEEE international conference on acoustics, speech and signal processing*, pp. 8609–8613, IEEE, 2013.

[98] D. Molchanov, A. Ashukha, and D. Vetrov, "Variational dropout sparsifies deep neural networks," in *International Conference on Machine Learning*, pp. 2498–2507, PMLR, 2017.

[99] V. Nair and G. E. Hinton, "Rectified linear units improve restricted boltzmann machines," in *Icml*, 2010.

[100] B. Xu, N. Wang, T. Chen, and M. Li, "Empirical evaluation of rectified activations in convolutional network," *arXiv preprint arXiv:1505.00853*, 2015.

[101] D.-A. Clevert, T. Unterthiner, and S. Hochreiter, "Fast and accurate deep network learning by exponential linear units (elus)," *arXiv preprint arXiv:1511.07289*, 2015.

[102] G. Klambauer, T. Unterthiner, A. Mayr, and S. Hochreiter, "Self-normalizing neural networks," *Advances in neural information processing systems*, vol. 30, 2017.

[103] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," in *International conference on machine learning*, pp. 448–456, PMLR, 2015.

[104] S. Santurkar, D. Tsipras, A. Ilyas, and A. Madry, "How does batch normalization help optimization?," *Advances in neural information processing systems*, vol. 31, 2018.

[105] A. Krogh and J. Hertz, "A simple weight decay can improve generalization," *Advances in neural information processing systems*, vol. 4, 1991.

[106] I. Loshchilov and F. Hutter, "Decoupled weight decay regularization," *arXiv preprint arXiv:1711.05101*, 2017.

[107] L. N. Smith, "A disciplined approach to neural network hyper-parameters: Part 1–learning rate, batch size, momentum, and weight decay," *arXiv preprint arXiv:1803.09820*, 2018.

[108] I. Goodfellow, "Nips 2016 tutorial: Generative adversarial networks," *arXiv preprint arXiv:1701.00160*, 2016.

[109] A. Radford, L. Metz, and S. Chintala, "Unsupervised representation learning with deep convolutional generative adversarial networks," *arXiv preprint arXiv:1511.06434*, 2015.

[110] M. Arjovsky, S. Chintala, and L. Bottou, "Wasserstein generative adversarial networks," in *International conference on machine learning*, pp. 214–223, PMLR, 2017.

[111] M. Jaderberg, K. Simonyan, and A. Zisserman, "Spatial transformer networks," *Advances in neural information processing systems*, vol. 28, 2015.

[112] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, \. Kaiser, and I. Polosukhin, "Attention is all you need," in *Advances in neural information processing systems*, pp. 5998–6008, 2017.

[113] T. Wolf, L. Debut, V. Sanh, J. Chaumond, C. Delangue, A. Moi, P. Cistac, T. Rault, R. Louf, and M. Funtowicz, "Huggingface's transformers: State-of-the-art natural language processing," *arXiv preprint arXiv:1910.03771*, 2019.

[114] J. C.-W. Lin, Y. Shao, Y. Djenouri, and U. Yun, "ASRNN: a recurrent neural network with an attention model for sequence labeling," *Knowledge-Based Systems*, vol. 212, p. 106548, 2021. Publisher: Elsevier.

[115] N. Kalchbrenner, E. Grefenstette, and P. Blunsom, "A convolutional neural network for modelling sentences," *arXiv preprint arXiv:1404.2188*, 2014.

[116] B. Graham, "Sparse 3D convolutional neural networks," *arXiv preprint arXiv:1505.02890*, 2015.

[117] J. Gu, Z. Wang, J. Kuen, L. Ma, A. Shahroudy, B. Shuai, T. Liu, X. Wang, G. Wang, and J. Cai, "Recent advances in convolutional neural networks," *Pattern Recognition*, vol. 77, pp. 354–377, 2018. Publisher: Elsevier.

[118] F. Scarselli, M. Gori, A. C. Tsoi, M. Hagenbuchner, and G. Monfardini, "The graph neural network model," *IEEE transactions on neural networks*, vol. 20, no. 1, pp. 61–80, 2008. Publisher: IEEE.

[119] T. N. Kipf and M. Welling, "Semi-supervised classification with graph convolutional networks," *arXiv preprint arXiv:1609.02907*, 2016.

[120] H. W. Lin, M. Tegmark, and D. Rolnick, "Why does deep and cheap learning work so well?," *Journal of Statistical Physics*, vol. 168, no. 6, pp. 1223–1247, 2017. Publisher: Springer.

[121] T. Poggio, F. Anselmi, and L. Rosasco, "I-theory on depth vs width: hierarchical function composition," tech. rep., Center for Brains, Minds and Machines (CBMM), 2015.

[122] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow,

A. Harp, G. Irving, M. Isard, R. Jozefowicz, Y. Jia, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, M. Schuster, R. Monga, S. Moore, D. Murray, C. Olah, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng, "TensorFlow, Large-scale machine learning on heterogeneous systems," Nov. 2015.

[123] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, and L. Antiga, "Pytorch: An imperative style, high-performance deep learning library," *Advances in neural information processing systems*, vol. 32, pp. 8026–8037, 2019.

[124] T. Chen, M. Li, Y. Li, M. Lin, N. Wang, M. Wang, T. Xiao, B. Xu, C. Zhang, and Z. Zhang, "Mxnet: A flexible and efficient machine learning library for heterogeneous distributed systems," *arXiv preprint arXiv:1512.01274*, Dec. 2015. arXiv: 1512.01274.

[125] A. Krizhevsky and G. Hinton, "Learning multiple layers of features from tiny images," 2009.

[126] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "ImageNet: A large-scale hierarchical image database," in *2009 IEEE Conference on Computer Vision and Pattern Recognition*, pp. 248–255, June 2009. ISSN: 1063-6919.

[127] J. Deng, A. C. Berg, K. Li, and L. Fei-Fei, "What Does Classifying More Than 10,000 Image Categories Tell Us?," in *Computer Vision – ECCV 2010* (K. Daniilidis, P. Maragos, and N. Paragios, eds.), Lecture Notes in Computer Science, (Berlin, Heidelberg), pp. 71–84, Springer, 2010.

[128] S. Abu-El-Haija, N. Kothari, J. Lee, P. Natsev, G. Toderici, B. Varadarajan, and S. Vijayanarasimhan, "YouTube-8M: A Large-Scale Video Classification Benchmark," *arXiv:1609.08675 [cs]*, Sept. 2016. arXiv: 1609.08675.

[129] O. Bojar, C. Buck, C. Federmann, B. Haddow, P. Koehn, J. Leveling, C. Monz, P. Pecina, M. Post, H. Saint-Amand, R. Soricut, L. Specia, and A. Tamchyna, "Findings of the 2014 Workshop on Statistical Machine Translation," in *Proceedings of the Ninth Workshop on Statistical Machine Translation*, (Baltimore, Maryland, USA), pp. 12–58, Association for Computational Linguistics, June 2014.

[130] V. Vanhoucke, A. Senior, and M. Z. Mao, "Improving the speed of neural networks on CPUs," 2011.

[131] D. Steinkraus, I. Buck, and P. Simard, "Using GPUs for machine learning algorithms," in *Eighth International Conference on Document Analysis and Recognition (ICDAR'05)*, pp. 1115–1120 Vol. 2, Aug. 2005. ISSN: 2379-2140.

[132] K. Chellapilla, S. Puri, and P. Simard, "High Performance Convolutional Neural Networks for Document Processing," Suvisoft, Oct. 2006.

[133] R. Raina, A. Madhavan, and A. Y. Ng, "Large-scale deep unsupervised learning using graphics processors," in *Proceedings of the 26th Annual International Conference on Machine Learning*, ICML '09, (New York, NY, USA), pp. 873–880, Association for Computing Machinery, June 2009.

[134] D. C. Cireşan, U. Meier, L. M. Gambardella, and J. Schmidhuber, "Deep, Big, Simple Neural Nets for Handwritten Digit Recognition," *Neural Computation*, vol. 22, pp. 3207–3220, Dec. 2010.

[135] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mane, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viegas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng, "TensorFlow: Large-Scale Machine Learning on Heterogeneous Distributed Systems," *arXiv:1603.04467 [cs]*, Mar. 2016. arXiv: 1603.04467.

[136] M. Shoeybi, M. Patwary, R. Puri, P. LeGresley, J. Casper, and B. Catanzaro, "Megatron-LM: Training Multi-Billion Parameter Language Models Using Model Parallelism," *arXiv:1909.08053 [cs]*, Mar. 2020. arXiv: 1909.08053.

[137] T. B. Brown, B. Mann, N. Ryder, M. Subbiah, J. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, S. Agarwal, A. Herbert-Voss, G. Krueger, T. Henighan, R. Child, A. Ramesh, D. M. Ziegler, J. Wu, C. Winter, C. Hesse, M. Chen, E. Sigler, M. Litwin, S. Gray, B. Chess, J. Clark, C. Berner, S. McCandlish, A. Radford, I. Sutskever, and D. Amodei, "Language Models are Few-Shot Learners," *arXiv:2005.14165 [cs]*, July 2020. arXiv: 2005.14165.

[138] A. Coates, B. Huval, T. Wang, D. Wu, B. Catanzaro, and N. Andrew, "Deep learning with COTS HPC systems," in *International Conference on*

Machine Learning, pp. 1337–1345, PMLR, May 2013.

[139] S. Kumar, "Fundamental Limits to Moore's Law," *arXiv:1511.05956 [cond-mat]*, Nov. 2015. arXiv: 1511.05956.

[140] M. M. Waldrop, "The chips are down for Moore's law," *Nature News*, vol. 530, p. 144, Feb. 2016.

[141] Y. Xian, B. Schiele, and Z. Akata, "Zero-Shot Learning - the Good, the Bad and the Ugly," pp. 4582–4591, 2017.

[142] Y. Deng, "Deep learning on mobile devices: a review," in *Mobile Multimedia/Image Processing, Security, and Applications 2019*, vol. 10993, p. 109930A, International Society for Optics and Photonics, 2019.

[143] J. Chen and X. Ran, "Deep Learning With Edge Computing: A Review.," *Proc. IEEE*, vol. 107, no. 8, pp. 1655–1674, 2019.

[144] V. Sze, Y.-H. Chen, T.-J. Yang, and J. S. Emer, "Efficient Processing of Deep Neural Networks: A Tutorial and Survey," *Proceedings of the IEEE*, vol. 105, pp. 2295–2329, Dec. 2017.

[145] F. A. Berezin, "General concept of quantization," *Communications in Mathematical Physics*, vol. 40, no. 2, pp. 153–174, 1975.

[146] R. M. Gray and D. L. Neuhoff, "Quantization," *IEEE transactions on information theory*, vol. 44, no. 6, pp. 2325–2383, 1998.

[147] G. Dundar and K. Rose, "The effects of quantization on multilayer neural networks," *IEEE Transactions on Neural Networks*, vol. 6, pp. 1446–1451, Nov. 1995.

[148] A. Krishnamurthy, S. Ahalt, D. Melton, and P. Chen, "Neural networks for vector quantization of speech and images," *IEEE Journal on Selected Areas in Communications*, vol. 8, pp. 1449–1457, Oct. 1990.

[149] Y. Xie and M. A. Jabri, "Analysis of the effects of quantization in multilayer neural networks using a statistical model," *IEEE Transactions on Neural Networks*, vol. 3, no. 2, pp. 334–338, 1992.

[150] B. Jacob, S. Kligys, B. Chen, M. Zhu, M. Tang, A. Howard, H. Adam, and D. Kalenichenko, "Quantization and training of neural networks for efficient integer-arithmetic-only inference," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 2704–2713, 2018.

[151] J. L. Holt and T. E. Baker, "Back propagation simulations using limited precision calculations," in *IJCNN-91-Seattle International Joint Conference on Neural Networks*, vol. 2, pp. 121–126, IEEE, 1991.

[152] R. Presley and R. Haggard, "A fixed point implementation of the backpropagation learning algorithm," in *Proceedings of SOUTHEASTCON '94*, pp. 136–138, Apr. 1994.

[153] M. Courbariaux, Y. Bengio, and J.-P. David, "Training deep neural networks with low precision multiplications," *arXiv preprint arXiv:1412.7024*, 2014.

[154] I. Hubara, M. Courbariaux, D. Soudry, R. El-Yaniv, and Y. Bengio, "Quantized neural networks: Training neural networks with low precision weights and activations," *The Journal of Machine Learning Research*, vol. 18, no. 1, pp. 6869–6898, 2017. Publisher: JMLR. org.

[155] Y. Chauvin, "A Back-Propagation Algorithm with Optimal Use of Hidden Units.," in *NIPS*, vol. 1, pp. 519–526, 1988.

[156] A. S. Weigend, D. E. Rumelhart, and B. A. Huberman, "Backpropagation, weight-elimination and time series prediction," in *Connectionist models*, pp. 105–116, Elsevier, 1991.

[157] Y. LeCun, J. S. Denker, and S. A. Solla, "Optimal brain damage," in *Advances in neural information processing systems*, pp. 598–605, 1990.

[158] S. Han, H. Mao, and W. J. Dally, "Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding," *arXiv preprint arXiv:1510.00149*, 2015.

[159] Z. Liu, J. Li, Z. Shen, G. Huang, S. Yan, and C. Zhang, "Learning efficient convolutional networks through network slimming," in *Proceedings of the IEEE international conference on computer vision*, pp. 2736–2744, 2017.

[160] S. Vadera and S. Ameen, "Methods for Pruning Deep Neural Networks," *arXiv preprint arXiv:2011.00241*, 2020.

[161] J. Frankle and M. Carbin, "The Lottery Ticket Hypothesis: Finding Sparse, Trainable Neural Networks," *arXiv:1803.03635 [cs]*, Mar. 2019. arXiv: 1803.03635.

[162] Z. Zhou, W. Zhou, H. Li, and R. Hong, "Online Filter Clustering and Pruning for Efficient Convnets," in *2018 25th IEEE International Conference on Image Processing (ICIP)*, pp. 11–15, Oct. 2018. ISSN: 2381-8549.

[163] H. J. Sussmann, "Uniqueness of the weights for minimal feedforward nets with a given input-output map," *Neural Networks*, vol. 5, pp. 589–593, July 1992.

[164] Z. Zhou, W. Zhou, R. Hong, and H. Li, "Online filter weakening and pruning for efficient convnets," in *2018 IEEE International Conference on Multimedia and Expo (ICME)*, pp. 1–6, IEEE, 2018.

[165] M. C. Mozer and P. Smolensky, "Skeletonization: A technique for trimming the fat from a network via relevance assessment," in *Advances in neural information processing systems*, pp. 107–115, 1989.

[166] S. Lin, R. Ji, Y. Li, Y. Wu, F. Huang, and B. Zhang, "Accelerating Convolutional Networks via Global & Dynamic Filter Pruning.," in *IJCAI*, vol. 2, p. 8, 2018.

[167] S. Anwar, K. Hwang, and W. Sung, "Structured Pruning of Deep Convolutional Neural Networks," *ACM Journal on Emerging Technologies in Computing Systems*, vol. 13, pp. 32:1–32:18, Feb. 2017.

[168] J. Park, S. Li, W. Wen, P. T. P. Tang, H. Li, Y. Chen, and P. Dubey, "Faster CNNs with Direct Sparse Convolutions and Guided Pruning," *arXiv:1608.01409 [cs]*, July 2017. arXiv: 1608.01409.

[169] M. Zhu, T. Zhang, Z. Gu, and Y. Xie, "Sparse Tensor Core: Algorithm and Hardware Co-Design for Vector-wise Sparse Neural Networks on Modern GPUs," in *Proceedings of the 52nd Annual IEEE/ACM International Symposium on Microarchitecture*, MICRO '52, (New York, NY, USA), pp. 359–371, Association for Computing Machinery, Oct. 2019.

[170] L. Lu, J. Xie, R. Huang, J. Zhang, W. Lin, and Y. Liang, "An Efficient Hardware Accelerator for Sparse Convolutional Neural Networks on FPGAs," in *2019 IEEE 27th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*, pp. 17–25, Apr. 2019. ISSN: 2576-2621.

[171] C. Buciluă, R. Caruana, and A. Niculescu-Mizil, "Model compression," in *Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining*, KDD '06, (New York, NY, USA), pp. 535–541, Association for Computing Machinery, Aug. 2006.

[172] G. Hinton, O. Vinyals, and J. Dean, "Distilling the knowledge in a neural network," *arXiv preprint arXiv:1503.02531*, Mar. 2015.

[173] H. Touvron, M. Cord, M. Douze, F. Massa, A. Sablayrolles, and H. Jegou, "Training data-efficient image transformers & distillation through attention," in *International Conference on Machine Learning*, pp. 10347–10357, PMLR, July 2021.

[174] S. Sun, Y. Cheng, Z. Gan, and J. Liu, "Patient Knowledge Distillation for BERT Model Compression," *arXiv:1908.09355 [cs]*, Aug. 2019. arXiv: 1908.09355.

[175] G. Chen, W. Choi, X. Yu, T. Han, and M. Chandraker, "Learning efficient object detection models with knowledge distillation," in *Proceedings of the 31st International Conference on Neural Information Processing Systems*, vol. 30, pp. 742–751, 2017.

[176] Y. Burda, H. Edwards, A. Storkey, and O. Klimov, "Exploration by Random Network Distillation," *arXiv:1810.12894 [cs, stat]*, Oct. 2018. arXiv: 1810.12894.

[177] J. Yim, D. Joo, J. Bae, and J. Kim, "A Gift From Knowledge Distillation: Fast Optimization, Network Minimization and Transfer Learning," pp. 4133–4141, 2017.

[178] W. Park, D. Kim, Y. Lu, and M. Cho, "Relational Knowledge Distillation," pp. 3967–3976, 2019.

[179] R. Müller, S. Kornblith, and G. Hinton, "When Does Label Smoothing Help?," *arXiv:1906.02629 [cs, stat]*, June 2020. arXiv: 1906.02629.

[180] X. Zhang, X. Zhou, M. Lin, and J. Sun, "Shufflenet: An extremely efficient convolutional neural network for mobile devices," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 6848–6856, 2018.

[181] K. Cho, B. Van Merriënboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio, "Learning phrase representations using RNN encoder-decoder for statistical machine translation," *arXiv preprint arXiv:1406.1078*, Sept. 2014.

[182] M. Tan and Q. Le, "Efficientnet: Rethinking model scaling for convolutional neural networks," in *International Conference on Machine Learning*, pp. 6105–6114, PMLR, Sept. 2019.

[183] M. Tan and Q. V. Le, "Efficientnetv2: Smaller models and faster training," *arXiv preprint arXiv:2104.00298*, June 2021.

[184] Y. Cheng, F. X. Yu, R. S. Feris, S. Kumar, A. Choudhary, and S.-F. Chang, “An Exploration of Parameter Redundancy in Deep Networks With Circulant Projections,” pp. 2857–2865, 2015.

[185] J. Dai, H. Qi, Y. Xiong, Y. Li, G. Zhang, H. Hu, and Y. Wei, “Deformable convolutional networks,” in *Proceedings of the IEEE international conference on computer vision*, pp. 764–773, 2017.

[186] X. Zhu, H. Hu, S. Lin, and J. Dai, “Deformable ConvNets V2: More Deformable, Better Results,” pp. 9308–9316, 2019.

[187] T. Elsken, J. H. Metzen, and F. Hutter, “Neural architecture search: A survey,” *The Journal of Machine Learning Research*, vol. 20, no. 1, pp. 1997–2017, 2019.

[188] I. Fedorov, R. P. Adams, M. Mattina, and P. N. Whatmough, “SpArSe: Sparse Architecture Search for CNNs on Resource-Constrained Microcontrollers,” *arXiv:1905.12107 [cs]*, May 2019. arXiv: 1905.12107.

[189] T. Elsken, J. H. Metzen, and F. Hutter, “Efficient multi-objective neural architecture search via lamarckian evolution,” *arXiv preprint arXiv:1804.09081*, 2018.

[190] B. Zoph, V. Vasudevan, J. Shlens, and Q. V. Le, “Learning Transferable Architectures for Scalable Image Recognition,” in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 8697–8710, 2018.

[191] T. Wang, Y. Li, J. Peng, Y. Ma, X. Wang, F. Song, and Y. Yan, “Real-time Image Enhancer via Learnable Spatial-aware 3D Lookup Tables,” *arXiv:2108.08697 [cs, eess]*, Aug. 2021. arXiv: 2108.08697.

[192] H. Zeng, J. Cai, L. Li, Z. Cao, and L. Zhang, “Learning Image-adaptive 3D Lookup Tables for High Performance Photo Enhancement in Real-time,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, pp. 1–1, 2020.

[193] M. Gharbi, J. Chen, J. T. Barron, S. W. Hasinoff, and F. Durand, “Deep bilateral learning for real-time image enhancement,” *ACM Transactions on Graphics*, vol. 36, pp. 118:1–118:12, July 2017.

[194] J. Liang, Y. Xu, Y. Quan, J. Wang, H. Ling, and H. Ji, “Deep Bilateral Retinex for Low-Light Image Enhancement,” *arXiv:2007.02018 [cs, eess]*, July 2020. arXiv: 2007.02018.

[195] W.-S. Lai, J.-B. Huang, N. Ahuja, and M.-H. Yang, "Deep Laplacian Pyramid Networks for Fast and Accurate Super-Resolution," pp. 624–632, 2017.

[196] J. Liang, H. Zeng, and L. Zhang, "High-Resolution Photorealistic Image Translation in Real-Time: A Laplacian Pyramid Translation Network," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 9392–9400, 2021.

[197] Y. Song, H. Qian, and X. Du, "StarEnhancer: Learning Real-Time and Style-Aware Image Enhancement," *arXiv:2107.12898 [cs]*, Aug. 2021. arXiv: 2107.12898.

[198] Y. Jia, E. Shelhamer, J. Donahue, S. Karayev, J. Long, R. Girshick, S. Guadarrama, and T. Darrell, "Caffe: Convolutional architecture for fast feature embedding. arXiv 2014," *arXiv preprint arXiv:1408.5093*, 2019.

[199] S. Tokui, R. Okuta, T. Akiba, Y. Niitani, T. Ogawa, S. Saito, S. Suzuki, K. Uenishi, B. Vogel, and H. Yamazaki Vincent, "Chainer: A deep learning framework for accelerating the research cycle," in *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pp. 2002–2011, 2019.

[200] "Introduction to TensorFlow Mobile | TensorFlow," July 2018.

[201] "TensorFlow Lite | ML for Mobile and Edge Devices."

[202] "Neural Networks API | Android NDK | Android Developers."

[203] "Qualcomm Research brings server-class machine learning to everyday devices—making them smarter [VIDEO]," Oct. 2015.

[204] A. Ignatov, R. Timofte, W. Chou, K. Wang, M. Wu, T. Hartley, and L. Van Gool, "AI Benchmark: Running Deep Neural Networks on Android Smartphones," in *Computer Vision – ECCV 2018 Workshops* (L. Leal-Taixé and S. Roth, eds.), Lecture Notes in Computer Science, (Cham), pp. 288–314, Springer International Publishing, 2019.

[205] "TensorFlow Lite Delegates."

[206] J. Lee, N. Chirkov, E. Ignasheva, Y. Pisarchyk, M. Shieh, F. Riccardi, R. Sarokin, A. Kulik, and M. Grundmann, "On-device neural net inference with mobile gpus," *arXiv preprint arXiv:1907.01989*, July 2019. arXiv:1907.01989 [cs, stat] type: article.

[207] “PyTorch Mobile.”

[208] “eIQ Auto AI enablement.”

[209] “Edge AI | TI.com.”

[210] “CMSIS NN Software Library.”

[211] “uTensor - Test Release,” Apr. 2022. original-date: 2017-09-21T17:04:59Z.

[212] J. Ma, “Neural Network on Microcontroller (NNoM),” Apr. 2022. original-date: 2019-01-21T19:38:30Z.

[213] “Apache TVM.”

[214] “TensorFlow Lite for Microcontrollers.”

[215] “OpenVINO™ Documentation — OpenVINO™ documentation — Version(latest).”

[216] “NVIDIA TensorRT,” Apr. 2016.

[217] “Arc Kernel for Conditional Spaces · Issue #1023 · cornellius-gp/gpytorch.”

[218] BCJuan, “SparseMod,” Nov. 2021. original-date: 2020-04-20T15:14:03Z.

[219] BCJuan, “torch2cmsis,” Mar. 2022. original-date: 2020-09-02T05:57:06Z.

[220] “Add flop counter hook for rnn, gru and lstm by BCJuan · Pull Request #38 · sovrasov/flops-counter.pytorch.”

[221] A. Fan, P. Stock, B. Graham, E. Grave, R. Gribonval, H. Jegou, and A. Joulin, “Training with quantization noise for extreme model compression,” *arXiv preprint arXiv:2004.07320*, 2020.

[222] M. Rastegari, V. Ordonez, J. Redmon, and A. Farhadi, “Xnor-net: Imagenet classification using binary convolutional neural networks,” in *European conference on computer vision*, pp. 525–542, Springer, 2016.

[223] J. Holi and J.-N. Hwang, “Finite precision error analysis of neural network hardware implementations,” *IEEE Transactions on Computers*, vol. 42, pp. 281–290, Mar. 1993.

[224] P. Y. Simard and H. P. Graf, “Backpropagation without multiplication,” *Advances in Neural Information Processing Systems*, pp. 232–232, 1994.

[225] S. Gupta, A. Agrawal, K. Gopalakrishnan, and P. Narayanan, "Deep Learning with Limited Numerical Precision," in *International Conference on Machine Learning*, pp. 1737–1746, PMLR, June 2015.

[226] G. Menghani, "Efficient Deep Learning: A Survey on Making Deep Learning Models Smaller, Faster, and Better," *arXiv:2106.08962 [cs]*, June 2021. arXiv: 2106.08962.

[227] "gemmlowp: a small self-contained low-precision GEMM library," Oct. 2021. original-date: 2015-07-06T21:15:00Z.

[228] M. Courbariaux, Y. Bengio, and J.-P. David, "Binaryconnect: Training deep neural networks with binary weights during propagations," in *Advances in neural information processing systems*, pp. 3123–3131, 2015.

[229] P.-E. Novac, G. B. Hacene, A. Pegatoquet, B. Miramond, and V. Gripon, "Quantization and deployment of deep neural networks on microcontrollers," *Sensors*, vol. 21, no. 9, p. 2984, 2021. Publisher: Multidisciplinary Digital Publishing Institute.

[230] Z. He and D. Fan, "Simultaneously optimizing weight and quantizer of ternary neural network using truncated gaussian approximation," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 11438–11446, 2019.

[231] E. Lee and Y. Hwang, "Layer-Wise Network Compression Using Gaussian Mixture Model," *Electronics*, vol. 10, no. 1, p. 72, 2021. Publisher: MDPI.

[232] A. Polino, R. Pascanu, and D. Alistarh, "Model compression via distillation and quantization," *arXiv preprint arXiv:1802.05668*, 2018.

[233] H. Tann, S. Hashemi, R. I. Bahar, and S. Reda, "Hardware-software codesign of accurate, multiplier-free deep neural networks," in *2017 54th ACM/EDAC/IEEE Design Automation Conference (DAC)*, pp. 1–6, IEEE, 2017.

[234] A. Mishra and D. Marr, "Apprentice: Using knowledge distillation techniques to improve low-precision network accuracy," *arXiv preprint arXiv:1711.05852*, 2017.

[235] E. Park, S. Yoo, and P. Vajda, "Value-aware quantization for training and inference of neural networks," in *Proceedings of the European Conference on Computer Vision (ECCV)*, pp. 580–595, 2018.

[236] J. Yang, X. Shen, J. Xing, X. Tian, H. Li, B. Deng, J. Huang, and X.-s. Hua, "Quantization networks," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 7308–7316, 2019.

[237] A. Gholami, S. Kim, Z. Dong, Z. Yao, M. W. Mahoney, and K. Keutzer, "A survey of quantization methods for efficient neural network inference," *arXiv preprint arXiv:2103.13630*, 2021.

[238] K. Simonyan and A. Zisserman, "Very Deep Convolutional Networks for Large-Scale Image Recognition," *arXiv:1409.1556 [cs]*, Apr. 2015. arXiv: 1409.1556.

[239] E. Real, A. Aggarwal, Y. Huang, and Q. V. Le, "Aging evolution for image classifier architecture search," in *AAAI conference on artificial intelligence*, vol. 3, 2019.

[240] X. He, K. Zhao, and X. Chu, "AutoML: A survey of the state-of-the-art," *Knowledge-Based Systems*, vol. 212, p. 106622, Jan. 2021.

[241] F. Hutter, L. Kotthoff, and J. Vanschoren, *Automated machine learning: methods, systems, challenges*. Springer Nature, 2019.

[242] T. Yu and H. Zhu, "Hyper-parameter optimization: A review of algorithms and applications," *arXiv preprint arXiv:2003.05689*, 2020.

[243] M. Feurer and F. Hutter, "Hyperparameter optimization," in *Automated machine learning*, pp. 3–33, Springer, Cham, 2019.

[244] J. Vanschoren, "Meta-learning: A survey," *arXiv preprint arXiv:1810.03548*, 2018.

[245] H. Cai, L. Zhu, and S. Han, "Proxylessnas: Direct neural architecture search on target task and hardware," *arXiv preprint arXiv:1812.00332*, 2018.

[246] M. Tan, B. Chen, R. Pang, V. Vasudevan, M. Sandler, A. Howard, and Q. V. Le, "MnasNet: Platform-Aware Neural Architecture Search for Mobile," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 2820–2828, 2019.

[247] W. Jiang, X. Zhang, E. H.-M. Sha, L. Yang, Q. Zhuge, Y. Shi, and J. Hu, "Accuracy vs. efficiency: Achieving both through fpga-implementation aware neural architecture search," in *Proceedings of the 56th Annual Design Automation Conference 2019*, pp. 1–6, 2019.

[248] G. F. Miller, P. M. Todd, and S. U. Hegde, "Designing Neural Networks Using Genetic Algorithms.," in *ICGA*, vol. 89, pp. 379–384, 1989.

[249] P. J. Angeline, G. M. Saunders, and J. B. Pollack, "An evolutionary algorithm that constructs recurrent neural networks," *IEEE transactions on Neural Networks*, vol. 5, no. 1, pp. 54–65, 1994. Publisher: IEEE.

[250] K. O. Stanley and R. Miikkulainen, "Evolving neural networks through augmenting topologies," *Evolutionary computation*, vol. 10, no. 2, pp. 99–127, 2002. Publisher: MIT Press.

[251] D. Floreano, P. Dürr, and C. Mattiussi, "Neuroevolution: from architectures to learning," *Evolutionary intelligence*, vol. 1, no. 1, pp. 47–62, 2008. Publisher: Springer.

[252] X. Yao, "Evolving artificial neural networks," *Proceedings of the IEEE*, vol. 87, no. 9, pp. 1423–1447, 1999. Publisher: IEEE.

[253] L. Xie and A. Yuille, "Genetic cnn," in *Proceedings of the IEEE international conference on computer vision*, pp. 1379–1388, 2017.

[254] R. Miikkulainen, J. Liang, E. Meyerson, A. Rawal, D. Fink, O. Francon, B. Raju, H. Shahrzad, A. Navruzyan, and N. Duffy, "Evolving deep neural networks," in *Artificial intelligence in the age of neural networks and brain computing*, pp. 293–312, Elsevier, 2019.

[255] J. Bergstra, D. Yamins, and D. Cox, "Making a science of model search: Hyperparameter optimization in hundreds of dimensions for vision architectures," in *International conference on machine learning*, pp. 115–123, PMLR, 2013.

[256] H. Mendoza, A. Klein, M. Feurer, J. T. Springenberg, and F. Hutter, "Towards automatically-tuned neural networks," in *Workshop on Automatic Machine Learning*, pp. 58–65, PMLR, 2016.

[257] Z. Zhong, J. Yan, W. Wu, J. Shao, and C.-L. Liu, "Practical block-wise neural network architecture generation," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 2423–2432, 2018.

[258] H. Cai, T. Chen, W. Zhang, Y. Yu, and J. Wang, "Efficient architecture search by network transformation," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 32, 2018. Issue: 1.

[259] A. Brock, T. Lim, J. M. Ritchie, and N. Weston, “Smash: one-shot model architecture search through hypernetworks,” *arXiv preprint arXiv:1708.05344*, 2017.

[260] T. Elsken, J.-H. Metzen, and F. Hutter, “Simple and efficient architecture search for convolutional neural networks,” *arXiv preprint arXiv:1711.04528*, 2017.

[261] Z. Zhong, Z. Yang, B. Deng, J. Yan, W. Wu, J. Shao, and C.-L. Liu, “Block-qnn: Efficient block-wise neural network architecture generation,” *IEEE transactions on pattern analysis and machine intelligence*, vol. 43, no. 7, pp. 2314–2328, 2020. Publisher: IEEE.

[262] H. Liu, K. Simonyan, and Y. Yang, “Darts: Differentiable architecture search,” *arXiv preprint arXiv:1806.09055*, 2018.

[263] H. Cai, J. Yang, W. Zhang, S. Han, and Y. Yu, “Path-level network transformation for efficient architecture search,” in *International Conference on Machine Learning*, pp. 678–687, PMLR, 2018.

[264] H. Liu, K. Simonyan, O. Vinyals, C. Fernando, and K. Kavukcuoglu, “Hierarchical representations for efficient architecture search,” *arXiv preprint arXiv:1711.00436*, 2017.

[265] B. Zoph and Q. V. Le, “Neural architecture search with reinforcement learning,” *arXiv preprint arXiv:1611.01578*, 2016.

[266] B. Baker, O. Gupta, N. Naik, and R. Raskar, “Designing neural network architectures using reinforcement learning,” *arXiv preprint arXiv:1611.02167*, 2016.

[267] R. J. Williams, “Simple statistical gradient-following algorithms for connectionist reinforcement learning,” *Machine learning*, vol. 8, no. 3, pp. 229–256, 1992. Publisher: Springer.

[268] T. Chen, I. Goodfellow, and J. Shlens, “Net2Net: Accelerating Learning via Knowledge Transfer,” *arXiv:1511.05641 [cs]*, Apr. 2016. arXiv: 1511.05641.

[269] E. Real, S. Moore, A. Selle, S. Saxena, Y. L. Suematsu, J. Tan, Q. V. Le, and A. Kurakin, “Large-scale evolution of image classifiers,” in *International Conference on Machine Learning*, pp. 2902–2911, PMLR, 2017.

[270] M. Wistuba, A. Rawat, and T. Pedapati, “A survey on neural architecture search,” *arXiv preprint arXiv:1905.01392*, 2019.

[271] J. Bergstra, R. Bardenet, Y. Bengio, and B. Kégl, "Algorithms for hyperparameter optimization," *Advances in neural information processing systems*, vol. 24, 2011.

[272] F. Hutter, H. H. Hoos, and K. Leyton-Brown, "Sequential model-based optimization for general algorithm configuration," in *International conference on learning and intelligent optimization*, pp. 507–523, Springer, 2011.

[273] T. Domhan, J. T. Springenberg, and F. Hutter, "Speeding up automatic hyperparameter optimization of deep neural networks by extrapolation of learning curves," in *Twenty-fourth international joint conference on artificial intelligence*, 2015.

[274] M. Wistuba, "Finding competitive network architectures within a day using uct," *arXiv preprint arXiv:1712.07420*, 2017.

[275] C. K. Williams and C. E. Rasmussen, *Gaussian processes for machine learning*, vol. 2. MIT press Cambridge, MA, 2006.

[276] K. Swersky, D. Duvenaud, J. Snoek, F. Hutter, and M. A. Osborne, "Raiders of the lost architecture: Kernels for Bayesian optimization in conditional parameter spaces," *arXiv preprint arXiv:1409.4011*, 2014.

[277] "gpytorch.kernels — GPyTorch 1.6.0 documentation."

[278] K. Kandasamy, W. Neiswanger, J. Schneider, B. Poczos, and E. P. Xing, "Neural architecture search with bayesian optimisation and optimal transport," *Advances in neural information processing systems*, vol. 31, 2018.

[279] C. Liu, B. Zoph, M. Neumann, J. Shlens, W. Hua, L.-J. Li, L. Fei-Fei, A. Yuille, J. Huang, and K. Murphy, "Progressive neural architecture search," in *Proceedings of the European conference on computer vision (ECCV)*, pp. 19–34, 2018.

[280] R. Luo, F. Tian, T. Qin, E. Chen, and T.-Y. Liu, "Neural architecture optimization," *Advances in neural information processing systems*, vol. 31, 2018.

[281] P. Chrabaszcz, I. Loshchilov, and F. Hutter, "A downsampled variant of imagenet as an alternative to the cifar datasets," *arXiv preprint arXiv:1707.08819*, 2017.

[282] A. Zela, A. Klein, S. Falkner, and F. Hutter, "Towards automated deep learning: Efficient joint neural architecture and hyperparameter search," *arXiv preprint arXiv:1807.06906*, 2018.

[283] L. Li, K. Jamieson, G. DeSalvo, A. Rostamizadeh, and A. Talwalkar, "Hyperband: A novel bandit-based approach to hyperparameter optimization," *The Journal of Machine Learning Research*, vol. 18, no. 1, pp. 6765–6816, 2017. Publisher: JMLR. org.

[284] K. Swersky, J. Snoek, and R. P. Adams, "Freeze-thaw Bayesian optimization," *arXiv preprint arXiv:1406.3896*, 2014.

[285] A. Rawal and R. Miikkulainen, "From nodes to networks: Evolving recurrent neural networks," *arXiv preprint arXiv:1803.04439*, 2018.

[286] H. Jin, Q. Song, and X. Hu, "Auto-keras: Efficient neural architecture search with network morphism," *arXiv preprint arXiv:1806.10282*, vol. 5, 2018.

[287] T. Wei, C. Wang, Y. Rui, and C. W. Chen, "Network morphism," in *International conference on machine learning*, pp. 564–572, PMLR, 2016.

[288] S. Xie, H. Zheng, C. Liu, and L. Lin, "Snas: stochastic neural architecture search," *arXiv preprint arXiv:1812.09926*, 2018.

[289] G. Bender, P.-J. Kindermans, B. Zoph, V. Vasudevan, and Q. Le, "Understanding and simplifying one-shot architecture search," in *International conference on machine learning*, pp. 550–559, PMLR, 2018.

[290] H. Pham, M. Guan, B. Zoph, Q. Le, and J. Dean, "Efficient neural architecture search via parameters sharing," in *International conference on machine learning*, pp. 4095–4104, PMLR, 2018.

[291] Y. Gou, B. Li, Z. Liu, S. Yang, and X. Peng, "Clearer: Multi-scale neural architecture search for image restoration," *Advances in Neural Information Processing Systems*, vol. 33, pp. 17129–17140, 2020.

[292] L. Yao, H. Xu, W. Zhang, X. Liang, and Z. Li, "Sm-nas: Structural-to-modular neural architecture search for object detection," in *Proceedings of the AAAI conference on artificial intelligence*, vol. 34, pp. 12661–12668, 2020.

[293] C. Liu, L.-C. Chen, F. Schroff, H. Adam, W. Hua, A. L. Yuille, and L. Fei-Fei, "Auto-deeplab: Hierarchical neural architecture search for semantic image segmentation," in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pp. 82–92, 2019.

[294] B. Chen, G. Ghiasi, H. Liu, T.-Y. Lin, D. Kalenichenko, H. Adam, and Q. V. Le, "Mnasfpn: Learning latency-aware pyramid architecture for

object detection on mobile devices," in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pp. 13607–13616, 2020.

[295] Y. Hu, X. Wu, and R. He, "Tf-nas: Rethinking three search freedoms of latency-constrained differentiable neural architecture search," in *European Conference on Computer Vision*, pp. 123–139, Springer, 2020.

[296] Z. Lu, I. Whalen, V. Boddeti, Y. Dhebar, K. Deb, E. Goodman, and W. Banzhaf, "Nsga-net: neural architecture search using multi-objective genetic algorithm," in *Proceedings of the Genetic and Evolutionary Computation Conference*, pp. 419–427, 2019.

[297] C.-H. Hsu, S.-H. Chang, J.-H. Liang, H.-P. Chou, C.-H. Liu, S.-C. Chang, J.-Y. Pan, Y.-T. Chen, W. Wei, and D.-C. Juan, "Monas: Multi-objective neural architecture search using reinforcement learning," *arXiv preprint arXiv:1806.10332*, 2018.

[298] B. Wu, X. Dai, P. Zhang, Y. Wang, F. Sun, Y. Wu, Y. Tian, P. Vajda, Y. Jia, and K. Keutzer, "FBNet: Hardware-Aware Efficient ConvNet Design via Differentiable Neural Architecture Search," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 10734–10742, 2019.

[299] C. Banbury, C. Zhou, I. Fedorov, R. Matas, U. Thakker, D. Gope, V. Janapa Reddi, M. Mattina, and P. Whatmough, "Micronets: Neural network architectures for deploying tinyml applications on commodity microcontrollers," *Proceedings of Machine Learning and Systems*, vol. 3, pp. 517–532, 2021.

[300] X. Xia and W. Ding, "Hnas: Hierarchical neural architecture search on mobile devices," *arXiv preprint arXiv:2005.07564*, 2020.

[301] A. Wan, X. Dai, P. Zhang, Z. He, Y. Tian, S. Xie, B. Wu, M. Yu, T. Xu, K. Chen, *et al.*, "Fbnetv2: Differentiable neural architecture search for spatial and channel dimensions," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 12965–12974, 2020.

[302] L. Wang, S. Xie, T. Li, R. Fonseca, and Y. Tian, "Sample-efficient neural architecture search by learning action space," *arXiv preprint arXiv:1906.06832*, 2019.

[303] D. Stamoulis, R. Ding, D. Wang, D. Lymberopoulos, B. Priyantha, J. Liu,

and D. Marculescu, “Single-path nas: Designing hardware-efficient convnets in less than 4 hours,” in *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, pp. 481–497, Springer, 2019.

[304] C. Liu, L.-C. Chen, F. Schroff, H. Adam, W. Hua, A. L. Yuille, and L. Fei-Fei, “Auto-deeplab: Hierarchical neural architecture search for semantic image segmentation,” in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pp. 82–92, 2019.

[305] L. Li and A. Talwalkar, “Random search and reproducibility for neural architecture search,” in *Uncertainty in artificial intelligence*, pp. 367–377, PMLR, 2020.

[306] C. Ying, A. Klein, E. Christiansen, E. Real, K. Murphy, and F. Hutter, “Nas-bench-101: Towards reproducible neural architecture search,” in *International Conference on Machine Learning*, pp. 7105–7114, PMLR, 2019.

[307] X. Dong and Y. Yang, “Nas-bench-201: Extending the scope of reproducible neural architecture search,” *arXiv preprint arXiv:2001.00326*, 2020.

[308] J. Siems, L. Zimmer, A. Zela, J. Lukasik, M. Keuper, and F. Hutter, “Nas-bench-301 and the case for surrogate benchmarks for neural architecture search,” *arXiv preprint arXiv:2008.09777*, 2020.

[309] A. Klein, E. Christiansen, K. Murphy, and F. Hutter, “Towards reproducible neural architecture and hyperparameter search,” 2018.

[310] P. Ren, Y. Xiao, X. Chang, P.-Y. Huang, Z. Li, X. Chen, and X. Wang, “A comprehensive survey of neural architecture search: Challenges and solutions,” *ACM Computing Surveys (CSUR)*, vol. 54, no. 4, pp. 1–34, 2021. Publisher: ACM New York, NY, USA.

[311] Y. Liu, Y. Sun, B. Xue, M. Zhang, G. G. Yen, and K. C. Tan, “A survey on evolutionary neural architecture search,” *IEEE transactions on neural networks and learning systems*, 2021. Publisher: IEEE.

[312] S. A. Janowsky, “Pruning versus clipping in neural networks,” *Physical Review A*, vol. 39, no. 12, p. 6600, 1989.

[313] B. Hassibi and D. G. Stork, *Second order derivatives for network pruning: Optimal brain surgeon*. Morgan Kaufmann, 1993.

[314] S. Han, J. Pool, J. Tran, and W. J. Dally, "Learning both Weights and Connections for Efficient Neural Networks," *arXiv:1506.02626 [cs]*, Oct. 2015. arXiv: 1506.02626.

[315] Z. Liu, M. Sun, T. Zhou, G. Huang, and T. Darrell, "Rethinking the value of network pruning," *arXiv preprint arXiv:1810.05270*, 2018.

[316] T. Dettmers and L. Zettlemoyer, "Sparse networks from scratch: Faster training without losing performance," *arXiv preprint arXiv:1907.04840*, 2019.

[317] R. Yu, A. Li, C.-F. Chen, J.-H. Lai, V. I. Morariu, X. Han, M. Gao, C.-Y. Lin, and L. S. Davis, "Nisp: Pruning networks using neuron importance score propagation," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 9194–9203, 2018.

[318] H. Wang, C. Qin, Y. Zhang, and Y. Fu, "Neural pruning via growing regularization," *arXiv preprint arXiv:2012.09243*, 2020.

[319] J. Frankle, G. K. Dziugaite, D. M. Roy, and M. Carbin, "Stabilizing the Lottery Ticket Hypothesis," *arXiv:1903.01611 [cs, stat]*, July 2020. arXiv: 1903.01611.

[320] N. Lee, T. Ajanthan, and P. H. Torr, "Snip: Single-shot network pruning based on connection sensitivity," *arXiv preprint arXiv:1810.02340*, 2018.

[321] N. Lee, T. Ajanthan, S. Gould, and P. H. Torr, "A signal propagation perspective for pruning neural networks at initialization," *arXiv preprint arXiv:1906.06307*, 2019.

[322] W. Lei, H. Chen, and Y. Wu, "Compressing deep convolutional networks using k-means based on weights distribution," in *Proceedings of the 2nd International Conference on Intelligent Information Processing*, pp. 1–6, 2017.

[323] H. Li, A. Kadav, I. Durdanovic, H. Samet, and H. P. Graf, "Pruning Filters for Efficient ConvNets," *arXiv:1608.08710 [cs]*, Mar. 2017. arXiv: 1608.08710.

[324] M. Lin, L. Cao, S. Li, Q. Ye, Y. Tian, J. Liu, Q. Tian, and R. Ji, "Filter sketch for network pruning," *IEEE Transactions on Neural Networks and Learning Systems*, 2021.

[325] Y. He, P. Liu, Z. Wang, Z. Hu, and Y. Yang, "Filter pruning via geometric median for deep convolutional neural networks acceleration,"

in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 4340–4349, 2019.

[326] H. Hu, R. Peng, Y.-W. Tai, and C.-K. Tang, "Network trimming: A data-driven neuron pruning approach towards efficient deep architectures," *arXiv preprint arXiv:1607.03250*, 2016.

[327] Y. Guo, A. Yao, and Y. Chen, "Dynamic network surgery for efficient dnns," *arXiv preprint arXiv:1608.04493*, 2016.

[328] A. Polyak and L. Wolf, "Channel-level acceleration of deep face representations," *IEEE Access*, vol. 3, pp. 2163–2175, 2015.

[329] J.-H. Luo and J. Wu, "An entropy-based pruning method for cnn compression," *arXiv preprint arXiv:1706.05791*, 2017.

[330] J.-H. Luo, J. Wu, and W. Lin, "Thinet: A filter level pruning method for deep neural network compression," in *Proceedings of the IEEE international conference on computer vision*, pp. 5058–5066, 2017.

[331] Y. He, X. Zhang, and J. Sun, "Channel Pruning for Accelerating Very Deep Neural Networks," pp. 1389–1397, 2017.

[332] X. Ding, G. Ding, Y. Guo, J. Han, and C. Yan, "Approximated oracle filter pruning for destructive cnn width optimization," in *International Conference on Machine Learning*, pp. 1607–1616, PMLR, 2019.

[333] Z. Huang and N. Wang, "Data-driven sparse structure selection for deep neural networks," in *Proceedings of the European conference on computer vision (ECCV)*, pp. 304–320, 2018.

[334] B. Hassibi, D. G. Stork, and G. J. Wolff, "Optimal brain surgeon and general network pruning," in *IEEE international conference on neural networks*, pp. 293–299, IEEE, 1993.

[335] B. Hassibi, D. G. Stork, G. Wolff, and T. Watanabe, "Optimal Brain Surgeon: Extensions and performance comparison.," 1994.

[336] P. Molchanov, S. Tyree, T. Karras, T. Aila, and J. Kautz, "Pruning convolutional neural networks for resource efficient inference," *arXiv preprint arXiv:1611.06440*, 2016.

[337] P. Molchanov, A. Mallya, S. Tyree, I. Frosio, and J. Kautz, "Importance estimation for neural network pruning," in *Proceedings of the IEEE/CVF*

Conference on Computer Vision and Pattern Recognition, pp. 11264–11272, 2019.

[338] C. Wang, R. Grosse, S. Fidler, and G. Zhang, "Eigendamage: Structured pruning in the kronecker-factored eigenbasis," in *International Conference on Machine Learning*, pp. 6566–6575, PMLR, 2019.

[339] R. Grosse and J. Martens, "A kronecker-factored approximate fisher matrix for convolution layers," in *International Conference on Machine Learning*, pp. 573–582, PMLR, 2016.

[340] H. Peng, J. Wu, S. Chen, and J. Huang, "Collaborative channel pruning for deep networks," in *International Conference on Machine Learning*, pp. 5113–5122, PMLR, 2019.

[341] A. P. Engelbrecht, "A new pruning heuristic based on variance analysis of sensitivity information," *IEEE transactions on Neural Networks*, vol. 12, no. 6, pp. 1386–1399, 2001.

[342] A. RoyChowdhury, P. Sharma, E. Learned-Miller, and A. Roy, "Reducing duplicate filters in deep neural networks," in *NIPS workshop on Deep Learning: Bridging Theory and Practice*, vol. 1, p. 1, 2017.

[343] Y. Li, S. Lin, B. Zhang, J. Liu, D. Doermann, Y. Wu, F. Huang, and R. Ji, "Exploiting kernel sparsity and entropy for interpretable CNN compression," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 2800–2809, 2019.

[344] S. Son, S. Nah, and K. M. Lee, "Clustering convolutional kernels to compress deep neural networks," in *Proceedings of the European Conference on Computer Vision (ECCV)*, pp. 216–232, 2018.

[345] S. Srinivas and R. V. Babu, "Data-free parameter pruning for Deep Neural Networks," *arXiv:1507.06149 [cs]*, July 2015. arXiv: 1507.06149.

[346] Z. Mariet and S. Sra, "Diversity networks: Neural network compression using determinantal point processes," *arXiv preprint arXiv:1511.05077*, 2015.

[347] B. O. Ayinde, T. Inanc, and J. M. Zurada, "Redundant feature pruning for accelerated inference in deep neural networks," *Neural Networks*, vol. 118, pp. 148–158, 2019.

[348] Y. Zhang, C. Zhao, B. Ni, J. Zhang, and H. Deng, "Exploiting Channel Similarity for Accelerating Deep Convolutional Neural Networks," *arXiv preprint arXiv:1908.02620*, 2019.

[349] H. Wang, Q. Zhang, Y. Wang, and H. Hu, "Structured probabilistic pruning for convolutional neural network acceleration," *arXiv preprint arXiv:1709.06994*, 2017.

[350] E. Elsen, M. Dukhan, T. Gale, and K. Simonyan, "Fast Sparse ConvNets," *arXiv:1911.09723 [cs]*, Nov. 2019. arXiv: 1911.09723.

[351] T. Gale, E. Elsen, and S. Hooker, "The State of Sparsity in Deep Neural Networks," *arXiv:1902.09574 [cs, stat]*, Feb. 2019. arXiv: 1902.09574.

[352] D. Blalock, J. J. G. Ortiz, J. Frankle, and J. Guttag, "What is the state of neural network pruning?," *arXiv preprint arXiv:2003.03033*, 2020.

[353] H. Wang, C. Qin, Y. Zhang, and Y. Fu, "Emerging paradigms of neural network pruning," *arXiv preprint arXiv:2103.06460*, 2021.

[354] N. Hubens, M. Mancas, M. Decombas, M. Preda, T. Zaharia, B. Gosselin, and T. Dutoit, "An Experimental Study of the Impact of Pre-Training on the Pruning of a Convolutional Neural Network," in *Proceedings of the 3rd International Conference on Applications of Intelligent Systems*, pp. 1–6, 2020.

[355] H. Yu, S. Edunov, Y. Tian, and A. S. Morcos, "Playing the lottery with rewards and multiple languages: lottery tickets in rl and nlp," *arXiv preprint arXiv:1906.02768*, 2019.

[356] A. S. Morcos, H. Yu, M. Paganini, and Y. Tian, "One ticket to win them all: generalizing lottery ticket initializations across datasets and optimizers," *arXiv preprint arXiv:1906.02773*, 2019.

[357] C. Wang, G. Zhang, and R. Grosse, "Picking winning tickets before training by preserving gradient flow," *arXiv preprint arXiv:2002.07376*, 2020.

[358] V. Ramanujan, M. Wortsman, A. Kembhavi, A. Farhadi, and M. Rastegari, "What's Hidden in a Randomly Weighted Neural Network?," in *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 11890–11899, June 2020. ISSN: 2575-7075.

[359] H. Zhou, J. Lan, R. Liu, and J. Yosinski, "Deconstructing lottery tickets: Zeros, signs, and the supermask," *arXiv preprint arXiv:1905.01067*, 2019.

[360] E. Malach, G. Yehudai, S. Shalev-Schwartz, and O. Shamir, "Proving the lottery ticket hypothesis: Pruning is all you need," in *International Conference on Machine Learning*, pp. 6682–6691, PMLR, 2020.

[361] S. Han, X. Liu, H. Mao, J. Pu, A. Pedram, M. A. Horowitz, and W. J. Dally, "EIE: Efficient inference engine on compressed deep neural network," *ACM SIGARCH Computer Architecture News*, vol. 44, no. 3, pp. 243–254, 2016.

[362] S. Gray, A. Radford, and D. P. Kingma, "Gpu kernels for block-sparse weights," *arXiv preprint arXiv:1711.09224*, vol. 3, 2017.

[363] "XNNPACK," Oct. 2021. original-date: 2019-09-13T23:48:37Z.

[364] H. Mao, S. Han, J. Pool, W. Li, X. Liu, Y. Wang, and W. J. Dally, "Exploring the granularity of sparsity in convolutional neural networks," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*, pp. 13–20, 2017.

[365] L. Theis, I. Korshunova, A. Tejani, and F. Huszár, "Faster gaze prediction with dense networks and fisher pruning," *arXiv preprint arXiv:1801.05787*, 2018.

[366] A. Ashok, N. Rhinehart, F. Beainy, and K. M. Kitani, "N2n learning: Network to network compression via policy gradient reinforcement learning," *arXiv preprint arXiv:1709.06030*, 2017.

[367] T.-J. Yang, Y.-H. Chen, and V. Sze, "Designing Energy-Efficient Convolutional Neural Networks Using Energy-Aware Pruning," pp. 5687–5695, 2017.

[368] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going deeper with convolutions," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 1–9, 2015.

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

[370] S. Xie, R. Girshick, P. Dollár, Z. Tu, and K. He, "Aggregated residual transformations for deep neural networks," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 1492–1500, 2017.

[371] M. Tan and Q. V. Le, "Mixconv: Mixed depthwise convolutional kernels," *arXiv preprint arXiv:1907.09595*, 2019.

[372] Y. Xu, L. Xie, X. Zhang, X. Chen, G.-J. Qi, Q. Tian, and H. Xiong, "Pc-darts: Partial channel connections for memory-efficient architecture search," *arXiv preprint arXiv:1907.05737*, 2019.

[373] S. Jetley, N. A. Lord, N. Lee, and P. H. S. Torr, "Learn To Pay Attention," Apr. 2018. Number: arXiv:1804.02391 arXiv:1804.02391 [cs].

[374] B. Zhou, A. Khosla, A. Lapedriza, A. Oliva, and A. Torralba, "Learning deep features for discriminative localization," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 2921–2929, 2016.

[375] J. Hu, L. Shen, and G. Sun, "Squeeze-and-excitation networks," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 7132–7141, 2018.

[376] S. Woo, J. Park, J.-Y. Lee, and I. S. Kweon, "Cbam: Convolutional block attention module," in *Proceedings of the European conference on computer vision (ECCV)*, pp. 3–19, 2018.

[377] J. Park, S. Woo, J.-Y. Lee, and I. S. Kweon, "A simple and light-weight attention module for convolutional neural networks," *International journal of computer vision*, vol. 128, no. 4, pp. 783–798, 2020. Publisher: Springer.

[378] R. Saini, N. K. Jha, B. Das, S. Mittal, and C. K. Mohan, "Ulsam: Ultra-lightweight subspace attention module for compact convolutional neural networks," in *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision*, pp. 1627–1636, 2020.

[379] S. Hochreiter and J. Schmidhuber, "LSTM can solve hard long time lag problems," *Advances in neural information processing systems*, pp. 473–479, 1997.

[380] H. Salehinejad, S. Sankar, J. Barfett, E. Colak, and S. Valaee, "Recent Advances in Recurrent Neural Networks," Feb. 2018. Number: arXiv:1801.01078 arXiv:1801.01078 [cs].

[381] L. Jing, C. Gulcehre, J. Peurifoy, Y. Shen, M. Tegmark, M. Soljacic, and Y. Bengio, "Gated orthogonal recurrent units: On learning to forget," *Neural computation*, vol. 31, no. 4, pp. 765–783, 2019. Publisher: MIT Press One Rogers Street, Cambridge, MA 02142-1209, USA journals-info ....

[382] C.-J. Zhang, H.-Y. Wang, J. Zeng, L.-M. Ma, and L. Guan, "Tiny-rainnet: a deep convolutional neural network with bi-directional long short-term memory model for short-term rainfall prediction," *Meteorological Applications*, vol. 27, no. 5, p. e1956, 2020. Publisher: Wiley Online Library.

[383] I. Fedorov, M. Stamenovic, C. Jensen, L.-C. Yang, A. Mandell, Y. Gan, M. Mattina, and P. N. Whatmough, "TinyLSTMs: Efficient Neural Speech Enhancement for Hearing Aids," *arXiv:2005.11138 [cs, eess, stat]*, May 2020. arXiv: 2005.11138.

[384] U. Thakker, J. Beu, D. Gope, C. Zhou, I. Fedorov, G. Dasika, and M. Mattina, "Compressing RNNs for IoT devices by 15-38x using Kronecker Products," *arXiv:1906.02876 [cs, stat]*, Jan. 2020. arXiv: 1906.02876.

[385] F. Iandola, M. Moskewicz, S. Karayev, R. Girshick, T. Darrell, and K. Keutzer, "Densenet: Implementing efficient convnet descriptor pyramids," *arXiv preprint arXiv:1404.1869*, 2014.

[386] K. Han, Y. Wang, Q. Tian, J. Guo, C. Xu, and C. Xu, "Ghostnet: More features from cheap operations," in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pp. 1580–1589, 2020.

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

[388] Y. Tay, M. Dehghani, D. Bahri, and D. Metzler, "Efficient transformers: A survey," *ACM Computing Surveys (CSUR)*, 2020. Publisher: ACM New York, NY.

[389] "STM32Cube.AI: Convert Neural Networks into Optimized Code for STM32," Jan. 2019. Section: AI.

[390] A. Garofalo, G. Tagliavini, F. Conti, D. Rossi, and L. Benini, "XpulpNN: Accelerating quantized neural networks on RISC-V processors through ISA extensions," in *2020 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pp. 186–191, IEEE, 2020.

[391] L. Lai, N. Suda, and V. Chandra, "CMSIS-NN: Efficient Neural Network Kernels for Arm Cortex-M CPUs," Jan. 2018.

[392] T. Chen, T. Moreau, Z. Jiang, L. Zheng, E. Yan, H. Shen, M. Cowan, L. Wang, Y. Hu, and L. Ceze, "${$tvm$}$: An Automated ${$end-to-end$}$ Optimizing Compiler for Deep Learning," in *13th USENIX*

Symposium on Operating Systems Design and Implementation (OSDI 18), pp. 578–594, 2018.

[393] P. Warden and D. Situnayake, *Tinyml: Machine learning with tensorflow lite on arduino and ultra-low-power microcontrollers*. O'Reilly Media, 2019.

[394] "Use ONNX," May 2022. original-date: 2017-09-07T04:53:45Z.

[395] J. Nordby, "emlearn: Machine Learning inference engine for Microcontrollers and Embedded Devices," *Mar-2019*, 2019.

[396] T. Givargis, "Gravity: An Artificial Neural Network Compiler for Embedded Applications," in *2021 26th Asia and South Pacific Design Automation Conference (ASP-DAC)*, pp. 715–721, IEEE, 2021.

[397] X. Wang, M. Magno, L. Cavigelli, and L. Benini, "FANN-on-MCU: An open-source toolkit for energy-efficient neural network inference at the edge of the Internet of Things," *IEEE Internet of Things Journal*, vol. 7, no. 5, pp. 4403–4417, 2020. Publisher: IEEE.

[398] F. Sakr, F. Bellotti, R. Berta, and A. De Gloria, "Machine learning on mainstream microcontrollers," *Sensors*, vol. 20, no. 9, p. 2638, 2020. Publisher: Multidisciplinary Digital Publishing Institute.

[399] "oneDNN Documentation — oneDNN v2.4.0 documentation."

[400] X. Jiang, H. Wang, Y. Chen, Z. Wu, L. Wang, B. Zou, Y. Yang, Z. Cui, Y. Cai, and T. Yu, "Mnn: A universal and efficient inference engine," *Proceedings of Machine Learning and Systems*, vol. 2, pp. 1–13, 2020.

[401] Y. Chen, B. Zheng, Z. Zhang, Q. Wang, C. Shen, and Q. Zhang, "Deep learning on mobile and embedded devices: State-of-the-art, challenges, and future directions," *ACM Computing Surveys (CSUR)*, vol. 53, pp. 1–37, Aug. 2020. Publisher: ACM New York, NY, USA.

[402] A. Ignatov, R. Timofte, A. Kulik, S. Yang, K. Wang, F. Baum, M. Wu, L. Xu, and L. Van Gool, "AI Benchmark: All About Deep Learning on Smartphones in 2019," *arXiv:1910.06663 [cs]*, Oct. 2019. arXiv: 1910.06663.

[403] "ROCm™: Machine Learning."

[404] D. H. Noronha, B. Salehpour, and S. J. E. Wilton, "LeFlow: Enabling Flexible FPGA High-Level Synthesis of Tensorflow Deep Neural Networks," p. 8, 2018.

[405] Z. Jia, B. Tillman, M. Maggioni, and D. P. Scarpazza, "Dissecting the Graphcore IPU Architecture via Microbenchmarking," Tech. Rep. arXiv:1912.03413, arXiv, Dec. 2019. arXiv:1912.03413 [cs] type: article.

[406] R. Wimmer, P. Holleis, M. Kranz, and A. Schmidt, "Thracker-using capacitive sensing for gesture recognition," in *26th IEEE International Conference on Distributed Computing Systems Workshops (ICDCSW'06)*, pp. 64–64, IEEE, 2006.

[407] G. Singh, A. Nelson, R. Robucci, C. Patel, and N. Banerjee, "Inviz: Low-power personalized gesture recognition using wearable textile capacitive sensor arrays," in *2015 IEEE international conference on pervasive computing and communications (PerCom)*, pp. 198–206, IEEE, 2015.

[408] S. Escalera, I. Guyon, and V. Athitsos, *Gesture recognition*. Springer, 2017.

[409] W. E. Rhodes, "Capacitive sensing device for detecting passage of particles," Sept. 1969. Publisher: Google Patents.

[410] J. E. Shea, "Capacitive sensing device having a slidable probe," Jan. 1971. Publisher: Google Patents.

[411] A. Braun, S. Frank, M. Majewski, and X. Wang, "CapSeat: capacitive proximity sensing for automotive activity recognition," in *Proceedings of the 7th International Conference on Automotive User Interfaces and Interactive Vehicular Applications*, pp. 225–232, 2015.

[412] S. Frank and A. Kuijper, "HUDConCap-automotive head-up display controlled with capacitive proximity sensing," in *European Conference on Ambient Intelligence*, pp. 197–213, Springer, 2017.

[413] B. Osoinach, "Proximity capacitive sensor technology for touch sensing applications," *Freescale White Paper*, vol. 12, 2007.

[414] Y. Ye, J. Deng, S. Shen, Z. Hou, and Y. Liu, "A novel method for proximity detection of moving targets using a large-scale planar capacitive sensor system," *Sensors*, vol. 16, no. 5, p. 699, 2016. Publisher: Multidisciplinary Digital Publishing Institute.

[415] F. Althoff, R. Lindl, L. Walchshausl, and S. Hoch, "Robust multimodal hand-and head gesture recognition for controlling automotive infotainment systems," *VDI BERICHTE*, vol. 1919, p. 187, 2005. Publisher: Citeseer.

[416] M. Stecher, E. Baseler, L. Draxler, L. Fricke, B. Michel, A. Zimmermann, and K. Bengler, "Tracking down the intuitiveness of gesture interaction in the truck domain," *Procedia Manufacturing*, vol. 3, pp. 3176–3183, 2015. Publisher: Elsevier.

[417] H. V. Le, S. Mayer, and N. Henze, "Investigating the feasibility of finger identification on capacitive touchscreens using deep learning," in *Proceedings of the 24th International Conference on Intelligent User Interfaces*, pp. 637–649, 2019.

[418] S. Venkataramani, A. Ranjan, K. Roy, and A. Raghunathan, "AxNN: Energy-efficient neuromorphic systems using approximate computing," in *2014 IEEE/ACM International Symposium on Low Power Electronics and Design (ISLPED)*, pp. 27–32, IEEE, 2014.

[419] N. Rotem, J. Fix, S. Abdulrasool, G. Catron, S. Deng, R. Dzhabarov, N. Gibson, J. Hegeman, M. Lele, and R. Levenstein, "Glow: Graph lowering compiler techniques for neural networks," *arXiv preprint arXiv:1805.00907*, 2018.

[420] "Arm NN SDK – Arm®."

[421] M. Loni, S. Sinaei, A. Zoljodi, M. Daneshtalab, and M. Sjödin, "DeepMaker: A multi-objective optimization framework for deep neural networks in embedded systems," *Microprocessors and Microsystems*, vol. 73, p. 102989, 2020. Publisher: Elsevier.

[422] E. Li, Z. Zhou, and X. Chen, "Edge intelligence: On-demand deep learning model co-inference with device-edge synergy," in *Proceedings of the 2018 Workshop on Mobile Edge Communications*, pp. 31–36, 2018.

[423] B. Lu, J. Yang, L. Y. Chen, and S. Ren, "Automating deep neural network model selection for edge inference," in *2019 IEEE First International Conference on Cognitive Machine Intelligence (CogMI)*, pp. 184–193, IEEE, 2019.

[424] R. Binns, M. Veale, M. V. Kleek, and N. Shadbolt, "Like trainer, like bot? Inheritance of bias in algorithmic content moderation," in *International conference on social informatics*, pp. 405–415, Springer, 2017.

[425] P. Vincent, H. Larochelle, Y. Bengio, and P.-A. Manzagol, "Extracting and composing robust features with denoising autoencoders," in *Proceedings of the 25th international conference on Machine learning*, pp. 1096–1103, 2008.

[426] P. Baldi, "Autoencoders, unsupervised learning, and deep architectures," in *Proceedings of ICML workshop on unsupervised and transfer learning*, pp. 37–49, JMLR Workshop and Conference Proceedings, 2012.

[427] A. Makhzani, J. Shlens, N. Jaitly, I. Goodfellow, and B. Frey, "Adversarial autoencoders," *arXiv preprint arXiv:1511.05644*, 2015.

[428] R. R. Sokal, "The principles and practice of numerical taxonomy," *Taxon*, pp. 190–199, 1963. Publisher: JSTOR.

[429] L. Rokach and O. Maimon, "Clustering methods," in *Data mining and knowledge discovery handbook*, pp. 321–352, Springer, 2005.

[430] L. Du, "An overview of mobile capacitive touch technologies trends," *arXiv preprint arXiv:1612.08227*, 2016.

[431] T. Grosse-Puppendahl, C. Holz, G. Cohn, R. Wimmer, O. Bechtold, S. Hodges, M. S. Reynolds, and J. R. Smith, "Finding common ground: A survey of capacitive sensing in human-computer interaction," in *Proceedings of the 2017 CHI conference on human factors in computing systems*, pp. 3293–3315, 2017.

[432] N. Henze, E. Rukzio, and S. Boll, "100,000,000 taps: analysis and improvement of touch performance in the large," in *Proceedings of the 13th International Conference on Human Computer Interaction with Mobile Devices and Services*, pp. 133–142, 2011.

[433] D. Weir, S. Rogers, R. Murray-Smith, and M. Löchtefeld, "A user-specific machine learning approach for improving touch accuracy on mobile devices," in *Proceedings of the 25th annual ACM symposium on User interface software and technology*, pp. 465–476, 2012.

[434] T. Fischer, M. Etchart, and E. Biempica, "Frame-level proximity and touch recognition using capacitive sensing and semi-supervised sequential modeling," in *2018 IEEE 28th International Workshop on Machine Learning for Signal Processing (MLSP)*, pp. 1–6, IEEE, 2018.

[435] M. M. Jung, R. Poppe, M. Poel, and D. K. Heylen, "Touching the void–introducing CoST: corpus of social touch," in *Proceedings of the 16th International Conference on Multimodal Interaction*, pp. 120–127, 2014.

[436] "Kernel class for arc kernel by BCJuan · Pull Request #1027 · cornellius-gp/gpytorch."

[437] J. R. Gardner, G. Pleiss, D. Bindel, K. Q. Weinberger, and A. G. Wilson, “GPyTorch: Blackbox Matrix-Matrix Gaussian Process Inference with GPU Acceleration,” *arXiv:1809.11165 [cs, stat]*, June 2021. arXiv: 1809.11165.

[438] A. Moschetti, L. Fiorini, D. Esposito, P. Dario, and F. Cavallo, “Toward an unsupervised approach for daily gesture recognition in assisted living applications,” *IEEE sensors journal*, vol. 17, no. 24, pp. 8395–8403, 2017. Publisher: IEEE.

[439] C. Song, F. Liu, Y. Huang, L. Wang, and T. Tan, “Auto-encoder based data clustering,” in *Iberoamerican congress on pattern recognition*, pp. 117–124, Springer, 2013.

[440] Z. Chen and R. C. Luo, “Design and implementation of capacitive proximity sensor using microelectromechanical systems technology,” *IEEE Transactions on Industrial Electronics*, vol. 45, no. 6, pp. 886–894, 1998. Publisher: IEEE.

[441] D. Kurup, W. Joseph, G. Vermeeren, and L. Martens, “In-body path loss model for homogeneous human tissues,” *IEEE Transactions on Electromagnetic Compatibility*, vol. 54, no. 3, pp. 556–564, 2011. Publisher: IEEE.

[442] S.-H. Lee, J.-S. An, S.-K. Hong, and O.-K. Kwon, “In-cell capacitive touch panel structures and their readout circuits,” in *2016 23rd International Workshop on Active-Matrix Flatpanel Displays and Devices (AM-FPD)*, pp. 258–261, IEEE, 2016.

[443] T. Grosse-Puppendahl and A. Braun, “Honeyfish-a high resolution gesture recognition system based on capacitive proximity sensing,” in *Embedded World Conference*, vol. 12, p. 5, 2012.

[444] F. Miedl and T. Tille, “3-D surface-integrated touch-sensor system for automotive HMI applications,” *IEEE/ASME Transactions on Mechatronics*, vol. 21, no. 2, pp. 787–794, 2015. Publisher: IEEE.

[445] B. Liu, Z. Hoseini, K.-S. Lee, and Y.-M. Lee, “On-chip touch sensor readout circuit using passive sigma-delta modulator capacitance-to-digital converter,” *IEEE Sensors Journal*, vol. 15, no. 7, pp. 3893–3902, 2015. Publisher: IEEE.

[446] S.-Y. Peng, M. S. Qureshi, P. E. Hasler, N. A. Hall, and F. L. Degertekin, “High SNR capacitive sensing transducer,” in *2006 IEEE International Symposium on Circuits and Systems*, pp. 4–pp, IEEE, 2006.

[447] K. Watanabe and W.-S. Chung, "A switched-capacitor interface for intelligent capacitive transducers," *IEEE transactions on instrumentation and measurement*, no. 4, pp. 472–476, 1986. Publisher: IEEE.

[448] X. Zhang and P. K. Chan, "A low-power switched-capacitor capacitive transducer with high resolution," *IEEE Transactions on Instrumentation and Measurement*, vol. 57, no. 7, pp. 1492–1499, 2008. Publisher: IEEE.

[449] H. Philipp, "Charge transfer sensing," *Sensor Review*, 1999. Publisher: MCB UP Ltd.

[450] J. E. Gaitán-Pitre, M. Gasulla, and R. Pallàs-Areny, "Direct interface for capacitive sensors based on the charge transfer method," in *2007 IEEE Instrumentation & Measurement Technology Conference IMTC 2007*, pp. 1–5, IEEE, 2007.

[451] O. Lopez-Lapeña, E. Serrano-Finetti, and O. Casas, "Calibration-less direct capacitor-to-microcontroller interface," *Electronics Letters*, vol. 52, no. 4, pp. 289–291, 2016. Publisher: IET.

[452] U. Borgmann and C. C. LEXOW, "Verfahren zum messen eines kapazitätswertes," Nov. 2015.

[453] J. E. Gaitán-Pitre, M. Gasulla, and R. Pallas-Areny, "Analysis of a direct interface circuit for capacitive sensors," *IEEE Transactions on Instrumentation and Measurement*, vol. 58, no. 9, pp. 2931–2937, 2009. Publisher: IEEE.

[454] R. Schweigert, J. Leusmann, S. Hagenmayer, M. Weiß, H. V. Le, S. Mayer, and A. Bulling, "Knuckletouch: Enabling knuckle gestures on capacitive touchscreens using deep learning," in *Proceedings of Mensch Und Computer 2019*, pp. 387–397, 2019.

[455] A. R. Jensen, *Clocking the mind: Mental chronometry and individual differences*. Elsevier, 2006.

[456] D. E. Meyer, S. Yantis, A. M. Osman, and J. K. Smith, "Temporal properties of human information processing: Tests of discrete versus continuous models," *Cognitive Psychology*, vol. 17, no. 4, pp. 445–518, 1985. Publisher: Elsevier.

[457] Y. Yin and R. Davis, "Real-time continuous gesture recognition for natural human-computer interaction," in *2014 IEEE Symposium on Visual*

Languages and Human-Centric Computing (VL/HCC), pp. 113–120, IEEE, 2014.

[458] Y. Song, D. Demirdjian, and R. Davis, "Continuous body and hand gesture recognition for natural human-computer interaction," *ACM Transactions on Interactive Intelligent Systems (TiiS)*, vol. 2, no. 1, pp. 1–28, 2012. Publisher: ACM New York, NY, USA.

[459] P. Fryzlewicz, "Wild binary segmentation for multiple change-point detection," *The Annals of Statistics*, vol. 42, no. 6, pp. 2243–2281, 2014. Publisher: Institute of Mathematical Statistics.

[460] F. J. Pineda, "Generalization of back-propagation to recurrent neural networks," *Physical review letters*, vol. 59, no. 19, p. 2229, 1987. Publisher: APS.

[461] K. Cho, B. Van Merriënboer, D. Bahdanau, and Y. Bengio, "On the properties of neural machine translation: Encoder-decoder approaches," *arXiv preprint arXiv:1409.1259*, 2014.

[462] J. Bergstra and Y. Bengio, "Random search for hyper-parameter optimization.," *Journal of machine learning research*, vol. 13, no. 2, 2012.

[463] E. Brochu, V. M. Cora, and N. De Freitas, "A tutorial on Bayesian optimization of expensive cost functions, with application to active user modeling and hierarchical reinforcement learning," *arXiv preprint arXiv:1012.2599*, 2010.

[464] P. I. Frazier, "A tutorial on Bayesian optimization," *arXiv preprint arXiv:1807.02811*, 2018.

[465] P. I. Frazier, "Bayesian optimization," in *Recent advances in optimization and modeling of contemporary problems*, pp. 255–278, Informs, 2018.

[466] J. Bergstra, D. Yamins, and D. D. Cox, "Hyperopt: A python library for optimizing the hyperparameters of machine learning algorithms," in *Proceedings of the 12th Python in science conference*, vol. 13, p. 20, Citeseer, 2013.

[467] R. J. Williams and J. Peng, "An efficient gradient-based algorithm for on-line training of recurrent network trajectories," *Neural computation*, vol. 2, no. 4, pp. 490–501, 1990. Publisher: MIT Press One Rogers Street, Cambridge, MA 02142-1209, USA journals-info . . . .

[468] R. Takada, B. Shizuki, and J. Tanaka, "MonoTouch: Single capacitive touch sensor that differentiates touch gestures," in *Proceedings of the 2016 CHI Conference Extended Abstracts on Human Factors in Computing Systems*, pp. 2736–2743, 2016.

[469] W.-C. Chuang, W.-J. Hwang, T.-M. Tai, D.-R. Huang, and Y.-J. Jhang, "Continuous finger gesture recognition based on flex sensors," *Sensors*, vol. 19, no. 18, p. 3986, 2019. Publisher: Multidisciplinary Digital Publishing Institute.

[470] H.-R. Tsai, M.-C. Hsiu, J.-C. Hsiao, L.-T. Huang, M. Chen, and Y.-P. Hung, "TouchRing: subtle and always-available input using a multi-touch ring," in *Proceedings of the 18th International Conference on Human-Computer Interaction with Mobile Devices and Services Adjunct*, pp. 891–898, 2016.

[471] B. Shahriari, K. Swersky, Z. Wang, R. P. Adams, and N. De Freitas, "Taking the human out of the loop: A review of Bayesian optimization," *Proceedings of the IEEE*, vol. 104, no. 1, pp. 148–175, 2015. Publisher: IEEE.

[472] B. Shahriari, Z. Wang, M. W. Hoffman, A. Bouchard-Côté, and N. de Freitas, "An entropy search portfolio for Bayesian optimization," *arXiv preprint arXiv:1406.4625*, 2014.

[473] R. B. Miller, "Response time in man-computer conversational transactions," in *Proceedings of the December 9-11, 1968, fall joint computer conference, part I*, pp. 267–277, 1968.

[474] V.-C. Ta, W. Johal, M. Portaz, E. Castelli, and D. Vaufreydaz, "The Grenoble system for the social touch challenge at ICMI 2015," in *Proceedings of the 2015 ACM on International Conference on Multimodal Interaction*, pp. 391–398, 2015.

[475] D. Hughes, A. Krauthammer, and N. Correll, "Recognizing social touch gestures using recurrent and convolutional neural networks," in *2017 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 2315–2321, IEEE, 2017.

[476] E. Bakshy, L. Dworkin, B. Karrer, K. Kashin, B. Letham, A. Murthy, and S. Singh, "AE: A domain-agnostic platform for adaptive experimentation," in *Conference on Neural Information Processing Systems*, pp. 1–8, 2018.

[477] C. Jose, P. Goyal, P. Aggrwal, and M. Varma, “Local deep kernel learning for efficient non-linear svm prediction,” in *International conference on machine learning*, pp. 486–494, PMLR, 2013.

[478] U. Borgmann, “Method for detecting contact on a capacitive sensor element,” Sept. 2020. Publisher: Google Patents.

[479] D. Arthur and S. Vassilvitskii, “k-means++: The advantages of careful seeding,” tech. rep., Stanford, 2006.

[480] A. Rosenberg and J. Hirschberg, “V-measure: A conditional entropy-based external cluster evaluation measure,” in *Proceedings of the 2007 joint conference on empirical methods in natural language processing and computational natural language learning (EMNLP-CoNLL)*, pp. 410–420, 2007.

[481] M. Balandat, B. Karrer, D. Jiang, S. Daulton, B. Letham, A. G. Wilson, and E. Bakshy, “BoTorch: a framework for efficient Monte-Carlo Bayesian optimization,” *Advances in neural information processing systems*, vol. 33, pp. 21524–21538, 2020.

[482] L. Van der Maaten and G. Hinton, “Visualizing data using t-SNE.,” *Journal of machine learning research*, vol. 9, no. 11, 2008.

[483] V. Prokhorov, E. Shareghi, Y. Li, M. T. Pilehvar, and N. Collier, “On the importance of the Kullback-Leibler divergence term in variational autoencoders for text generation,” *arXiv preprint arXiv:1909.13668*, 2019.

[484] E. Reinhard, W. Heidrich, P. Debevec, S. Pattanaik, G. Ward, and K. Myszkowski, *High dynamic range imaging: acquisition, display, and image-based lighting*. Morgan Kaufmann, 2010.

[485] A. O. Akyüz, R. Fleming, B. E. Riecke, E. Reinhard, and H. H. Bülthoff, “Do HDR displays support LDR content? A psychophysical evaluation,” *ACM Transactions on Graphics (TOG)*, vol. 26, no. 3, pp. 38–es, 2007.

[486] P. Hanhart, P. Korshunov, and T. Ebrahimi, “Subjective evaluation of higher dynamic range video,” in *Applications of Digital Image Processing XXXVII*, vol. 9217, p. 92170L, International Society for Optics and Photonics, 2014.

[487] S. , “HDR TV shipments worldwide by region 2016-2019,” 2020. Publication Title: Statista.

[488] T. Akenine-Möller, E. Haines, and N. Hoffman, *Real-Time Rendering*. CRC Press, Jan. 2019.

[489] P. E. Debevec and J. Malik, "Recovering high dynamic range radiance maps from photographs," in *Proceedings of the 24th annual conference on Computer graphics and interactive techniques*, pp. 369–378, 1997.

[490] O. Gallo, A. Troccoli, J. Hu, K. Pulli, and J. Kautz, "Locally non-rigid registration for mobile HDR photography," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*, pp. 49–56, 2015.

[491] R. Mantiuk, S. Daly, and L. Kerofsky, "Display adaptive tone mapping," *ACM Transactions on Graphics (TOG)*, vol. 27, no. 3, pp. 1–10, 2008.

[492] F. Drago, K. Myszkowski, T. Annen, and N. Chiba, "Adaptive logarithmic mapping for displaying high contrast scenes," in *Computer graphics forum*, vol. 22, pp. 419–426, Wiley Online Library, 2003.

[493] E. Reinhard, M. Stark, P. Shirley, and J. Ferwerda, "Photographic tone reproduction for digital images," in *Proceedings of the 29th annual conference on Computer graphics and interactive techniques*, pp. 267–276, 2002.

[494] G. Eilertsen, J. Unger, and R. K. Mantiuk, "Evaluation of tone mapping operators for HDR video," in *High Dynamic Range Video*, pp. 185–207, Elsevier, 2016.

[495] F. Banterle, P. Ledda, K. Debattista, and A. Chalmers, "Inverse tone mapping," in *Proc. of the 4th Int. Conf. on Computer Graphics and Interactive Techniques in Australasia and Southeast Asia*, pp. 349–356, 2006.

[496] A. G. Rempel, M. Trentacoste, H. Seetzen, H. D. Young, W. Heidrich, L. Whitehead, and G. Ward, "Ldr2hdr: on-the-fly reverse tone mapping of legacy video and photographs," *ACM transactions on graphics (TOG)*, vol. 26, no. 3, pp. 39–es, 2007.

[497] P.-H. Kuo, C.-S. Tang, and S.-Y. Chien, "Content-adaptive inverse tone mapping," in *2012 Visual Communications and Image Processing*, pp. 1–6, IEEE, 2012.

[498] Y. Huo, F. Yang, L. Dong, and V. Brost, "Physiological inverse tone mapping based on retina response," *The Visual Computer*, vol. 30, no. 5, pp. 507–517, 2014.

[499] B. Masia, S. Agustin, R. W. Fleming, O. Sorkine, and D. Gutierrez, "Evaluation of reverse tone mapping through varying exposure conditions," *ACM Transactions on Graphics (TOG)*, vol. 28, no. 5, pp. 1–8, 2009.

[500] F. Banterle, P. Ledda, K. Debattista, M. Bloj, A. Artusi, and A. Chalmers, "A psychophysical evaluation of inverse tone mapping techniques," in *Computer Graphics Forum*, vol. 28, pp. 13–25, Wiley Online Library, 2009.

[501] A. Serrano, F. Heide, D. Gutierrez, G. Wetzstein, and B. Masia, "Convolutional sparse coding for high dynamic range imaging," in *Computer Graphics Forum*, vol. 35, pp. 153–163, Wiley Online Library, 2016.

[502] J. H. Kim, S. Lee, S. Jo, and S.-J. Kang, "End-to-End Differentiable Learning to HDR Image Synthesis for Multi-exposure Images," *arXiv preprint arXiv:2006.15833*, 2020.

[503] M. C. , "Auto HDR Preview for PC Available Today," Mar. 2021. Publication Title: DirectX Developer Blog.

[504] Y. Endo, Y. Kanamori, and J. Mitani, "Deep reverse tone mapping," *ACM Transactions on Graphics (TOG)*, vol. 36, no. 6, pp. 1–10, 2017.

[505] S. Lee, G. H. An, and S.-J. Kang, "Deep chain hdri: Reconstructing a high dynamic range image from a single low dynamic range image," *IEEE Access*, vol. 6, pp. 49913–49924, 2018.

[506] B. Masia, A. Serrano, and D. Gutierrez, "Dynamic range expansion based on image statistics," *Multimedia Tools and Applications*, vol. 76, pp. 631–648, Jan. 2017.

[507] C. Zhou, H. Zhao, J. Han, C. Xu, C. Xu, T. Huang, and B. Shi, "UnModNet: Learning to Unwrap a Modulo Image for High Dynamic Range Imaging," *Advances in Neural Information Processing Systems*, vol. 33, 2020.

[508] S. Ning, H. Xu, L. Song, R. Xie, and W. Zhang, "Learning an inverse tone mapping network with a generative adversarial regularizer," in *2018 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pp. 1383–1387, IEEE, 2018.

[509] X. Yang, K. Xu, Y. Song, Q. Zhang, X. Wei, and R. W. Lau, "Image correction via deep reciprocating HDR transformation," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 1798–1807, 2018.

[510] J. Zhang and J.-F. Lalonde, "Learning high dynamic range from outdoor panoramas," in *Proceedings of the IEEE International Conference on Computer Vision*, pp. 4519–4528, 2017.

[511] A. G. Vien and C. Lee, "Single-Shot High Dynamic Range Imaging via Multiscale Convolutional Neural Network," *IEEE Access*, 2021.

[512] N. Ye, Y. Huo, S. Liu, and H. Li, "Single Exposure High Dynamic Range Image Reconstruction Based on Deep Dual-Branch Network," *IEEE Access*, vol. 9, pp. 9610–9624, 2021.

[513] S. M. A. Sharif, R. A. Naqvi, M. Biswas, and K. Sungjun, "A Two-stage Deep Network for High Dynamic Range Image Reconstruction," *arXiv preprint arXiv:2104.09386*, 2021.

[514] E. Pérez-Pellitero, S. Catley-Chandar, A. Leonardis, and R. Timofte, "NTIRE 2021 challenge on high dynamic range imaging: Dataset, methods and results," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 691–700, 2021.

[515] K. A. Akhil and C. V. Jiji, "Single image hdr synthesis using a densely connected dilated convnet," in *IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops*, 2021.

[516] G. Chen, L. Zhang, M. Sun, Y. Gao, P. N. Michelini, and Y. Wu, "Single-image hdr reconstruction with task-specific network based on channel adaptive RDN," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 398–403, 2021.

[517] E. Pan and A. Vento, "MetaHDR: Model-Agnostic Meta-Learning for HDR Image Reconstruction," *arXiv preprint arXiv:2103.12545*, 2021.

[518] Q. Yan, D. Gong, Q. Shi, A. van den Hengel, C. Shen, I. Reid, and Y. Zhang, "Attention-Guided Network for Ghost-Free High Dynamic Range Imaging," in *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 1751–1760, IEEE, 2019.

[519] Y. Niu, J. Wu, W. Liu, W. Guo, and R. W. Lau, "HDR-GAN: HDR image reconstruction from multi-exposed ldr images with large motions," *arXiv preprint arXiv:2007.01628*, 2020.

[520] Q. Yan, L. Zhang, Y. Liu, Y. Zhu, J. Sun, Q. Shi, and Y. Zhang, "Deep HDR Imaging via A Non-Local Network," *IEEE Transactions on Image Processing*, vol. 29, pp. 4308–4322, 2020.

[521] S. Wu, J. Xu, Y.-W. Tai, and C.-K. Tang, "Deep high dynamic range imaging with large foreground motions," in *Proceedings of the European Conference on Computer Vision (ECCV)*, pp. 117–132, 2018.

[522] N. K. Kalantari and R. Ramamoorthi, “Deep high dynamic range imaging of dynamic scenes.,” *ACM Trans. Graph.*, vol. 36, no. 4, pp. 144–1, 2017.

[523] Z. Liu, W. Lin, X. Li, Q. Rao, T. Jiang, M. Han, H. Fan, J. Sun, and S. Liu, “ADNet: Attention-guided Deformable Convolutional Network for High Dynamic Range Imaging,” *arXiv preprint arXiv:2105.10697*, 2021.

[524] K. R. Prabhakar, G. Senthil, S. Agrawal, R. V. Babu, and R. K. S. S. Gorthi, “Labeled From Unlabeled: Exploiting Unlabeled Data for Few-Shot Deep HDR Deghosting,” in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 4875–4885, June 2021.

[525] S. Lee, G. Hwan An, and S.-J. Kang, “Deep recursive hdri: Inverse tone mapping using generative adversarial networks,” in *Proceedings of the European Conference on Computer Vision (ECCV)*, pp. 596–611, 2018.

[526] S. Sudhakaran, S. Escalera, and O. Lanz, “Gate-Shift Networks for Video Action Recognition,” in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 1102–1111, June 2020.

[527] T. Liu, “Depth-wise Separable Convolutions: Performance Investigations,” 2020.

[528] J. Lee, D. Kang, and S. Ha, “S3nas: Fast npu-aware neural architecture search methodology,” *arXiv preprint arXiv:2009.02009*, 2020.

[529] Y. Choi and M. Rhu, “Prema: A predictive multi-task scheduling algorithm for preemptible neural processing units,” in *2020 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, pp. 220–233, IEEE, 2020.

[530] O. Ronneberger, P. Fischer, and T. Brox, “U-net: Convolutional networks for biomedical image segmentation,” in *International Conference on Medical image computing and computer-assisted intervention*, pp. 234–241, Springer, 2015.

[531] Z. Zhou, M. M. R. Siddiquee, N. Tajbakhsh, and J. Liang, “Unet++: A nested u-net architecture for medical image segmentation,” in *Deep learning in medical image analysis and multimodal learning for clinical decision support*, pp. 3–11, Springer, 2018.

[532] X. Huang and S. Belongie, “Arbitrary style transfer in real-time with adaptive instance normalization,” in *Proceedings of the IEEE International Conference on Computer Vision*, pp. 1501–1510, 2017.

[533] C. Baskin, N. Liss, E. Zheltonozhskii, A. M. Bronstein, and A. Mendelson, “Streaming architecture for large-scale quantized neural networks on an FPGA-based dataflow platform,” in *2018 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*, pp. 162–169, IEEE, 2018.

[534] P. Gysel, J. Pimentel, M. Motamedi, and S. Ghiasi, “Ristretto: A Framework for Empirical Study of Resource-Efficient Inference in Convolutional Neural Networks,” *IEEE Transactions on Neural Networks and Learning Systems*, vol. 29, pp. 5784–5789, Nov. 2018.

[535] J. Johnson, A. Alahi, and L. Fei-Fei, “Perceptual losses for real-time style transfer and super-resolution,” in *European conference on computer vision*, pp. 694–711, Springer, 2016.

[536] B. Funt and L. Shi, “The effect of exposure on MaxRGB color constancy,” in *Human Vision and Electronic Imaging XV*, vol. 7527, p. 75270Y, International Society for Optics and Photonics, 2010.

[537] G. Ward, “High dynamic range image encodings,” 2006.

[538] R. M. , “PFSTools. High Dynamic Range Images and Videos,” 2015.

[539] S. W. Hasinoff, D. Sharlet, R. Geiss, A. Adams, J. T. Barron, F. Kainz, J. Chen, and M. Levoy, “Burst photography for high dynamic range and low-light imaging on mobile cameras,” *ACM Transactions on Graphics*, vol. 35, pp. 192:1–192:12, Nov. 2016.

[540] H. Nemoto, P. Korshunov, P. Hanhart, and T. Ebrahimi, “Visual attention in LDR and HDR images,” in *9th International Workshop on Video Processing and Quality Metrics for Consumer Electronics (VPQM)*, 2015.

[541] D.-T. Dang-Nguyen, C. Pasquini, V. Conotter, and G. Boato, “Raise: A raw images dataset for digital image forensics,” in *Proceedings of the 6th ACM Multimedia Systems Conference*, pp. 219–224, 2015.

[542] O. S. V. F. , “OpenCV,” 2021. Publication Title: OpenCV.

[543] R. Mantiuk, K. J. Kim, A. G. Rempel, and W. Heidrich, "HDR-VDP-2: A calibrated visual metric for visibility and quality predictions in all luminance conditions," *ACM Transactions on graphics (TOG)*, vol. 30, no. 4, pp. 1–14, 2011.

[544] T. , "Performance measurement \textbar TensorFlow Lite," 2021.

[545] "Post-training integer quantization with int16 activations," 2021.

[546] X. Chen, Y. Liu, Z. Zhang, Y. Qiao, and C. Dong, "HDRUNet: Single Image HDR Reconstruction with Denoising and Dequantization," *arXiv:2105.13084*, 2021.

[547] M. S. Santos, T. I. Ren, and N. K. Kalantari, "Single image HDR reconstruction using a CNN with masked features and perceptual loss," *arXiv preprint arXiv:2005.07335*, 2020.

[548] U. Pastorino, M. Silva, S. Sestini, F. Sabia, M. Boeri, A. Cantarutti, N. Sverzellati, G. Sozzi, G. Corrao, and A. Marchianò, "Prolonged lung cancer screening reduced 10-year mortality in the MILD trial: new confirmation of lung cancer screening efficacy," *Annals of Oncology*, vol. 30, no. 7, pp. 1162–1169, 2019.

[549] H. J. de Koning, C. M. van der Aalst, P. A. de Jong, E. T. Scholten, K. Nackaerts, M. A. Heuvelmans, J.-W. J. Lammers, C. Weenink, U. Yousaf-Khan, and N. Horeweg, "Reduced lung-cancer mortality with volume CT screening in a randomized trial," *New England Journal of Medicine*, vol. 382, no. 6, pp. 503–513, 2020.

[550] F. Asano, R. Eberhardt, and F. J. F. Herth, "Virtual bronchoscopic navigation for peripheral pulmonary lesions," *Respiration; International Review of Thoracic Diseases*, vol. 88, no. 5, pp. 430–440, 2014.

[551] T. Ishiwata, A. Gregor, T. Inage, and K. Yasufuku, "Advances in interventional diagnostic bronchoscopy for peripheral pulmonary lesions," *Expert review of respiratory medicine*, vol. 13, no. 9, pp. 885–897, 2019.

[552] Y. Han, H. J. Kim, K. A. Kong, S. J. Kim, S. H. Lee, Y. J. Ryu, J. H. Lee, Y. Kim, S. S. Shim, and J. H. Chang, "Diagnosis of small pulmonary lesions by transbronchial lung biopsy with radial endobronchial ultrasound and virtual bronchoscopic navigation versus CT-guided transthoracic needle biopsy: A systematic review and meta-analysis," *PLOS ONE*, vol. 13, p. e0191590, Jan. 2018.

[553] M. K. Gould, J. Donington, W. R. Lynch, P. J. Mazzone, D. E. Midthun, D. P. Naidich, and R. S. Wiener, "Evaluation of individuals with pulmonary nodules: When is it lung cancer?: Diagnosis and management of lung cancer: American College of Chest Physicians evidence-based clinical practice guidelines," *Chest*, vol. 143, no. 5, pp. e93S–e120S, 2013.

[554] K. Mori, D. Deguchi, T. Kitasaka, Y. Suenaga, Y. Hasegawa, K. Imaizumi, and H. Takabatake, "Improvement of accuracy of marker-free bronchoscope tracking using electromagnetic tracker based on bronchial branch information," in *International Conference on Medical Image Computing and Computer-Assisted Intervention*, pp. 535–542, Springer, 2008.

[555] A. Skalski, M. Socha, M. Duplaga, K. Duda, and T. Zieliński, "3D segmentation and visualisation of mediastinal structures adjacent to tracheobronchial tree from CT data," in *Information Technologies in Biomedicine*, pp. 523–534, Springer, 2010.

[556] D. Gil, C. Sanchez, A. Borras, M. Diez-Ferrer, and A. Rosell, "Segmentation of distal airways using structural analysis," *PLOS ONE*, vol. 14, p. e0226006, Dec. 2019.

[557] J.-C. Chien, J.-D. Lee, E. Su, and S.-H. Li, "A Bronchoscope Localization Method Using an Augmented Reality Co-Display of Real Bronchoscopy Images with a Virtual 3D Bronchial Tree Model," *Sensors*, vol. 20, p. 6997, Jan. 2020.

[558] P. D. Byrnes and W. E. Higgins, "Construction of a multimodal CT-video chest model," in *Medical Imaging 2014: Image-Guided Procedures, Robotic Interventions, and Modeling*, vol. 9036, p. 903607, International Society for Optics and Photonics, Mar. 2014.

[559] X. Luó, M. Feuerstein, T. Kitasaka, H. Natori, H. Takabatake, Y. Hasegawa, and K. Mori, "On scale invariant features and sequential Monte Carlo sampling for bronchoscope tracking," in *Medical Imaging 2011: Visualization, Image-Guided Procedures, and Modeling*, vol. 7964, p. 79640Q, International Society for Optics and Photonics, Mar. 2011.

[560] X. Luó, M. Feuerstein, D. Deguchi, T. Kitasaka, H. Takabatake, and K. Mori, "Development and comparison of new hybrid motion tracking for bronchoscopic navigation," *Medical image analysis*, vol. 16, no. 3, pp. 577–596, 2012.

[561] I. Bricault, G. Ferretti, and P. Cinquin, "Registration of real and CT-derived virtual bronchoscopic images to assist transbronchial biopsy," *IEEE transactions on medical imaging*, vol. 17, no. 5, pp. 703–714, 1998.

[562] K. Mori, D. Deguchi, J.-i. Hasegawa, Y. Suenaga, J.-i. Toriwaki, H. Takabatake, and H. Natori, "A method for tracking the camera motion of real endoscope by epipolar geometry analysis and virtual endoscopy system," in *International Conference on Medical Image Computing and Computer-Assisted Intervention*, pp. 1–8, Springer, 2001.

[563] J. P. Helferty and W. E. Higgins, "Combined endoscopic video tracking and virtual 3D CT registration for surgical guidance," in *Proceedings. International Conference on Image Processing*, vol. 2, pp. II–II, IEEE, 2002.

[564] K. Mori, D. Deguchi, J. Sugiyama, Y. Suenaga, J.-i. Toriwaki, C. R. Maurer Jr, H. Takabatake, and H. Natori, "Tracking of a bronchoscope using epipolar geometry analysis and intensity-based image registration of real and virtual endoscopic images," *Medical Image Analysis*, vol. 6, no. 3, pp. 321–336, 2002.

[565] F. Deligianni, A. Chung, and G.-z. Yang, "Patient-specific bronchoscope simulation with pq-space-based 2D/3D registration," *Computer Aided Surgery*, vol. 9, pp. 215–226, Jan. 2004.

[566] J. Nagao, K. Mori, T. Enjouji, D. Deguchi, T. Kitasaka, Y. Suenaga, J.-i. Hasegawa, J.-i. Toriwaki, H. Takabatake, and H. Natori, "Fast and Accurate Bronchoscope Tracking Using Image Registration and Motion Prediction," in *Medical Image Computing and Computer-Assisted Intervention MICCAI 2004* (C. Barillot, D. R. Haynor, and P. Hellier, eds.), Lecture Notes in Computer Science, (Berlin, Heidelberg), pp. 551–558, Springer, 2004.

[567] R. Shinohara, K. Mori, D. Deguchi, T. Kitasaka, Y. Suenaga, H. Takabatake, M. Mori, and H. Natori, "Branch identification method for CT-guided bronchoscopy based on eigenspace image matching between real and virtual bronchoscopic images," in *Medical Imaging 2006: Physiology, Function, and Structure from Medical Images*, vol. 6143, p. 614314, International Society for Optics and Photonics, Mar. 2006.

[568] R. Khare, K.-C. Yu, and W. E. Higgins, “Improved navigation for image-guided bronchoscopy,” in *Medical Imaging 2009: Visualization, Image-Guided Procedures, and Modeling*, vol. 7261, p. 72612J, International Society for Optics and Photonics, Mar. 2009.

[569] X. Luo, T. Kitasaka, and K. Mori, “ManiSMC: A New Method Using Manifold Modeling and Sequential Monte Carlo Sampler for Boosting Navigated Bronchoscopy,” in *Medical Image Computing and Computer-Assisted Intervention MICCAI 2011* (G. Fichtinger, A. Martel, and T. Peters, eds.), Lecture Notes in Computer Science, (Berlin, Heidelberg), pp. 248–255, Springer, 2011.

[570] X. Luó, M. Feuerstein, T. Kitasaka, and K. Mori, “Robust bronchoscope motion tracking using sequential Monte Carlo methods in navigated bronchoscopy: dynamic phantom and patient validation,” *International Journal of Computer Assisted Radiology and Surgery*, vol. 7, pp. 371–387, May 2012.

[571] X. Luo and K. Mori, “A discriminative structural similarity measure and its application to video-volume registration for endoscope three-dimensional motion tracking,” *IEEE transactions on medical imaging*, vol. 33, no. 6, pp. 1248–1261, 2014.

[572] M. Shen, S. Giannarou, and G.-Z. Yang, “Robust camera localisation with depth reconstruction for bronchoscopic navigation,” *International Journal of Computer Assisted Radiology and Surgery*, vol. 10, pp. 801–813, June 2015.

[573] A. Esteban-Lansaque, C. Sánchez, A. Borràs, M. Diez-Ferrer, A. Rosell, and D. Gil, “Stable Anatomical Structure Tracking for Video-Bronchoscopy Navigation,” in *Clinical Image-Based Procedures. Translational Research in Medical Imaging* (R. Shekhar, S. Wesarg, M. Á. González Ballester, K. Drechsler, Y. Sato, M. Erdt, M. G. Linguraru, and C. Oyarzun Laura, eds.), Lecture Notes in Computer Science, (Cham), pp. 18–26, Springer International Publishing, 2016.

[574] M. Visentini-Scarzanella, T. Sugiura, T. Kaneko, and S. Koto, “Deep monocular 3D reconstruction for assisted navigation in bronchoscopy,” *International Journal of Computer Assisted Radiology and Surgery*, vol. 12, pp. 1089–1099, July 2017.

[575] M. Shen, S. Giannarou, P. L. Shah, and G.-Z. Yang, “BRANCH:Bifurcation Recognition for Airway Navigation based on struCtural cHaracteristics,” in *Medical Image Computing and Computer-Assisted Intervention MICCAI 2017* (M. Descoteaux, L. Maier-Hein, A. Franz, P. Jannin, D. L. Collins, and S. Duchesne, eds.), Lecture Notes in Computer Science, (Cham), pp. 182–189, Springer International Publishing, 2017.

[576] M. Shen, Y. Gu, N. Liu, and G.-Z. Yang, “Context-Aware Depth and Pose Estimation for Bronchoscopic Navigation,” *IEEE Robotics and Automation Letters*, vol. 4, pp. 732–739, Apr. 2019.

[577] C. Zhao, M. Shen, L. Sun, and G.-Z. Yang, “Generative Localization With Uncertainty Estimation Through Video-CT Data for Bronchoscopic Biopsy,” *IEEE Robotics and Automation Letters*, vol. 5, pp. 258–265, Jan. 2020.

[578] C. Wang, M. Oda, Y. Hayashi, B. Villard, T. Kitasaka, H. Takabatake, M. Mori, H. Honma, H. Natori, and K. Mori, “A visual SLAM-based bronchoscope tracking scheme for bronchoscopic navigation,” *International Journal of Computer Assisted Radiology and Surgery*, vol. 15, pp. 1619–1630, Oct. 2020.

[579] A. Banach, F. King, F. Masaki, H. Tsukada, and N. Hata, “Visually Navigated Bronchoscopy using Three Cycle-Consistent Generative Adversarial Network for Depth Estimation,” *Medical Image Analysis*, p. 102164, 2021.

[580] C. Sánchez, M. Diez-Ferrer, J. Bernal, F. J. Sánchez, A. Rosell, and D. Gil, “Navigation Path Retrieval from Videobronchoscopy Using Bronchial Branches,” in *Clinical Image-Based Procedures. Translational Research in Medical Imaging* (C. Oyarzun Laura, R. Shekhar, S. Wesarg, M. Á. González Ballester, K. Drechsler, Y. Sato, M. Erdt, and M. G. Linguraru, eds.), Lecture Notes in Computer Science, (Cham), pp. 62–70, Springer International Publishing, 2016.

[581] J. Sganga, D. Eng, C. Graetzel, and D. B. Camarillo, “Autonomous driving in the lung using deep learning for localization,” *arXiv preprint arXiv:1907.08136*, 2019.

[582] M. Turan, Y. Almalioglu, H. Gilbert, A. E. Sari, U. Soylu, and M. Sitti, “Endo-VMFuseNet: Deep Visual-Magnetic Sensor Fusion Approach for

Uncalibrated, Unsynchronized and Asymmetric Endoscopic Capsule Robot Localization Data," *arXiv:1709.06041 [cs]*, Sept. 2017.

[583] D. Recasens, J. Lamarca, J. M. Fácil, J. M. M. Montiel, and J. Civera, "Endo-Depth-and-Motion: Localization and Reconstruction in Endoscopic Videos using Depth Networks and Photometric Constraints," *arXiv preprint arXiv:2103.16525*, 2021.

[584] X. Liu, J. Berg, F. King, and N. Hata, "Computer vision-guided bronchoscopic navigation using dual CNN-generated depth images and ICP registration," in *Medical Imaging 2020: Image-Guided Procedures, Robotic Interventions, and Modeling*, vol. 11315, p. 113152C, International Society for Optics and Photonics, Mar. 2020.

[585] M. Schuster and K. Paliwal, "Bidirectional recurrent neural networks," *IEEE Transactions on Signal Processing*, vol. 45, pp. 2673–2681, Nov. 1997.

[586] D. Tran, H. Wang, L. Torresani, J. Ray, Y. LeCun, and M. Paluri, "A Closer Look at Spatiotemporal Convolutions for Action Recognition," *arXiv:1711.11248 [cs]*, Apr. 2018.

[587] J. Carreira and A. Zisserman, "Quo vadis, action recognition? a new model and the kinetics dataset," in *proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 6299–6308, 2017.

[588] M. Diez-Ferrer, D. Gil, E. Carreño, S. Padrones, S. Aso, V. Vicens, C. Noelia, R. L. Lisbona, C. Sanchez, and A. Borras, "Positive airway pressure-enhanced CT to improve virtual bronchoscopic navigation," *Chest*, vol. 150, no. 4, p. 1003A, 2016.

[589] S. A. Merritt, R. Khare, R. Bascom, and W. E. Higgins, "Interactive CT-Video Registration for the Continuous Guidance of Bronchoscopy," *IEEE Transactions on Medical Imaging*, vol. 32, pp. 1376–1396, Aug. 2013.

[590] G. Ning, Z. Zhang, C. Huang, X. Ren, H. Wang, C. Cai, and Z. He, "Spatially supervised recurrent convolutional neural networks for visual object tracking," in *2017 IEEE International Symposium on Circuits and Systems (ISCAS)*, pp. 1–4, May 2017.

[591] A. Ullah, J. Ahmad, K. Muhammad, M. Sajjad, and S. W. Baik, "Action Recognition in Video Sequences using Deep Bi-Directional LSTM With CNN Features," *IEEE Access*, vol. 6, pp. 1155–1166, 2018.

[592] T. Jin, Y. Li, and Z. Zhang, “Recurrent convolutional video captioning with global and local attention,” *Neurocomputing*, vol. 370, pp. 118–127, Dec. 2019.

[593] X. Shi, Z. Chen, H. Wang, D.-Y. Yeung, W.-k. Wong, and W.-c. Woo, “Convolutional LSTM Network: a machine learning approach for precipitation nowcasting,” in *Proceedings of the 28th International Conference on Neural Information Processing Systems - Volume 1*, NIPS’15, (Cambridge, MA, USA), pp. 802–810, MIT Press, Dec. 2015.

[594] S. Ji, C. Zhang, A. Xu, Y. Shi, and Y. Duan, “3D Convolutional Neural Networks for Crop Classification with Multi-Temporal Remote Sensing Images,” *Remote Sensing*, vol. 10, p. 75, Jan. 2018.

[595] S. Guo, Y. Lin, S. Li, Z. Chen, and H. Wan, “Deep Spatial–Temporal 3D Convolutional Neural Networks for Traffic Data Forecasting,” *IEEE Transactions on Intelligent Transportation Systems*, vol. 20, pp. 3913–3926, Oct. 2019.

[596] J. Sganga, D. Eng, C. Graetzel, and D. B. Camarillo, “Deep learning for localization in the lung,” *arXiv preprint arXiv:1903.10554*, 2019.

[597] Z. Liu, Y. Wang, K. Han, W. Zhang, S. Ma, and W. Gao, “Post-training quantization for vision transformer,” *Advances in Neural Information Processing Systems*, vol. 34, pp. 28092–28103, 2021.

[598] F. Lagunas, E. Charlaix, V. Sanh, and A. M. Rush, “Block pruning for faster transformers,” *arXiv preprint arXiv:2109.04838*, 2021.

[599] W. Wang, F. Wei, L. Dong, H. Bao, N. Yang, and M. Zhou, “Minilm: Deep self-attention distillation for task-agnostic compression of pre-trained transformers,” *Advances in Neural Information Processing Systems*, vol. 33, pp. 5776–5788, 2020.

[600] W. You and C. Wu, “Rsnn: A software/hardware co-optimized framework for sparse convolutional neural networks on fpgas,” *IEEE Access*, vol. 9, pp. 949–960, 2020.

[601] S.-E. Chang, Y. Li, M. Sun, R. Shi, H. K.-H. So, X. Qian, Y. Wang, and X. Lin, “Mix and match: A novel fpga-centric deep neural network quantization framework,” in *2021 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*, pp. 208–220, IEEE, 2021.