

Characterizing Face Recognition for Resource Efficient Deployment on Edge

A thesis submitted in partial fulfillment
of the requirements for the degree of

Master of Science
in
Computer Science & Engineering

by

Ayan Biswas
2019121009

ayan.biswas@research.iiit.ac.in

Advisor: Dr. Anoop M. Namboodiri



INTERNATIONAL INSTITUTE OF
INFORMATION TECHNOLOGY
HYDERABAD

International Institute of Information Technology Hyderabad
500 032, India

June 2026

Copyright © Ayan Biswas, 2026
All Rights Reserved

International Institute of Information Technology Hyderabad
Hyderabad, India

CERTIFICATE

This is to certify that work presented in this thesis proposal titled ***Characterizing Face Recognition for Resource Efficient Deployment on Edge*** by *Ayan Biswas* has been carried out under my supervision and is not submitted elsewhere for a degree.

Date

Advisor: Dr. Anoop M. Namboodiri

ACKNOWLEDGMENTS

I would like to express my gratitude to Dr Anoop, who has guided me through my research journey. He has given me both the freedom to work at my own pace and has been patient with me. Beyond that, I still remember his assuring presence when I first travelled abroad for a conference. He had kept my hotel information handy and asked me to reach out if any need arises. I've learnt a lot from him, and am grateful to have received this opportunity.

I could not have even begun this journey, let alone complete it, without the support of my parents. My mother, who has always had clear vision and foresight, has always guided me towards what is truly important. My father, who has been a shield, has absorbed all the harshness of life to make mine comfortable. Their hard work and perseverance over the years cannot be described in words, but they have shaped me into who I am today.

I would also like to thank Dr A.H. Abdul Hafez, who had guided me through my paper alongside Dr Anoop. His guidance had arrived at a time when I was fairly new to research. The first drafts of my writing were conversational and did not present clear technical arguments. From there, almost every single reviewer had mentioned that my paper was "very well written". This is largely due to Dr Hafez's review, critique and mentorship.

I extend my gratitude to friends who have been with me through this journey. Archit, Rishav, BVK and BRJ. From academics to assignments, to research and beyond, their presence has been a massive support and is extremely valued. BVK's passing last year has left me feeling extremely hollow; it has been the most tragic of events. I cannot imagine what college life would have been without them.

I'll also extend a note of gratitude to Souptik. While not directly involved with any academic work at IIT, he and Archit have been my closest friends, people whom I could reach out to and consult with for absolutely anything.

Finally, I'd like to extend my thanks to Mr Kishore Y V S K, whose prompt replies and helpful nature had made official work and processes much easier to navigate.

Abstract

Deployment of Face Recognition (FR) systems on edge devices has seen significant growth, driven by advances in hardware and neural architectures. However, tailoring State-of-the-art (SOTA) FR solutions to specific edge environments remains difficult and largely unexplored. Although benchmark data exists for some combinations of model, hardware, and frameworks, it does not scale to the evaluation of custom or modified architectures. Furthermore, theoretical metrics such as Floating-Point Operations (FLOPs) are often poor predictors of real-world performance, as edge inference throughput is frequently constrained by system-level factors including data movement, memory access, and pipeline overheads.

This limitation arises from a combinatorial design space spanning architectural variants, input resolutions, deployment frameworks, and hardware configurations, making exhaustive benchmarking infeasible. An analysis of recent FR literature performed in this thesis reveals that while substantial innovation is concentrated in loss functions, training strategies, and data augmentation, the underlying Convolutional Neural Network (CNN) backbone is routinely treated as a fixed abstraction. These backbones are typically inherited from large-scale image classification, where models are optimized to separate visually distinct categories, whereas FR requires learning discriminative embeddings that distinguish between highly similar identities. Consequently, the exploration of lightweight, task-specific architectures for edge deployment remains limited.

To address this gap, this thesis models the relationship between network architecture and inference throughput in an edge deployment setting, demonstrating that throughput follows a predictable structure when grouped by architectural families. Building on this modeling, this work proposes a data-efficient, architecture-aware framework to estimate the throughput of custom FR models from a sparse set of empirical measurements, with predictions that are explainable and exhibit errors small enough to be treated as standard on-device observational noise.

Further, Face Anti-Spoofing (FAS) pipelines are evaluated to quantify the trade-offs between inference throughput and error rates, highlighting deployment challenges arising from specialized operations

that do not map efficiently to standard edge-acceleration frameworks. In parallel, empirical experimentation on FR backbones is conducted to characterize the trade-off between discriminative power and inference throughput, and its impact on deployability on edge hardware, revealing non-linear scaling behavior and throughput saturation driven by data pipeline constraints.

Collectively, these findings address a key gap in FR deployment: the absence of a scalable mapping between architectural design and on-device inference performance for custom models. By establishing an explainable mapping from model architecture to throughput, the proposed framework enables systematic performance estimation without exhaustive benchmarking, supporting hardware-aware design and exploration of custom architecture for FR. To the best of the author’s knowledge, this is the first work to directly address this problem. The complete benchmarking suite and source code are publicly available at https://github.com/AyanBiswas19/Resource_Efficient_FR.

Contents

Chapter	Page
1 Introduction	1
1.1 Motivation	5
1.2 Contribution	6
1.3 Thesis Organisation	7
2 Background and Literature Review	8
2.1 Architectural Evolution Toward Edge-Efficient CNNs	8
2.2 Structuring Innovation in Face Recognition	11
2.3 Literature in the benchmarking of Edge AI	12
2.3.1 The MLPerf Inference Suite	12
2.3.2 AI Benchmark and Mobile Empirical Measurement	14
3 Architectural Characterization of Face Recognition Backbones for Edge Deployment	16
3.1 Deployment and Testing Environment	16
3.1.1 Nvidia Jetson Nano as a representative edge device	16
3.1.2 TensorRT and Deployment Tooling	18
3.1.3 Model Conversion Pipeline	18
3.2 Design of Experimentation	20
3.2.1 Metrics	20
3.2.2 Setup and Measurement Protocol	21
3.3 Throughput Estimation of Face Recognition Backbones	26
3.4 Quality of estimates: Noise & on-device fluctuation	29
4 Deployability of Face Anti-Spoofing Models	33
4.1 Introduction	33
4.2 Datasets and Metrics in FAS	35
4.2.1 Metrics	35
4.2.2 Datasets	36
4.3 Observed Edge Behavior and Trade-offs	37
5 Experimentation and Empirical Analysis	41
5.1 Throughput Scaling Across Input Resolution	41
5.2 Hardware Utilization Constraints	44
5.2.1 Efficacy of FP16 Precision Reduction	44
5.2.2 Data Pipeline and Memory Transfer Bottlenecks	46

5.3 Trade-off Between Discriminative Capacity and Edge Inference	47
6 Conclusion	49
Bibliography	51

List of Figures

Figure		Page
1.1	Methods employed to advance SOTA FR. Works involving models & architecture for FR are extremely rare. Most research is towards enhancing other components - Data, Training Strategy, & Loss Functions. Tailoring these solutions to edge devices or the use of backbones beyond common Object Recognition models is unexplored. Table 1.1 captures details of the same.	2
2.1	A basic building block of ResNet with the residual connection [7].	9
3.1	Deployment Environment: The NVIDIA Jetson Nano serves as the representative edge hardware. Experimental consistency is enforced via hardware-level CUDA stream synchronization, tensor preloading, and isolated warm-up passes. The measurement protocol is executed across standard PyTorch [38] and TensorRT [36] utilizing both 32-bit Floating Point Precision (FP32) and 16-bit Floating Point Precision (FP16) precision. Key measurement variables include the trial size (ts), batch size (b), start time (st), and end time (et). Details can be found in Protcols 1 and 2.	22
3.2	A plot of throughput against model scale (number of parameters). Each point corresponds to an empirically observed benchmark. Models belonging to the same architecture $\mathcal{A}$ [7] are represented with a circular marker of an identical color. The X-axis is plotted on a logarithmic scale. The naive global approach (top) demonstrates poor predictive performance. Conversely, the proposed architecture-aware estimator (bottom) achieves a highly accurate fit by grouping models categorically by $\mathcal{A}$, yielding an 84% reduction in Mean Absolute Error.	26
3.3	A comparative analysis of empirical performance on the local hardware unit against manufacturer-claimed statistics for the NVIDIA Jetson Nano, executed via the official benchmarking script [39]. The plot illustrates the Observed-to-Claimed throughput ratios for PyTorch (blue) and TensorRT (orange). The resulting observed-to-claimed ratio in Speedup (green) exhibits severe oscillations rather than a flat linear trend, providing clear visual evidence that intra-device variance is highly non-linear and explicitly dependent on model architecture.	31
4.1	Typical face spoofing attacks and the FAS integration pipeline. The FAS module acts as a critical gatekeeper (often in a serial scheme), meaning its inference latency directly impacts the total throughput of the authentication system. This thesis analyzes the edge-inference performance of CNNs designed to detect such presentation attacks. Adapted from Yu et al. [42].	34

5.1	Throughput ($\mathcal{T}$) (log scale) against resolution r on the X-axis, where $r = i \implies i \times i$ pixels. The results demonstrate substantial throughput degradation with increasing resolution, while throughput at lower resolutions approaches a performance ceiling rather than continuing to scale proportionally. Measured on an NVIDIA Jetson Nano using TensorRT at FP16 precision.	43
5.2	Number of equivalent 112×112 patches processed per second against r . This metric estimates the volume of input processed per unit time rather than parallel inputs (batch size). The results demonstrate improved effective hardware utilization at higher resolutions despite declining raw throughput. Measured on an NVIDIA Jetson Nano using TensorRT at FP16 precision.	44
5.3	Ratio of inference throughput achieved using FP16 relative to FP32 precision on TensorRT, plotted against absolute FP16 throughput. The data demonstrates a distinct architectural separation: heavier, compute-bound models (e.g., ResNets) exhibit substantial speed-up ratios approaching $2\times$ due to improved memory bandwidth utilization and parallel execution efficiency. Conversely, lightweight architectures demonstrate limited gains, indicating a shift toward memory or pipeline-bound operational regimes at high baseline throughputs.	45
5.4	Top: ImageNet Top-1 Accuracy against Throughput ($\mathcal{T}$). Bottom: Accuracy against Number of Parameters ($\mathcal{P}$). The results highlight the trade-off between architectural discriminative capacity and edge deployment efficiency.	47

List of Tables

Table		Page
1.1	As of the time of the related publication [P1], State-of-the-art Face Recognition research was primarily concentrated on loss functions, training methodology, and data-centric strategies. Model architecture itself was largely treated as an abstraction, with off the shelf object classification models used as backbones. ResNet variants dominate among these backbones, leaving custom or efficient architecture usage largely unexplored. . .	4
3.1	Train & Test errors of the proposed estimation method. Absolute errors may be measured in inferences/s at batch size = 1. Test models are custom models made from the same architecture. α refers to the width multiplier, as proposed in [25]. B and N denote the type of blocks used to make custom ResNet models - Basic Residual Block, Bottleneck Block.	28
3.2	Empirical evaluation of intra-device variance on the NVIDIA Jetson Nano. The table contrasts manufacturer-claimed throughput benchmarks [39] against actual observed performance on the local physical unit utilized in this study. This experiment was conducted on a fresh installation of NVIDIA JetPack using the official benchmarking script provided in the NVIDIA torch2trt repository [39]. The data demonstrates significant, non-linear performance scaling across different model architectures; for instance, AlexNet exhibits substantial degradation under PyTorch but yields unexpected performance gains under TensorRT.	30
4.1	Performance analysis of selected FAS models. * - Approximated throughput $\mathcal{T}$, as models did not execute fully on TensorRT. $\mathcal{P}$ - number of parameters (in millions). $\dagger$ - FP16, $\ddagger$ - FP32. HTER is computed under cross-dataset evaluation (trained on Replay-Attack [53], CASIA-MFSD [45], MSU-MFSD [54], tested on OULU-NPU [44]). ACER is reported on Protocol 1 of OULU-NPU. FeatherNet models report accuracy only on CASIA-SURF [46], and are therefore not directly comparable.	38
5.1	Maximum Resolutions achieved in experimentation, using TensorRT with FP16 Precision. Beyond these resolutions, random crashes occur, & the process is killed. RetinaFace [55] is paired with Mn25 = MobileNetV2 0.25 & Re50 = ResNet50 backbones.	41

Abbreviations

CNN	Convolutional Neural Network
CPU	Central Processing Unit
FAS	Face Anti-Spoofing
FLOP	Floating-Point Operation
FP16	16-bit Floating Point Precision
FP32	32-bit Floating Point Precision
FPS	Frames Per Second
FR	Face Recognition
GFLOPs	10^9 Floating-Point Operations
GPU	Graphics Processing Unit
I/O	Input/Output
MAC	Multiply-Accumulate
PyTorch	Open-source deep learning framework
SOTA	State-of-the-art
TensorRT	NVIDIA deep learning inference optimization and runtime SDK
torch2trt	PyTorch-to-TensorRT model conversion library

Symbols

$\mathcal{A}$	Family of models sharing similar architectural design patterns, operational blocks, and layer connectivity
$\mathcal{T}_{r,b}$	Inference throughput at input resolution $r \times r$ and batch size b
$\mathcal{T}$	$\mathcal{T}_{r=224,b=1}$ unless specified otherwise
ℓ	Inference latency of a single input, approximated as $\mathcal{T}_{r,b=1}^{-1}$
$\mathcal{F}$	Computational complexity measured in 10^9 Floating-Point Operations (GFLOPs) required for a single forward pass
$\mathcal{P}$	Total number of learnable parameters
r	Input resolution where $r = i \Rightarrow i \times i$ pixels
b	Batch size
m	Model instance
ts	Trial size used during throughput benchmarking
$\mathcal{L}$	Localized set of empirical benchmarks (<i>Landmarks</i>) used to fit throughput estimates for a specific physical device
$\mathcal{Z}_{\mathcal{L}}$	Architecture-aware throughput estimator fit on $\mathcal{L}$
ϵ	Observational noise arising from runtime and hardware variability
S	Ratio of FP16 throughput relative to FP32 throughput

List of Related Publications

- [P1] Ayan Biswas, Sai Amrit Patnaik, A. H. Abdul Hafez, and Anoop M. Namboodiri, “**Characterizing Face Recognition for Resource Efficient Deployment on Edge**”, in *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV) Workshops*, pp. 1312–1321, October 2023.

Chapter 1

Introduction

Recent years have seen rapid growth in edge computing and smart technologies, leading to substantial interest in performing previously difficult, compute-intensive deep learning tasks on resource-constrained endpoints such as edge devices. In particular, Face Recognition (FR) has emerged as a prominent edge workload [1–3], as it enables systems to respond intelligently to individual users in real time. However, transitioning these deep learning models to edge platforms introduces significant challenges, including availability of frameworks to run them, limited compute capability, memory bottlenecks, strict power budgets, and heterogeneous hardware characteristics.

Despite these hardware constraints, the future of authentication and access-control systems increasingly lies in Edge AI solutions, as processing sensitive sensor data locally provides a cost-effective, responsive, and privacy-preserving alternative to cloud transmission [4, 5]. Among various biometric modalities, face recognition remains one of the most widely adopted due to its non-intrusive nature and ease of integration. Consequently, there has been extensive interest and research work on Face Recognition and Face Anti-Spoofing (FAS) methodology. Yet, successfully tailoring and evaluating these specific biometric solutions for the resource-constrained environments outlined above remains a critical, underexplored gap. While this thesis focuses specifically on FR and FAS, the analytical framework presented is directly extensible to other computer vision tasks.

Before moving forward, it is necessary to define Face Recognition (FR) and Face Anti-Spoofing (FAS) within the context of this thesis:

Face Recognition (FR) refers to the task of automatically identifying or verifying an individual based on facial characteristics extracted from images or video streams. Modern FR systems typically rely on deep neural networks to learn discriminative facial embeddings that are robust to variations in pose, illumination, expression, and occlusion. These embeddings are subsequently compared against stored representations to perform identification or identity verification.

Face Anti-Spoofing (FAS) complements FR by attempting to detect any form of deception to the FR pipeline, for example, presentation attacks that attempt to deceive FR systems using artifacts such as printed photographs, replayed videos, or three-dimensional masks. Other than presentation, there are attacks such as deepfake-based impersonation. In practical systems, FAS is commonly integrated as a pre-filter or auxiliary module to ensure the security and trustworthiness of FR-based authentication.

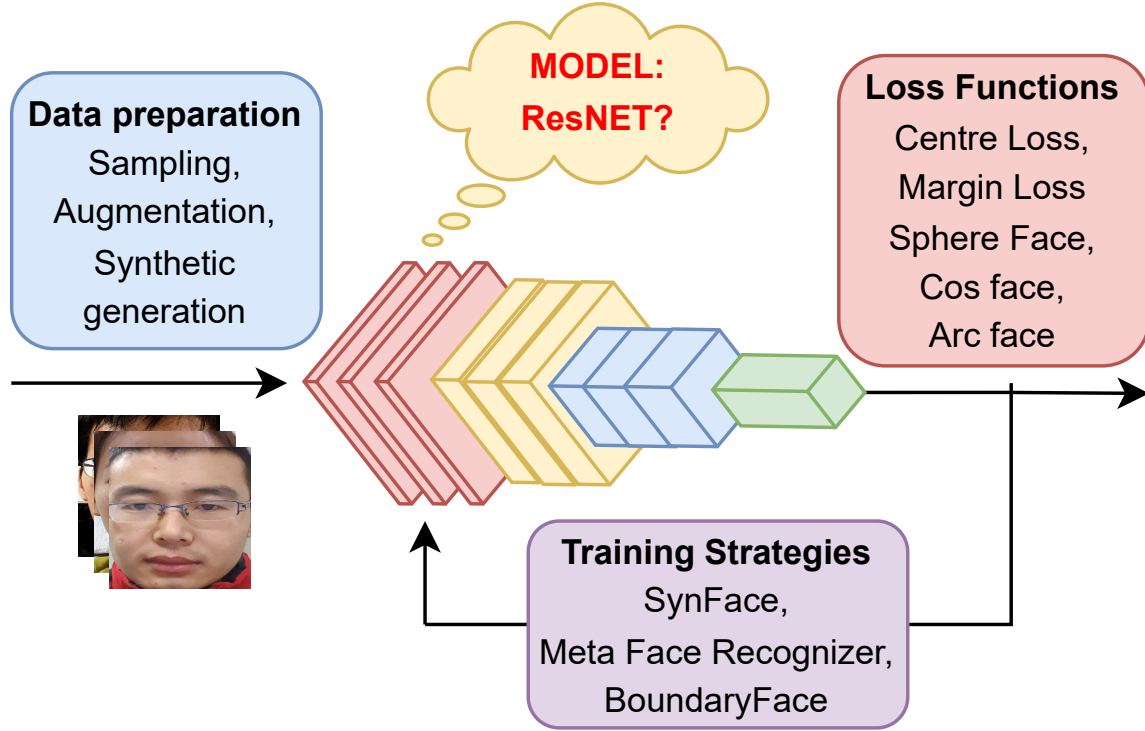


Figure 1.1 Methods employed to advance SOTA FR. Works involving models & architecture for FR are extremely rare. Most research is towards enhancing other components - Data, Training Strategy, & Loss Functions. Tailoring these solutions to edge devices or the use of backbones beyond common Object Recognition models is unexplored. Table 1.1 captures details of the same.

While recent advances have significantly improved FAS accuracy, many approaches introduce additional computational overhead, making efficient deployment alongside FR particularly challenging on edge platforms. As a result, understanding the resource and performance characteristics of both FR and FAS under strict edge constraints is essential for building secure and practical edge-based biometric systems. This thesis is not primarily concerned with the adversarial dynamics of attack generation and detection, but rather the inference pipeline of these models on edge devices. A detailed analysis of FAS models and their deployability is presented in Chapter 4.

As shown in Table 1.1, solutions in FR draw heavily from network architectures originally designed for macro-level image classification, such as those of the scale of ImageNet [6]. Although standard configurations like ResNet18 frequently serve as a "gold standard" backbone, the fundamental feature-extraction objectives of generic object classification and FR diverge significantly. Object classification focuses on separating visually distinct macroscopic categories characterized by massive inter-class variance (e.g., distinguishing a golden retriever from a commercial airplane). In contrast, FR requires highly fine-grained discrimination; the network must distinguish between millions of unique identities that share similar underlying morphological structure, resulting in significantly lower inter-class¹ variance.

¹Generally, a unique identity is considered to be a class in FR

This divergence in objective extends directly to inference, training and loss function. Standard object classification operates fundamentally as a closed-set problem, evaluating discrete probability distributions using top- k metrics over predefined training labels. Conversely, modern FR functions as an open-set metric learning problem. Rather than outputting categorical probabilities, the network must extract highly discriminative, continuous spatial embeddings, optimised to minimise intraclass and maximise interclass distances. For identification, a new embedding is queried against a database for the nearest neighbours by the same distance metric.

Further, most solutions default to standard model configurations, such as ResNet18 from the ResNet family [7]. While frameworks like MFR [8] represent a step in the desired direction by employing non-standard ResNet depth variants, the utilization of custom or task-specific models remains largely unexplored. Figure 1.1 illustrates the topology of SOTA FR literature². Most works do not concern themselves with architectural optimization; instead, the deep learning backbone is treated as an abstraction. While Face Anti-Spoofing (FAS) methodologies frequently employ hand-crafted, task-specific architectures (as analyzed later in Chapter 4), this structural stagnation is heavily pronounced within FR. This is likely due to prohibitive design costs and the unavailability of deployment benchmarks for modified architectures.

As a result, highly efficient topologies such as ShuffleNetV2 [9] and MnasNet [10] remain vastly unexploited in SOTA FR literature. This represents a critical gap: these models have been explicitly designed for resource-constrained environments and have demonstrated highly competitive performance compared to much heavier ResNet variants. Substituting these heavy backbones within existing methodologies presents a theoretical opportunity to achieve comparable benchmark results while consuming a fraction of the computational resources. However, theoretical efficiency must be validated empirically; it remains necessary to explicitly explore the on-device inference performance of these modified and resource-efficient architectures. By leveraging customized variations of these tried-and-tested backbones, researchers can theoretically tailor a model to consume the exact computational resources dictated by the task, accuracy constraints, and physical edge hardware.

Parallel to these architectural developments, there has been rapid growth in the performance and variety of chips available for AI or Neural Network tasks. Two projects deserve mention in this area: AI-Benchmark [11, 12], which approaches this problem specifically for mobile devices such as Android Smartphones. It has detailed work on chips produced by specific manufacturers including but not limited to Qualcomm, MediaTek, Samsung and HiSilicon. It has discussed supported libraries, frameworks, operations, and hardware acceleration. As of 2023, they have examples of 700+ phones on their site. Similarly, MLPerf [13] provides benchmarking tools and data for various Machine Learning tasks, with heavy industry collaboration across multiple categories of devices. They are further discussed in Section 2.3.

While suites like AI-Benchmark and MLPerf provide valuable insight into hardware and software support for edge deployment, their empirical data remains sparsely distributed across highly fragmented

²As of the time of the related publication [P1], August 2023.

Method	Venue	Primary Contribution	Backbones Used
ArcFace [14]	CVPR '19	Additive Angular Margin Loss	ResNet*50, 100
MFR [8]	CVPR '20	Domain Generalization via Meta-Learning	28-Layer ResNet [†]
MagFace [15]	CVPR '21	Magnitude-Aware Margin Loss	ResNet50, 100
SynFace [16]	ICCV '21	Synthetic-to-Real Data Augmentation	LResNet50E-IR
PatchNet [17]	CVPR '22	Patch-Wise Spoof Artifact Learning	ResNet*18, 34, 50, MobileNetV2, ShuffleNetV2 0.5
AdaFace [18]	CVPR '22	Quality-Adaptive Margin Loss	ResNet*18, 50, 100
ElasticFace [19]	CVPR '22	Stochastic Angular Margin	ResNet100
PartialFC [20]	CVPR '22	Sparse Softmax Training	ResNet50, 100, 200
BoundaryFace [21]	ECCV '22	Noisy Label Self-Correction	ResNet50
CoupleFace [22]	ECCV '22	Coupled Representation Distillation	ResNet, MobileNet
CFSM [23]	ECCV '22	Style-Guided Data Synthesis	ResNet50
Zhang <i>et al.</i> [24]	ECCV '22	Joint Architecture–Loss Co-Optimization	ResNet, MobileNet

* refers to the architectural modification popularized by ArcFace [14], namely the removal of the bottleneck structure. [†] indicates a width multiplier of 0.5.

Table 1.1 As of the time of the related publication [P1], State-of-the-art Face Recognition research was primarily concentrated on loss functions, training methodology, and data-centric strategies. Model architecture itself was largely treated as an abstraction, with off the shelf object classification models used as backbones. ResNet variants dominate among these backbones, leaving custom or efficient architecture usage largely unexplored.

framework, hardware and model combinations. Although standard architectures such as ResNet, MobileNet, DenseNet, and ShuffleNet [7, 9, 10, 25–27] are relatively well-represented, relying solely on static benchmarks remains insufficient, as even identical model-hardware pairings yield varying performance metrics depending on the underlying runtime, firmware, and even physical manufacturing variances across multiple units of the same device.

To ensure a highly focused analysis, it is necessary to explicitly define the boundaries of this research. Face detection can be formulated as a specialized instance of the broader object detection problem, employing similar network architectures, loss functions, and inference pipelines. While it may incorporate task-specific adaptations, its overall computational characteristics and deployment behavior on edge devices remain closely aligned with those of generic object detection systems. Consequently, given that these underlying properties have been extensively studied, face detection is not explored further within the scope of this work.

The primary objective of this research is to analyze the inference pipelines of FR and FAS models; the problem of model training on edge devices is outside the scope of this work. To this end, this thesis formalizes the assumptions, protocols, and metrics needed to describe and measure model behavior on the edge. However, certain systemic bottlenecks that are not directly related to deep learning algorithms still exert a significant impact on model inference. One primary example is the data pipeline feeding input to the model, which remains largely implementation-dependent. These pipeline bottlenecks are addressed empirically within this thesis (Chapter 5) to highlight their effect on overall throughput, alongside an analysis of how these systemic factors interact with different network architectures.

1.1 Motivation

The rapid growth of edge AI has led to widespread deployment of deep learning models on resource-constrained devices such as embedded systems and single-board computers. Among these, Face Recognition (FR) systems are particularly prominent due to their use in security, authentication, and surveillance applications, where real-time performance is often a critical requirement.

Despite significant advances in model accuracy, modern FR innovation uses off the shelf backbones built for ImageNet [6]; architectures that are typically developed and evaluated in high-resource environments and for object classification rather than embedding generation. When deployed on edge devices, these models must operate under strict constraints on compute, memory, and power consumption. As a result, their inference behavior can differ significantly from expectations based on desktop or server-class evaluations. Furthermore, standard efficiency proxies such as Floating-Point Operations (FLOPs) or Multiply-Accumulates (MACs) often provide an incomplete or misleading characterization of real-world performance on edge devices, and do not correlate well to throughput in isolation [9] [10].

A key challenge in this setting is that, while benchmark data is often available for specific combinations of FR models, hardware platforms, and deployment frameworks, such results are inherently limited in scope. The design space spans multiple dimensions, including model architecture, input res-

olution, hardware characteristics, and software backends, making exhaustive benchmarking across all configurations infeasible. This limitation is particularly pronounced for custom or modified models, for which no prior performance data exists.

Consequently, there is no standard way to estimate the inference performance of a new model without exhaustive measurement, making design decisions difficult and inefficient. This highlights the need to move beyond isolated measurements toward a systematic understanding of how architectural design choices influence inference performance on edge hardware. These challenges extend to auxiliary components in FR pipelines, such as Face Anti-Spoofing (FAS), which introduce additional constraints and architectural variations.

These challenges motivate the need for a systematic and scalable methodology to characterize and predict the inference performance of FR and related models on edge devices. Such an approach would enable informed decision-making during model design and deployment, reducing reliance on ad hoc benchmarking and facilitating more efficient use of limited hardware resources.

1.2 Contribution

This thesis makes the following primary contributions toward the deployment of Face Recognition (FR) and Face Anti-Spoofing (FAS) models on resource-constrained edge devices:

1. **Structural characterization of State-of-the-art (SOTA) Face Recognition (FR) models and their inference behavior on edge devices.** This work analyzes solutions in State-of-the-art (SOTA) FR literature and demonstrates that they can be effectively grouped into a small set of backbone families (e.g., ResNet-based models), enabling a structured representation of the model design space. It further establishes an empirical mapping between these architectural families and their corresponding inference performance under real-world edge deployment constraints, within a systematically defined methodology for analyzing model inference on edge devices.
2. **An explainable, architecture-aware framework for throughput estimation of custom models from limited data.** Building on the identified architectural grouping, this thesis proposes a data-efficient methodology for predicting inference throughput of Convolutional Neural Network (CNN) models from a sparse set of empirical measurements, enabling scalable performance estimation without exhaustive benchmarking. Although developed in the context of SOTA FR models, the approach is generalizable to a broader class of architectures and deployment settings.
3. **Analysis of the trade-off between inference time and error rates for FAS models under edge constraints.** This thesis evaluates FAS models by examining the trade-off between throughput and error rates in resource-constrained environments, while identifying deployment challenges arising from specialized operations that do not map efficiently to optimized runtimes such as TensorRT. Complementary empirical experimentation on FR backbones further characterizes the

trade-off between discriminative power and inference throughput, alongside the impact of system-level bottlenecks in practical edge deployment.

1.3 Thesis Organisation

- **Chapter 2** provides background on the evolution of Convolutional Neural Network (CNN) architecture, taxonomy of innovation in the field of Face Recognition (FR), and work done in benchmarking of Edge AI
- **Chapter 3** provides an analysis of throughput of backbone models used in State-of-the-art (SOTA) FR. It further proposes a method to estimate the throughput of the same, with high accuracy, limited data and in an explainable fashion.
- **Chapter 4** provides an analysis of throughput of the models used in Face Anti-Spoofing (FAS), their tradeoff with error-rates along and challenges in deploying them to the edge.
- **Chapter 5** captures the experimentation and empirical analysis performed as a part of this work, presenting insights on bottlenecks, deployment strategies, tradeoff in resolution, precision and discriminative power to throughput.
- **Chapter 6** captures conclusion and provides ideation for future work.

Chapter 2

Background and Literature Review

This chapter contextualizes the thesis at the intersection of hardware-aware CNN design, Face Recognition (FR) methodologies, and edge AI benchmarking. Section 2.1 traces the structural evolution of Convolutional Neural Networks (CNNs) from compute-intensive baselines to hardware-aware topologies, highlighting the limitations of theoretical efficiency metrics such as FLOPs and parameter counts in accurately reflecting real-world inference performance on edge hardware. Section 2.2 analyzes contemporary FR literature, showing that despite significant algorithmic advancements, the underlying inference backbone is largely treated as an abstraction, limiting the exploration of throughput–accuracy trade-offs for edge deployment. Finally, Section 2.3 analyzes established edge AI benchmarking suites, emphasizing their reliance on fixed model sets and costly empirical measurement, and their limited capacity to predictively evaluate custom architectures. Collectively, this review establishes both the absence of, and the need for, a practical methodology to predict the edge inference performance of custom neural architectures without exhaustive deployment.

2.1 Architectural Evolution Toward Edge-Efficient CNNs

In the early development of deep-learning based modern computer vision, the foundational era was catalyzed by AlexNet (2012) [28], which significantly outperformed prior hand-engineered feature pipelines by achieving a 16.4% top-5 error rate on ImageNet [6]. It demonstrated the viability of large-scale Convolutional Neural Networks (CNNs), utilizing a roughly 60-million-parameter architecture with ReLU activations and dropout for regularization, enabled by GPU-accelerated training. Subsequent iterations pursued lower error rates by increasing network depth and architectural complexity. VGGNet (2014) [29] extended depth up to 19 layers by replacing large spatial filters with stacked 3×3 convolutions; however, this scaling pushed the model to 144 million parameters, with roughly 90% concentrated in dense fully connected layers. Conversely, GoogLeNet (2015)¹ [30] attempted to limit parameter growth using a 22-layer topology with Inception blocks, leveraging 1×1 bottleneck convolutions to compress channel dimensions before parallel filtering. It also introduced parallel, multi-scale convolutional paths with dimensionality reduction. Beyond feature extraction, this was an early attempt

¹Published 2015, was submitted to ILSVRC in 2014

at increasing computational efficiency, albeit on server-class systems. However, this introduced non-uniform execution paths that are difficult to deploy on edge devices. Ultimately, whether characterized by large parameter counts or heterogeneous execution branches, these pioneering models prioritized classification accuracy over inference efficiency, which remains suboptimal for resource-constrained edge deployment.

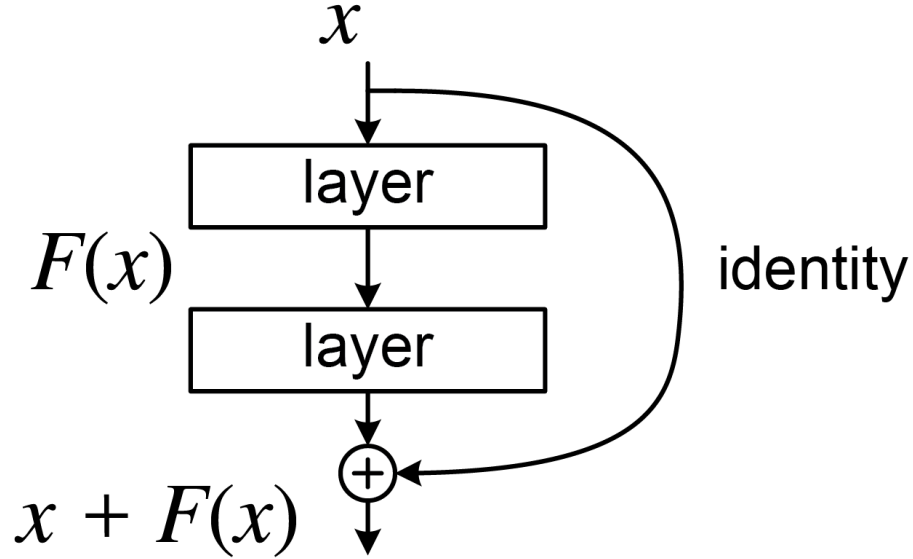


Figure 2.1 A basic building block of ResNet with the residual connection [7].

As networks grew increasingly deep in pursuit of lower error rates, optimization was hindered by the degradation problem, where accuracy saturated and subsequently degraded with depth. ResNet (2016)² [7] mitigated this issue by introducing residual connections, a structural innovation that enabled the learning of residual functions via identity mapping ($f(x) + x$) (Fig 2.1). This maintained a stable gradient flow, allowing networks to scale past 100 layers without incurring optimization instability. DenseNet (2017) [27] subsequently iterated on this principle of feature reuse by replacing additive skip connections with channel-wise concatenation ($x_i = H([x_0, x_1, \dots, x_{i-1}])$), achieving comparable accuracy with fewer absolute parameters. However, while DenseNet offered high theoretical parameter efficiency, its repeated feature concatenation had caused significant memory bandwidth utilization and increased data traffic during inference. Consequently, ResNet’s balanced structural simplicity established it as a widely adopted, often “gold” standard for deep feature extraction. This architectural baseline is particularly evident in modern Face Recognition (FR) pipelines, where variations of ResNet are commonly utilized as a fixed backbone architecture, directly setting the stage for the edge-deployment bottlenecks evaluated in this thesis.

²preprint was available in 2015

As deep learning expanded into mobile and embedded domains, architectural design branched to accommodate strict computational and power constraints. MobileNetV1 (2017) [25], one of the iconic early edge efficient models, had replaced standard convolutional filters with depthwise separable convolutions. This structural factorization isolates spatial filtering from channel mixing, analytically reducing the computational cost of a layer from $O(D_K^2 \cdot M \cdot N \cdot D_F^2)$ to $O(D_K^2 \cdot M \cdot D_F^2 + M \cdot N \cdot D_F^2)$, where M and N represent input and output channels, and D_K and D_F denote kernel and feature map dimensions. This formulation had provided explicit width and resolution multipliers, allowing developers to systematically trade classification accuracy for reduced Multiply-Accumulates (MACs). This lineage was subsequently established as a primary baseline for edge deployment by MobileNetV2 (2018) [26], which introduced inverted residuals and linear bottlenecks a topology engineered to preserve representational capacity while enabling memory-efficient inference. Concurrently, ShuffleNet (2018) [31] further optimized the paradigm by directly addressing the computational bottleneck inherent in MobileNetV1’s dense 1×1 pointwise convolutions. By introducing pointwise group convolutions alongside a channel shuffle operation, ShuffleNet reduced the MAC footprint of channel mixing while maintaining cross-group information flow. Collectively, these architectures established a highly tunable approach to network scaling, demonstrating that deep learning topologies could be explicitly engineered to operate within tight resource budgets.

Despite this structural shift toward parameter efficiency, the proxy metrics traditionally used to evaluate these architectures proved insufficient for rigorous edge deployment. Theoretical metrics such as Floating-Point Operations (FLOPs) and absolute parameter counts often correlate inconsistently with real-world execution speed across diverse architectures, as they do not explicitly model memory access costs, operator fusion opportunities, or hardware-specific cache behavior among a long list of factors. MnasNet (2019) [10] highlighted this divergence by empirically demonstrating that models with similar FLOP counts can exhibit significantly different inference latencies when executed on physical edge devices. To overcome this abstraction gap, MnasNet deprioritized theoretical proxies, directly integrating physical hardware-in-the-loop profiling into its neural architecture search (NAS) reward function. However, while executing models on target hardware yields high-fidelity latency measurements, it introduces a severe evaluation bottleneck. Exhaustive empirical profiling is computationally prohibitive and scales poorly across the highly fragmented ecosystem of edge accelerators and mobile runtimes. Hence, optimizing feature extraction backbones used in FR, such as the established ResNet [7] models, for edge deployment, requires a methodology that bridges the gap between theoretical efficiency and physical execution without relying on exhaustive hardware-in-the-loop testing. This is applicable for tailoring efficient architectures to FR as well and defines the latency estimation challenge that is addressed in this thesis.

2.2 Structuring Innovation in Face Recognition

Following ArcFace [14], the primary driver of State-of-the-art (SOTA) accuracy became the manipulation of margin-based loss functions. For instance, to prevent models from overfitting to low-quality or unidentifiable faces, AdaFace [18] and MagFace [15] introduced dynamic, quality-aware loss formulations. Similarly, ElasticFace [19] introduced randomized margin penalties to allow for flexible face clustering. However, while these mathematical innovations vastly improved accuracy, they universally treated the deep learning backbone as a fixed variable, relying almost exclusively on computationally massive ResNet-50 and ResNet-100 architectures to validate their claims.

This architectural rigidity extends well beyond margin-based learning into broader data and training strategy innovations. For instance, even when researchers explicitly optimize for computational efficiency, the focus remains fundamentally disconnected from edge inference. PartialFC [20] addresses a massive Graphics Processing Unit (GPU) memory bottleneck by approximating the softmax denominator using only a fraction of negative class centers. However, this strictly optimizes distributed *training* efficiency across server clusters, utilizing standard ResNets. Consequently, while training bottlenecks are addressed, the resulting inference architecture remains computationally heavy and unoptimized for edge environments.

Similarly, methodologies designed to overcome dataset limitations universally treat the underlying inference architecture as an abstraction. To address data privacy and collection bottlenecks, SynFace [16] utilizes synthetic/generated faces in augmenting training data, explicitly introducing mixed-data strategies (combining synthetic and real images) to bridge domain gaps. To handle unconstrained real-world environments, CFSM [23] extracts specific styles (e.g., varying poses and illuminations) to augment constrained datasets, while BoundaryFace [21] detects and self-corrects mislabeled noisy data during the training loop. Even frameworks addressing demographic domain generalization via meta-learning, such as MFR [8], default to standard ResNet topologies to isolate their variables. In every instance, the network structure is held constant, leaving the optimization of the deep learning backbone for constrained deployment entirely unexplored.

While there are rare exceptions in recent ³ SOTA literature that incorporate resource-efficient networks, their scope remains largely disconnected from physical edge deployment. For example, CoupleFace [22] utilizes edge constraints to motivate its knowledge distillation framework, successfully training a lightweight MobileNet student. Similarly, in adjacent pipeline stages such as Face Anti-Spoofing, frameworks like PatchNet [17] explicitly test lightweight Convolutional Neural Networks (CNNs) (e.g., MobileNet and ShuffleNet variants). By actively evaluating lightweight topologies, these works represent a highly relevant step toward resource-efficient deployment. However, because their primary objectives focus on distillation and classification rather than hardware benchmarking, their efficiency evaluations naturally rely on theoretical metrics such as model footprint, parameter counts, and Floating-Point Operations (FLOPs). While theoretically sound, these metrics historically correlate

³At of the time of the related work [P1]

poorly with empirical inference throughput on target hardware, leaving a critical gap for practical edge deployment.

Conversely, Zhang *et al.* [24] provide a crucial theoretical foundation by demonstrating that architecture, data augmentations, and loss functions are tightly coupled. They argue that SOTA performance requires a unified search across all three aspects, rather than single-aspect optimization, and actively evaluate MobileNet to prove this claim. While their comprehensive search optimizes strictly for dataset accuracy rather than structural execution speed, their core finding is in line with the premise of this thesis: optimizing across one of the dimensions alone while keeping the architecture constant “greatly limits the performance and biases algorithmic design”. Developers require the ability to rapidly iterate on network topologies to reduce the search space along the throughput dimension for edge deployment. Currently, this iterative process is severely bottlenecked by the prohibitive setup costs of empirical hardware profiling.

Ultimately, the literature reveals a massive, untapped opportunity for resource-constrained Face Recognition (FR) deployment. Because the community has historically treated the neural network backbone as a static control variable, the specific throughput-to-accuracy trade-offs of lightweight topologies remain unquantified for edge pipelines. To the best of the author’s knowledge, this thesis represents the first attempt to systematically map and predict the empirical edge throughput of diverse neural architectures specifically within the context of State-of-the-art Face Recognition. To realize this untapped potential and allow developers to rapidly explore optimal networks, Chapter 3 proposes a lightweight framework to predict custom model throughput on edge hardware, bypassing the prohibitive setup costs of exhaustive empirical deployment.

2.3 Literature in the benchmarking of Edge AI

This section reviews two commonly used benchmarking frameworks relevant to this work: MLPerf Inference and AI Benchmark. These frameworks are representative of established benchmarking efforts in the literature, and each is discussed in turn, outlining its evaluation setup and the scope of performance measurement.

2.3.1 The MLPerf Inference Suite

The drive toward standardized, objective evaluation of machine learning hardware led to the formation of MLCommons, an open engineering consortium. Its most prominent project is MLPerf, an industry-standard benchmarking suite. While MLPerf maintains separate divisions for both model training and inference, this work focuses strictly on the inference suite. MLPerf Inference evaluates system performance primarily latency and throughput, only after a device successfully passes a strict baseline accuracy threshold.

MLPerf is explicitly designed to enable standardized, comparable evaluation across diverse hardware platforms through fixed workloads and strict benchmarking protocols. To account for the vast disparities

in deployment environments, the MLPerf Inference ecosystem is built upon three foundational architectures that segment the hardware landscape based on power availability, operating systems, and user interaction scenarios:

- **The Datacenter and Edge Foundation:** The original MLPerf Inference architecture [13] established the benchmark for wall-powered systems running standard operating systems (e.g., Linux). It introduced discrete measurement scenarios, defining how local servers and edge accelerators process data.
- **The Mobile Boundary:** A subsequent suite [32] was developed specifically for consumer devices (smartphones and tablets) operating within strict 1W to 5W thermal limits. It benchmarks single-stream tasks running on consumer operating systems like Android and iOS.
- **The Tiny Boundary:** The most constrained suite [33] targets bare-metal microcontrollers (e.g., ARM Cortex-M microprocessors or Arduino boards) operating under a single milliWatt. It evaluates simple sensor tasks using localized software completely devoid of an overarching operating system. Crucially, despite its small physical form factor, hardware like the NVIDIA Jetson Nano [34] operates at a significantly higher power tier and does not fall under this Tiny classification.

In the context of this thesis, an “edge device” specifically refers to the entry-level, Graphics Processing Unit (GPU)-accelerated edge tier such as the NVIDIA Jetson Nano operating within a constrained power envelope, as formally established in Section 3.1.1.

These devices act as local bridges between sensory data collection and the wider network. In contrast, the MLPerf Edge division adopts a significantly broader classification. It evaluates everything from these constrained gateway devices up to ruggedized, 300W+ industrial edge servers utilizing the exact same criteria and measurement scenarios (such as single-stream and multi-stream processing).

Despite its strengths as a standardized benchmarking suite, MLPerf presents critical limitations for deploying specialized Face Recognition (FR) models. First, to maintain comparability across its broad hardware spectrum, MLPerf Edge relies on a static, shared set of standard reference models, such as ResNet-50. While a gateway device like the Jetson Nano can be highly capable of running ResNet-50 achieving approximately 30 inferences per second⁴, edge deployments frequently rely on highly customized or modified lightweight architectures rather than these standard models.

Second, because MLPerf evaluates performance on a fixed set of standardized reference workloads, performance metrics for custom FR architectures are inherently unavailable. Generating comprehensive, empirical benchmarks for every new or modified model architecture on edge hardware is prohibitively time consuming. Furthermore, falling back on theoretical complexity metrics such as 10^9 Floating-Point Operations (GFLOPs) is inadequate, as they fail to account for critical systems-level bottlenecks [9, 10]. Relying solely on generic MLPerf metrics or theoretical operation counts is therefore insufficient for

⁴with TensorRT, at 16-bit Floating Point Precision (FP16), under the experimental setup of this thesis

optimized, hardware-aware deployment. Moreover, while MLPerf evaluates end-to-end system performance for standardized workloads, it does not provide a controlled mechanism for isolating inference characteristics of custom architectures under consistent conditions. To address this gap, Chapter 3 proposes methodology to approximate the throughput of custom models, characterizing performance on edge devices using minimal data points.

2.3.2 AI Benchmark and Mobile Empirical Measurement

While MLPerf provides a broad industry standard across hardware tiers, the AI Benchmark suite [11, 12] emerged from the academic sector (ETH Zurich) to address the rapid evolution of mobile System-on-Chips (SoCs). Initially launched as a consumer-facing Android application, AI Benchmark serves as a premier empirical measurement tool designed to produce comparative global rankings for smartphones. It evaluates hardware by deploying a fixed taxonomy of standard computer vision architectures (e.g., MobileNet, Inception) through vendor-specific mobile APIs, such as Android’s Neural Networks API (NNAPI) and proprietary drivers from manufacturers including Qualcomm, MediaTek, and HiSilicon.

While highly effective for consumer ranking, AI Benchmark is fundamentally optimized for the mobile ecosystem, creating a significant disconnect when applied to localized gateway edge deployments. Devices like the NVIDIA Jetson Nano operate on streamlined, open-ecosystem Linux distributions, avoiding the heavy software overhead and Java Native Interface (JNI) layers inherent to consumer Android operating systems.

Furthermore, when AI Benchmark extends its evaluation to desktop or dedicated NVIDIA hardware, it relies on raw TensorFlow executing via CUDA and cuDNN [12]. This baseline fails to reflect actual edge deployment standards. In production environments, inference accelerators like TensorRT are mandatory; they perform critical graph optimization and precision calibration (quantizing 32-bit Floating Point Precision (FP32) models to 16-bit Floating Point Precision (FP16)), drastically reducing memory bandwidth and latency. As observed in the experiments conducted in this thesis, executing ResNet-50 directly in PyTorch yields approximately 9 inferences per second, whereas deployment via TensorRT at FP16 achieves approximately 34 inferences per second (Table 3.2). This transformation, detailed in Section 3.1.3, is automated and architecture-agnostic, requiring no manual intervention. Consequently, reliance on general purpose frameworks fails to capture the device’s true performance, necessitating the use of device-specific deployment stacks to establish representative baselines.

Beyond these environmental disparities, the core limitation of AI Benchmark lies in its strict reliance on empirical measurement. The authors themselves acknowledge the exceptionally high setup costs, severe library fragmentation, and frequent execution failures encountered when attempting to deploy models across diverse hardware [12]. Because AI Benchmark is constrained to measuring a predefined “zoo” of standard models, it is inherently rigid.

For optimizing edge Face Recognition (FR) deployment, it is necessary to iterate on custom architecture, modify layers, or build multi-stage cascading pipelines. AI Benchmark cannot accommodate these dynamic alterations; to determine the throughput of a modified model, a developer must endure the

high setup costs of physically deploying and empirically measuring the new architecture on the target hardware.

While AI Benchmark excels as an observational tool to track the evolution of mobile consumer hardware, but it is not a predictive diagnostic tool. While a complete FR pipeline contains multiple stages, the deep learning backbone remains the primary computational bottleneck. This confirms a critical gap in the literature: there exists the a lack of standardised methodology to predict the performance of custom models without enduring exhaustive empirical deployment.

To address this gap, Chapter 3 proposes a methodology to approximate throughput of custom backbones, on edge devices using minimal, pre-calculated data points. Chapter 4 examines challenges in deploying Face Anti-Spoofing (FAS) models to edge and tradeoff with throughput and error rates. Similar to the observations of AI Benchmark, FAS models use custom operations which run into multiple issues in the process of conversion to an edge deployment runtime - TensorRT. Further, Chapter 5 expands upon FR backbones by empirically examining the broader systemic challenges of edge deployment, including the impact of input resolution, data pipeline bottlenecks and trade off with inference throughput with discriminative power.

Chapter 3

Architectural Characterization of Face Recognition Backbones for Edge Deployment

Addressing the predictive evaluation gap established in the preceding chapters, this chapter presents the empirical methodology and analytical framework required to benchmark and predict Convolutional Neural Network (CNN) inference performance on constrained edge hardware. Section 3.1 formalizes the physical and software testing environment, establishing the NVIDIA Jetson Nano and TensorRT compilation pipeline as the representative deployment baseline. Section 3.2 details the design of experimentation, defining the core architectural metrics and the hardware-aware protocols necessary for reproducible profiling. Building upon this foundation, Section 3.3 addresses the evaluation bottleneck by formulating an architecture-aware estimator, utilizing a sparse set of localized empirical measurements to map architectural characteristics to expected throughput. Finally, Section 3.4 validates the robustness of these estimates under intra-device variance and relative to generalized manufacturer benchmarks. Because Face Anti-Spoofing (FAS) models rely on highly specialized, task-specific topologies where backbone structure alone cannot reliably predict performance, they are excluded from this generalized estimator and analyzed separately in Chapter 4.

3.1 Deployment and Testing Environment

This section describes the deployment and testing environment used in this work. It outlines the choice of hardware platform - Jetson Nano, treated as representative edge device, the software stack used for optimized inference, and the model conversion pipeline employed for deployment. These elements collectively define the experimental environment for all subsequent measurements.

3.1.1 Nvidia Jetson Nano as a representative edge device

Edge devices encompass a broad spectrum of hardware, ranging from heavily constrained Central Processing Unit (CPU)-only IoT nodes to premium, dedicated neural processing units (NPUs). The devices considered in this work belong specifically to the entry-level, GPU-accelerated edge tier. These systems are characterized by limited computational boundaries constrained storage, shared memory, and

lower power envelopes that fall significantly below those of server-class architectures, yet possess the parallel processing capabilities required for computer vision tasks. Within this paradigm, the NVIDIA Jetson Nano [34] is selected as the primary representative hardware platform for this thesis.

From an architectural perspective, the Jetson Nano integrates a quad-core ARM Cortex-A57 CPU, 4 GB of LPDDR4 memory, and a CUDA-capable Graphics Processing Unit (GPU) with 128 cores based on the NVIDIA Maxwell architecture [35]. This specific configuration provides a practical baseline for evaluating Face Recognition (FR) workloads. Its standard CUDA support enables the execution of a broad spectrum of modern Convolutional Neural Network (CNN) architectures, provided they fit within the memory envelope and are paired with appropriate optimization and compilation frameworks such as TensorRT [36]. Concurrently, its strict 4 GB shared memory limit and 5–10 W power envelope expose the model to dynamic edge constraints such as runtime memory fragmentation, bandwidth bottlenecks, and thermal throttling. These constraints are particularly severe in real-world deployments, where input/output operations, video encoding, and data pipeline overhead must compete with model inference for a fraction of the same constrained compute budget. While introduced at an accessible official price point of USD 149, its practical acquisition cost is generally much higher; nonetheless, its specifications firmly anchor it as an entry-level testbed.

Establishing this exact hardware baseline is necessary, as prior literature often evaluates edge performance under substantially different conditions. For instance, prior work by Liu *et al.* [37] provided valuable characterizations of deep learning workloads using hardware-centric metrics such as CPU/GPU utilization and power consumption. However, their study utilized the Jetson TX2 a significantly more powerful, costly, and now-discontinued platform and centered its analytical focus on characterizing model *training* workloads on mobile devices.

This thesis diverges from such prior works by defining the following methodological scope boundaries:

- **Inference-Exclusive Scope:** The analysis is restricted entirely to model inference, as training and inference present fundamentally distinct compute bottlenecks on constrained devices.
- **Domain Specificity:** The evaluation is explicitly tailored to the operational requirements of FR and Face Anti-Spoofing (FAS) pipelines, differentiating it from generalized object detection benchmarks.
- **Architecture-Centric Evaluation:** Hardware is analyzed through the lens of mapping models of an architecture to their inference throughput on a specific edge device, rather than profiling the hardware’s baseline limits through generic workloads.

Establishing a reproducible hardware baseline for model inference is a necessary methodological step. However, hardware constraints alone are insufficient to characterize performance; maximizing inference throughput within these limits depends on an optimized deployment software stack, such as TensorRT. In addition, a well-defined test environment and evaluation protocols are required to ensure reproducible results. The following sections detail these components.

3.1.2 TensorRT and Deployment Tooling

While training of neural networks is generally done in controlled, resource-abundant environments, deploying the same is an entirely different problem space, especially when it comes to edge devices. In a standard NVIDIA Jetson-based deployment, the runtime pipeline typically involves capturing the video stream from a camera, passing it to the TensorRT model for Input/Output (I/O) and CUDA memory handling, and rendering the output. To maximize the limited compute budget of the Jetson Nano and utilize the software stack used for actual deployment, this thesis uses the TensorRT [36] SDK for its experimentation, while PyTorch [38] is used as a baseline. TensorRT ingests a trained model and builds an optimized inference engine for the specific target device, accounting for the exact GPU architecture, CUDA/cuDNN versions, and installed drivers. Typically, this is performed per target hardware configuration, with the TensorRT compilation happening as a one-time computation. This work explores TensorRT as a tool to maximize the performance of Convolutional Neural Network (CNN) models on edge devices rather than providing an end-to-end analysis of the tooling it offers for production deployment. At the same time, this thesis intentionally restricts conversions to be automated and generalizable, explicitly avoiding architecture-specific manual tuning or hints. While the TensorRT compilation process transforms the computational graph via operator fusion and kernel substitution, it does not alter the high-level functional mapping or user-defined structure of the model. All underlying optimizations, such as memory allocation and batching operations, are abstracted by the SDK and treated as opaque within the constraints of this automated pipeline. The conversion process is further detailed below in Section 3.1.3.

The primary objective of TensorRT compilation is the reduction of inference latency and the maximization of throughput, yielding secondary benefits in reducing the operational memory footprint. While a formal evaluation of numerical equivalence falls outside the primary experimental scope of this thesis, reduced-precision inference introduces minor numerical variation due to rounding effects. During throughput benchmarking with randomized inputs, only small activation-level differences between 32-bit Floating Point Precision (FP32) and 16-bit Floating Point Precision (FP16) executions were observed, consistent with expected reduced-precision behavior. In practice, FP16 deployment typically does not require additional calibration or post-conversion tuning [39]. This demonstrates negligible numerical deviation within this experimental setup, confirming that FP16 quantization can be aggressively employed to maximize edge throughput under the evaluated conditions.

3.1.3 Model Conversion Pipeline

While there are multiple ways to convert a model to TensorRT, the most popular approach utilizes an intermediate representation known as the Open Neural Network Exchange (ONNX). ONNX defines a portable model format and a directed acyclic computational graph. Within this graph, nodes represent operations (e.g., ReLU or Convolution), edges represent the tensors flowing between nodes, and initializers hold the static weights and biases. ONNX acts strictly as a format and does not execute mod-

els; rather, execution is handled by external runtimes that map the graph to their respective hardware kernels, such as TensorRT or ONNX Runtime (a separate execution provider developed by Microsoft). This work utilizes a more direct, yet widely adopted compilation approach via NVIDIA’s torch2trt [39] library to convert models directly from PyTorch to TensorRT. This approach avoids intermediate graph serialization and simplifies the conversion pipeline, enabling tighter integration with the PyTorch-based experimentation setup. The methodology uses TorchScript tracing, a PyTorch module that traces the execution of a neural network so that its operations can be reconstructed into an independent flow outside the Python runtime. Within torch2trt [39], hooks are defined for a comprehensive array of standard PyTorch operations. As the tracer processes the model layer by layer, it replaces PyTorch function calls with equivalent TensorRT network API calls while maintaining the original programmatic structure. Consequently, the resulting TensorRT engine object can be used through a similar Python interface to the original PyTorch model. There are inherent limitations to this approach, primarily because TorchScript tracing constructs the computational graph based on a single forward pass of a dummy tensor. If the model contains input-dependent branching, tracing may fail to capture the complete computational graph. For example, a conditional execution path such as:

$$f(x) = \begin{cases} \text{ReLU}(x), & x > 0 \\ \text{Sigmoid}(x), & x \leq 0 \end{cases}$$

would only trace the specific branch executed by the input tensor used during the tracing step. Furthermore, direct translation may not be natively supported for all custom layers, particularly those involving dynamic behavior or non-standard operations. While custom modules composed entirely of standard PyTorch primitives can generally be translated seamlessly, torch2trt [39] occasionally struggles with highly specialized layers. To resolve this, the library provides an extensible registry to map custom layers directly to TensorRT primitives. This mechanism is demonstrated through the following code snippet for registering a custom ReLU6 activation:

```
import torch
import torch.nn as nn
import tensorrt as trt
from torch2trt import torch2trt, tensorrt_converter

# Define the custom layer in PyTorch
class ReLU6(nn.Module):
    def __init__(self, inplace=False):
        super(ReLU6, self).__init__()
        self.inplace = inplace

    def forward(self, x):
        return torch.clamp(x, 0, 6)
```

```

# Define a decorator to register the TensorRT translation mapping
@tensorrt_converter('ReLU6.forward')
def convert_relu6(ctx):
    input_trt = ctx.method_args[1]._trt
    output_trt = ctx.method_return

    # Add clamp layer in TensorRT to mimic ReLU6:  $x = \min(\max(x, 0), 6)$ 
    layer = ctx.network.add_activation(
        input_trt,
        type=trt.ActivationType.CLIP
    )
    layer.alpha = 0.0      # min value
    layer.beta = 6.0       # max value
    output_trt._trt = layer.get_output(0)

```

Through this direct tracing methodology and the explicit registration of unsupported custom layers, this thesis ensures a consistent and reproducible conversion pipeline within defined constraints, providing a stable baseline for all subsequent throughput measurements.

3.2 Design of Experimentation

3.2.1 Metrics

The performance and complexity of Convolutional Neural Network (CNN) models deployed on edge hardware are quantified using a specific set of evaluative metrics. In addition to standard computational measurements which define the speed and footprint of the model this thesis formally treats the model’s high-level architecture itself as a primary categorical design variable.

Architecture ($\mathcal{A}$): A family of models sharing similar design patterns, operational blocks, and layer connectivity. For example, ResNet18 and ResNet50 belong to the ResNet architecture $\mathcal{A}$, while MobileNetV2 with varying width multipliers ($width = x \mid x \in (0, 1]$) belongs to the MobileNet architecture. This thesis utilizes $\mathcal{A}$ to categorize models, as empirical observations indicate that grouping models by their architectural family reveals more consistent performance trends against other metrics.

Throughput ($\mathcal{T}_{r,b}$): The number of inputs that can be successfully inferred per unit time. If the inputs are images and the batch size (b) is 1, $\mathcal{T}$ is measured in Frames Per Second (FPS); otherwise, it is measured in Hertz (Hz). The variable r denotes the input resolution, with a default assumption of $r = 224$ (implying a 224×224 spatial resolution) unless otherwise specified. While $b = 1$ is the primary operational use case for real-time video streams, throughput can also measure batches, embeddings, or patches processed per second. For instance, in Section 5.1, throughput is evaluated as the number of 112×112 patches processed in parallel per unit time.

Latency: The total time required by the network to complete the inference of a single input. It is approximated as the inverse of throughput at a batch size of one: $\ell = \mathcal{T}_{b=1}^{-1}$.

Model GFLOPs ($\mathcal{F}$): The theoretical computational complexity of the model, defined as the number of Floating-Point Operations (FLOPs) required for the inference of a single input. While 10^9 Floating-Point Operations (GFLOPs) are widely used in literature to estimate model complexity, prior work has demonstrated that $\mathcal{F}$ is an indirect and often weak proxy for actual on-device metrics such as $\mathcal{T}$.

Number of Parameters ($\mathcal{P}$): The total count of parameters (weights and biases) within the model. Experimentally, $\mathcal{P}$ has been observed to be a stable metric for characterizing model scale within a specific architecture $\mathcal{A}$.

Primarily, this work utilizes $\mathcal{P}$ in conjunction with $\mathcal{A}$ to define and evaluate a model. Section 3.3 introduces a formulation to estimate $\mathcal{T}$ based on these two metrics.

While $\mathcal{F}$ is occasionally useful for drawing broad comparative inferences when grouped by $\mathcal{A}$, this thesis preferentially substitutes $\mathcal{F}$ with $\mathcal{P}$ due to three inherent limitations of GFLOPs as an evaluative metric:

- **Input-Size Dependency:** $\mathcal{F}$ is heavily dependent on input resolution and shape, creating significant variations in the metric even when the underlying model remains unchanged.
- **Measurement Ambiguity:** GFLOPs can be calculated theoretically (summing mathematical operations across layers) or measured empirically via runtime profilers. However, empirical measurements fluctuate significantly depending on the implementation details, the underlying execution framework, and the specific profiler used.
- **Incomplete Systems Profiling:** GFLOPs are frequently, and incorrectly, equated to the total “work done” by a model. In reality, mathematical operation counts fail to account for critical systems-level bottlenecks such as memory access costs (data movement overhead), cache utilization, and the degree of parallelizability, all of which heavily dictate actual execution speed [9] [10].

Conversely, $\mathcal{P}$ is an immutable structural property of the model itself. Independent of input size, it serves as a strong indicator of the static memory footprint and serialized model size. Furthermore, it is deterministic and trivially measurable without requiring specialized profiling tools.

Finally, regarding temporal performance, $\mathcal{T}$ is systematically favored over ℓ . Throughput naturally adapts to various deployment contexts whether processing multiple inputs simultaneously or handling sequential video streams.

3.2.2 Setup and Measurement Protocol

All evaluations on the edge hardware are conducted in headless mode operating without a graphical user interface (GUI) or connected display. Operating in headless mode accurately reflects real-world autonomous edge deployment scenarios where devices function without user interaction. Crucially, this conserves limited system resources (Central Processing Unit (CPU), Graphics Processing Unit (GPU),

and shared memory) that would otherwise be consumed by OS-level rendering processes, thereby preventing interference and ensuring more consistent timing of model inference. Throughput ($\mathcal{T}$) is measured across three distinct runtime configurations: standard PyTorch [38], TensorRT [36] at 32-bit Floating Point Precision (FP32), and TensorRT at 16-bit Floating Point Precision (FP16). As established in Section 3.1, these native environments and torch2trt [39] conversion pipelines provide a representative deployment and baseline software stack for the Jetson Nano platform. Figure 3.1 depicts the overarching measurement protocol. To isolate the pure inference cost of the model and eliminate data-pipeline overhead, the protocol is strictly independent of specific datasets. Instead, inputs are generated as random PyTorch CUDA tensors and preloaded directly into GPU memory prior to timing the forward pass. Because standard convolutional operations depend on tensor dimensions rather than input semantics, randomly generated tensors provide a mathematically valid proxy for measuring pure computational cost. Aside from the specific model being evaluated and the shape of input, the protocol requires two primary variables: the trial size ts , defining the number of iterations timed within a single trial, and the batch size (b). Unless explicitly specified otherwise, the default batch size is assumed to be $b = 1$ throughout all experimentation.

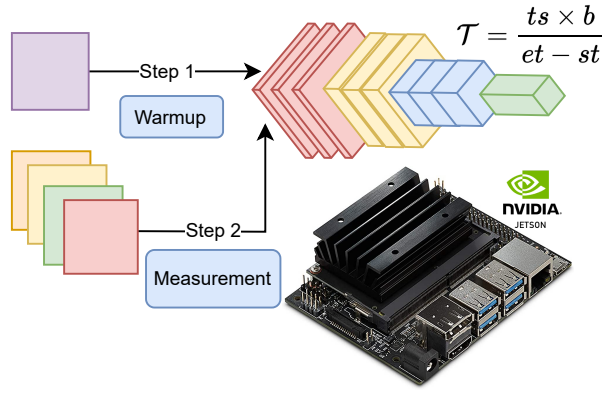


Figure 3.1 Deployment Environment: The NVIDIA Jetson Nano serves as the representative edge hardware. Experimental consistency is enforced via hardware-level CUDA stream synchronization, tensor preloading, and isolated warm-up passes. The measurement protocol is executed across standard PyTorch [38] and TensorRT [36] utilizing both FP32 and FP16 precision. Key measurement variables include the trial size (ts), batch size (b), start time (st), and end time (et). Details can be found in Protocols 1 and 2.

The specific procedures utilized to capture these measurements are formalized in Algorithms 1 and 2.

Algorithm 1 Measure Throughput ($F_{\mathcal{T}}$)

Input:

$M \leftarrow$ Model to be measured
 $ts \leftarrow$ Number of trials
 $b \leftarrow$ Batch size
 $s \leftarrow$ Input shape excluding batch dimension (l, w, h)
 $X \leftarrow$ Random tensor of shape ($ts * b * s$)
 $w \leftarrow$ Random tensor of shape ($1 * s$) (warm-up sample)

Notation: $*$ denotes concatenation of tensor dimensions. Given $a \in \mathbb{R}^{a_1, a_2, \dots, a_m}$ and $b \in \mathbb{R}^{b_1, b_2, \dots, b_n}$,

$$a * b \in \mathbb{R}^{a_1, a_2, \dots, a_m, b_1, b_2, \dots, b_n}.$$

Procedure:

Perform warm-up pass: $y \leftarrow M(w)$ ▷ Warm-up run
Synchronize CUDA stream ▷ Ensure pending operations complete
Record start time: $st \leftarrow$ Current time
for $i = 0$ **to** $ts - 1$ **do**
 $y \leftarrow M(X[i])$
end for
Synchronize CUDA stream ▷ Ensure pending operations complete
Record end time: $et \leftarrow$ Current time
Compute throughput:

$$\mathcal{T} \leftarrow \frac{ts \times b}{et - st}$$

Return $\mathcal{T}$

Algorithm 2 Measure throughput across PyTorch, TensorRT, and FP16

Input:

$M_{py} \leftarrow$ PyTorch model
 $s \leftarrow$ Input shape
 $n \leftarrow$ Number of independent trials
 $x \leftarrow$ Generate random input tensor of shape s

Definitions:

$F_{\mathcal{T}}$ as defined in Algorithm 1
 $G_{\mathcal{T}} \leftarrow$ Repeat $F_{\mathcal{T}}$ n times and return mean.
torch2trt: NVIDIA API for converting PyTorch model to TensorRT

Procedure:

Benchmark PyTorch model:

$$T_{py} \leftarrow G_{\mathcal{T}}(M_{py})$$

Convert PyTorch model to TensorRT:

$$M_{trt} \leftarrow \text{torch2trt}(M_{py}, x, \text{fp16=False})$$

Benchmark TensorRT model:

$$T_{trt} \leftarrow G_{\mathcal{T}}(M_{trt})$$

Convert PyTorch model to TensorRT with FP16 precision:

$$M_{trt16} \leftarrow \text{torch2trt}(M_{py}, x, \text{fp16=True})$$

Benchmark TensorRT FP16 model:

$$T_{trt16} \leftarrow G_{\mathcal{T}}(M_{trt16})$$

Return $T_{py}, T_{trt}, T_{trt16}$

The Synchronization Step: A critical component of these algorithms is the explicit synchronization of the hardware. Standard PyTorch execution on CUDA devices operates asynchronously; when a forward pass is called, the CPU queues the operations to the GPU and immediately proceeds to the next line of execution. Consequently, a Python-based timer would incorrectly record the dispatch time rather than the actual inference completion time. To ensure measurement integrity, the protocol invokes `torch.cuda.current_stream().synchronize()` before recording the start time (*st*) and end time (*et*). This command forces the host CPU to wait until all pending computations on the GPU are fully resolved,

ensuring that the measured duration closely reflects the time taken for the physical hardware to execute the assigned neural network operations.

3.3 Throughput Estimation of Face Recognition Backbones

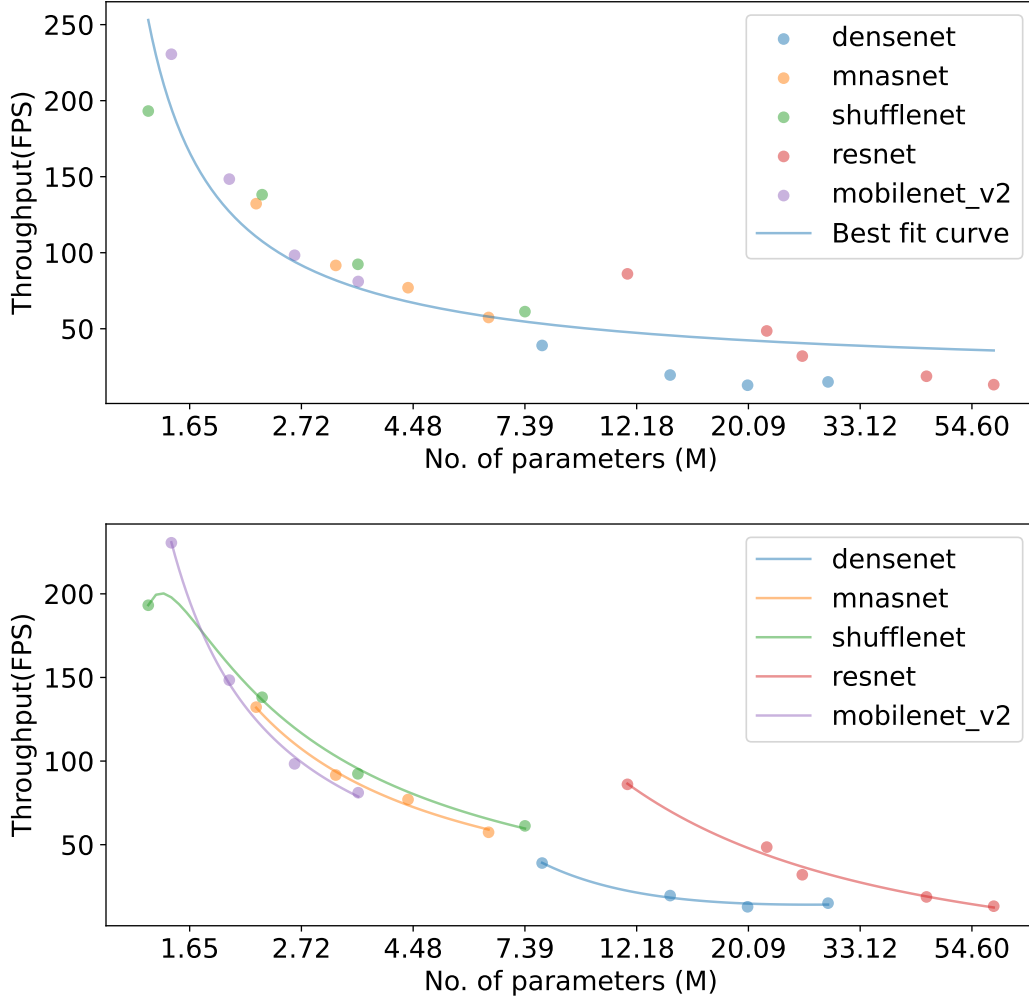


Figure 3.2 A plot of throughput against model scale (number of parameters). Each point corresponds to an empirically observed benchmark. Models belonging to the same architecture $\mathcal{A}$ [7] are represented with a circular marker of an identical color. The X-axis is plotted on a logarithmic scale. The naive global approach (top) demonstrates poor predictive performance. Conversely, the proposed architecture-aware estimator (bottom) achieves a highly accurate fit by grouping models categorically by $\mathcal{A}$, yielding an 84% reduction in Mean Absolute Error.

This section formulates a mapping from models of a specific architecture ($\mathcal{A}$) to their resultant throughput ($\mathcal{T}$), posing performance estimation as a regression-based learning problem. The empirical dataset utilized to train and fit this estimator function is defined within this thesis as “Landmarks” - $\mathcal{L}$.

Mathematically, $\mathcal{L}$ is a set of tuples where each entry pairs a model representation with its measured throughput: $(m, \mathcal{T})$. To facilitate regression, the model m is parameterized as a tuple of $(\mathcal{A}, \mathcal{P})$, where $\mathcal{A}$ denotes its architectural family and $\mathcal{P}$ denotes its total parameter count. For example, a landmark entry corresponding to ResNet18 [7] operating at 90 Frames Per Second (FPS) would be represented as $((\text{ResNet}, 11.69 \times 10^6), 90)$.

Crucially, the set $\mathcal{L}$ is strictly bound to the specific physical device upon which it was measured. For instance, consider a set of models M evaluated across three edge devices: D_1, D_2 , and D_3 , where D_1 and D_2 are Jetson Nanos [34] of identical hardware configurations, and D_3 is a different architecture entirely. Let $\mathcal{L}_{D_i}$ denote the set of measurements made on $D_i \in \{D_1, D_2, D_3\}$ for all $m \in M$. Expectedly, $\mathcal{L}_{D_1} \neq \mathcal{L}_{D_3}$ due to varying computational capacities. However, empirical observation reveals that $\mathcal{L}_{D_1} \neq \mathcal{L}_{D_2}$ as well. Because of minor intra-device manufacturing variances and operational conditions, (explored further in Sec 3.4) $\mathcal{L}$ cannot be generalized to a broad class of hardware; it is most accurately defined for the specific device under test. Thus, $\mathcal{L}$ acts as a localized proxy to capture the exact performance profile of a specific hardware unit, justifying the term “Landmarks.”

Observed throughput ($\mathcal{T}$) exhibits measurable variance across sequential executions of the identical model, driven by dynamic systems-level factors including thermal throttling, runtime memory fragmentation, and background OS scheduling. Consequently, the throughput of a model m is formulated as the sum of a deterministic estimation function Z and an observational noise variable ϵ . It should be noted that $\mathcal{T}$ relies on data captured under the strict, isolated experimental protocol detailed in Section 3.2.2 to minimize the magnitude of ϵ .

$$\mathcal{T}(m) = Z_{\mathcal{L}}(\mathcal{A}, \mathcal{P}) + \epsilon \quad (3.1)$$

where $\mathcal{T}(m)$ refers to the throughput of model m .

Ideally, a model m would be represented as an ordered sequence of parameterized operations and architectural blocks (e.g., specific Convolutional kernels, Residual Blocks, Fully Connected Layers). However, compiling exhaustive benchmarking data across every possible micro-configuration of layers, libraries, and hardware is infeasible. To enable scalable throughput prediction without exhaustive testing, the estimator Z must be computed from a sparsely populated set of $\mathcal{L}$.

To achieve this, this thesis abstracts the ordered set of operations into the categorical variable $\mathcal{A}$ (e.g., $\{\text{ResNet [7]}, \text{MobileNet [25]}, \text{DenseNet [27]}\}$). Experimental results confirm that the parameter count ($\mathcal{P}$), when contextualized by $\mathcal{A}$, captures sufficient structural information to accurately predict relative $\mathcal{T}$. Therefore, both $\mathcal{L}$ and m are denoted strictly in terms of $(\mathcal{A}, \mathcal{P})$. This abstraction ensures the estimator remains data-efficient, computationally lightweight, and trivial to evaluate without requiring deep graph-level profiling.

$$Z_{\mathcal{L}}(m) = Z_{\mathcal{L}}(\mathcal{A}, \mathcal{P})$$

Figure 3.2 (top) illustrates the global distribution of $\mathcal{T}$ against $\mathcal{P}$. Because inference latency inherently increases with model scale, $\mathcal{T}$ exhibits an inverse trend with respect to $\mathcal{P}$, initially suggesting a relationship similar to $y = c/x$. However, a naive global hyperbolic fit yields exceedingly poor approximations. However, by categorically grouping the data points by $\mathcal{A}$, the estimator isolates architecture-specific memory and compute efficiencies.

Figure 3.2 (bottom) plots the architecture-aware estimator $Z_{\mathcal{L}}$ against $\mathcal{P}$. Empirically, while a naive hyperbolic fit shows relatively poorer results, approximating this inverse relationship via a second-degree polynomial regression on $1/\mathcal{P}$ conditioned explicitly on $\mathcal{A}$ yields a highly stable and accurate fit. This produces the final formulation:

$$Z_{\mathcal{L}}(\mathcal{A}, \mathcal{P}) = \frac{C_1}{\mathcal{P}^2} + \frac{C_2}{\mathcal{P}} + I \quad (3.2)$$

where C_1 and C_2 are architecture-specific regression coefficients, and I serves as the intercept. These three parameters are computed independently for each architectural family $\mathcal{A}$.

Train			Test		
MobileNetV2, Width	Predicted	Actual	MobileNetV2, Width	Predicted	Actual
$\alpha = 0.25$	230.92	230.54	$\alpha = 0.2$	245.39	239.46
$\alpha = 0.5$	147.03	148.41	$\alpha = 0.6$	123.45	116.82
$\alpha = 0.75$	100.10	98.34	$\alpha = 0.8$	94.57	95.17
$\alpha = 1$	80.33	81.09	$\alpha = 0.9$	85.87	85.14
Train Error	RMSE	1.20	Test Error	RMSE	4.47
	MAE	1.07		MAE	3.47
Model Name	Predicted	Actual	Model Name	Predicted	Actual
ResNet18	86.36	86.09	ResNet28B	53.70	56.88
ResNet34	44.16	48.56	ResNet40B	37.26	43.13
ResNet50	36.89	32.02	ResNet77N	24.93	23.38
ResNet101	18.80	18.76	ResNet92N	20.79	20.19
ResNet152	12.44	13.23	ResNet134N	14.71	14.71
Train Error	RMSE	2.96	Test Error	RMSE	3.08
	MAE	2.08		MAE	2.24

Table 3.1 Train & Test errors of the proposed estimation method. Absolute errors may be measured in inferences/s at batch size = 1. Test models are custom models made from the same architecture. α refers to the width multiplier, as proposed in [25]. B and N denote the type of blocks used to make custom ResNet models - Basic Residual Block, Bottleneck Block.

As demonstrated in Figure 3.2, implementing this architecture-aware formulation reduces the Mean Absolute Error (MAE) in $\mathcal{T}$ estimation by 84% compared to the naive global fit. The robustness of this hypothesis is further validated through the interpolation of non-standard model configurations. Detailed predictive results are provided in Table 3.1.

For validation testing, custom models were generated by scaling standard architectures, including MobileNetV2 [26] with varying width multipliers and ResNets [7] of varying block depths. The resulting mean absolute errors were exceptionally low: 3.4 FPS for MobileNetV2 [26] derivatives and 3.08 FPS for ResNet [7] derivatives. Analyzed at the individual data-point level, the discrepancy between predicted and actual throughput falls within the expected bounds of on-device observational noise (ϵ). Therefore, this thesis demonstrates that the proposed estimator function $Z_{\mathcal{L}}$ reliably predicts custom model throughput from highly constrained data without exhibiting overfitting. By treating $\mathcal{L}$ as training data and $(\mathcal{A}, \mathcal{P})$ as localized features, the mapping of custom architectural variations to edge throughput is posed as a learning problem with an explainable and data-efficient solution.

3.4 Quality of estimates: Noise & on-device fluctuation

The formulation of the Landmark set (L) reduces the barrier to designing custom, task-specific models for the edge. During the initial architectural design phase, baseline Landmark data can be sourced from large-scale public benchmarking repositories such as MLPerf [13] and AI Benchmark [11, 12]. Leveraging these public datasets allows researchers to approximate expected throughput and substantially reduce the uncertainty of computational performance prior to hardware procurement.

However, while generalized public data is invaluable for preliminary prototyping, translating these estimates to real-world deployment introduces significant physical variability. Hardware performance variance occurs along two distinct axes:

- **Inter-device variance:** differences across distinct hardware architectures or models.
- **Intra-device variance:** fluctuations across identical physical silicon units of the exact same model.

While public benchmarks effectively record inter-device capabilities, they typically present aggregated data or optimized top-tier submissions. Consequently, they mask the specific intra-device variance of an individual physical unit, a phenomenon commonly referred to as the ‘silicon lottery.’

It is a well-documented phenomenon that the performance of silicon dies exhibits significant variation even among units with identical specifications. For instance, prior profiling of edge GPU clusters has revealed up to a 22% variance in computational throughput across identical units, with severe outliers performing 1.5 times slower than the median [40]. Table 3.2 illustrates this intra-device variance on the NVIDIA Jetson Nano platform, comparing the specific physical unit utilized in this thesis against the idealized official benchmarks published by NVIDIA. It is critical to note that the localized mea-

Model Name	PyTorch		TensorRT (FP16)	
	Claimed	Observed	Claimed	Observed
alexnet	46.40	33.80	69.90	81.20
squeezenet1_0	44.00	32.00	137.00	132.00
squeezenet1_1	76.60	51.40	248.00	244.00
resnet18	29.40	20.70	90.20	90.20
resnet34	15.50	11.10	50.70	50.90
resnet50	12.40	9.19	34.20	34.10
resnet101	7.18	5.14	19.90	15.50
resnet152	4.96	3.47	14.10	12.70
densenet121	11.50	5.64	41.90	38.80
densenet169	8.25	5.80	33.20	19.10
densenet201	6.84	4.52	25.40	11.40
densenet161	4.71	3.00	15.60	15.10
vgg11	8.90	9.38	18.30	19.90
vgg13	6.53	7.00	14.70	15.60
vgg16	5.09	5.45	11.90	12.40
vgg11_bn	8.74	9.12	18.40	19.40
vgg13_bn	6.31	6.79	14.80	15.30
vgg16_bn	4.96	5.36	12.00	12.30

Table 3.2 Empirical evaluation of intra-device variance on the NVIDIA Jetson Nano. The table contrasts manufacturer-claimed throughput benchmarks [39] against actual observed performance on the local physical unit utilized in this study. This experiment was conducted on a fresh installation of NVIDIA JetPack using the official benchmarking script provided in the NVIDIA torch2trt repository [39]. The data demonstrates significant, non-linear performance scaling across different model architectures; for instance, AlexNet exhibits substantial degradation under PyTorch but yields unexpected performance gains under TensorRT.

measurements were captured utilizing the official benchmarking script provided in [39], operating under the strict headless environment conditions detailed in Section 3.1.

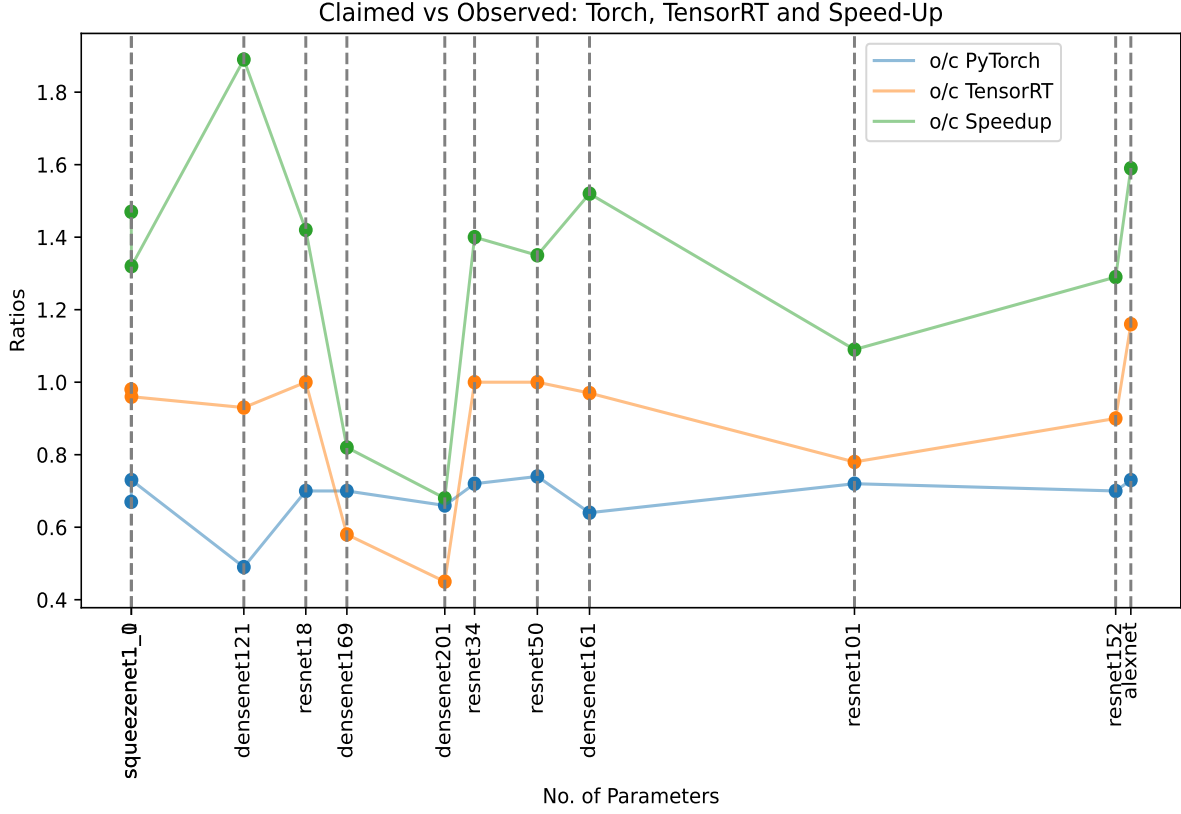


Figure 3.3 A comparative analysis of empirical performance on the local hardware unit against manufacturer-claimed statistics for the NVIDIA Jetson Nano, executed via the official benchmarking script [39]. The plot illustrates the Observed-to-Claimed throughput ratios for PyTorch (blue) and TensorRT (orange). The resulting observed-to-claimed ratio in Speedup (green) exhibits severe oscillations rather than a flat linear trend, providing clear visual evidence that intra-device variance is highly non-linear and explicitly dependent on model architecture.

Figure 3.3 contrasts the empirical performance of the local unit against the claimed manufacturer statistics. The PyTorch baseline (blue line) exhibits relatively consistent attenuation, operating at approximately 70% of the claimed throughput. Conversely, the TensorRT deployment (orange line) operates closer to the claimed baseline but exhibits significantly higher volatility across different model architectures.

To quantify this discrepancy, the relative speedup ratio is defined as:

$$\text{Observed/Claimed Speedup Ratio} = \frac{\mathcal{T}_{\text{TensorRT, Observed}} / \mathcal{T}_{\text{PyTorch, Observed}}}{\mathcal{T}_{\text{TensorRT, Claimed}} / \mathcal{T}_{\text{PyTorch, Claimed}}} = \frac{\text{Observed/Claimed}_{\text{TensorRT}}}{\text{Observed/Claimed}_{\text{PyTorch}}}$$

If intra-device variance manifested purely as a uniform linear scaling factor (e.g., the local unit is uniformly $X\%$ slower due to baseline thermal characteristics), this speedup ratio would resolve to 1.0 across all models. However, the significant oscillations in the speedup curve (green line) provide clear empirical evidence that intra-device variance is highly non-linear and explicitly architecture-dependent.

This non-linearity is particularly evident in memory-intensive architectures. For example, the TensorRT execution of DenseNet121 matches claimed performance (ratio ≈ 1.0), whereas its PyTorch execution operates at roughly half capacity (ratio ≈ 0.5). Conversely, for DenseNet169 and DenseNet201, the TensorRT ratio unexpectedly drops below the PyTorch ratio.

Given that the test environment utilizes a minimal, headless installation with no extraneous computational overhead, these anomalies are unlikely to be attributable to user-space interference. Instead, they highlight critical systems-level bottlenecks: heavy architectures approach or exceed the Jetson Nano’s shared 4GB memory capacity, likely triggering memory pressure and OS-level memory management mechanisms (e.g., paging or swapping). These background OS interventions introduce unpredictable, non-linear latency spikes that disproportionately penalize specific computational workloads.

This demonstrates that idealized manufacturer benchmarks may be insufficient for rigorous edge deployment planning; intra-device variance and OS-level constraints must be accounted for. The Landmark-based throughput estimation approach proposed in Section 3.3 directly neutralizes this vulnerability. By utilizing L as a localized proxy, the estimator implicitly captures the specific thermal, silicon, and memory characteristics of the target hardware. As established, the predictive error of the Landmark based estimator is small enough to be considered as noise in observation, rendering it empirically more reliable than generalized manufacturer claims for on-device deployment estimation.

With the throughput estimation framework established and its robustness under intra-device variability characterized, the subsequent chapters transition from methodological formulation to targeted empirical analysis. Specifically, Chapter 4 investigates the deployability of Face Anti-Spoofing (FAS) models on the edge, with emphasis on the trade-off between inference throughput and biometric error rates (ACER/HTER). Chapter 5 extends the scope to Face Recognition (FR) backbones, examining throughput scaling across input resolutions, hardware-level constraints such as floating-point precision and memory bottlenecks, and the trade-off between discriminative capacity and edge inference performance.

Chapter 4

Deployability of Face Anti-Spoofing Models

4.1 Introduction

This chapter examines the deployability of Convolutional Neural Network (CNN) architectures used in Face Anti-Spoofing (FAS), with a specific focus on their inference behavior particularly throughput-accuracy trade-offs on resource-constrained edge hardware.

Modern Face Recognition (FR) systems have demonstrated high accuracy following the widespread adoption of deep learning. For instance, baseline architectures such as FaceNet [41] achieved 99.63% accuracy on the Labeled Faces in the Wild (LFW) dataset as early as 2015. However, these systems are designed primarily to perform identity verification; they do not inherently assess the physical authenticity of the input. As a result, standard FR pipelines remain highly vulnerable to malicious attempts at impersonation.

Attacks on facial biometric systems are broadly categorized into adversarial attacks and presentation attacks (PAs). Adversarial attacks exploit the mathematical sensitivity of neural networks to carefully crafted perturbations, causing misclassification without perceptible changes to human observers. In contrast, presentation attacks involve physical artifacts presented directly to the sensor, such as printed images, replayed digital videos, or three-dimensional masks.

As illustrated in Figure 4.1, FAS systems are typically integrated as a gatekeeping module within the broader authentication pipeline. In common serial configurations, the FAS module must validate the liveness of the input before it is processed by the FR system. Consequently, the inference latency of the FAS model directly influences the overall throughput of the security system.

Detecting liveness from a single RGB image is a fundamentally constrained computer vision problem. While multi-modal approaches such as stereo depth sensing or infrared imaging can significantly improve robustness, they introduce additional hardware complexity, cost, and power requirements. Therefore, most practical edge deployments rely on monocular RGB input, placing the primary burden of liveness detection on the deployed model.

This hardware constraint drives the development of highly specialized, hand-crafted CNN architectures tailored specifically for anti-spoofing tasks. However, architectural modifications designed to capture subtle spoofing artifacts do not always map efficiently to standard edge-acceleration libraries

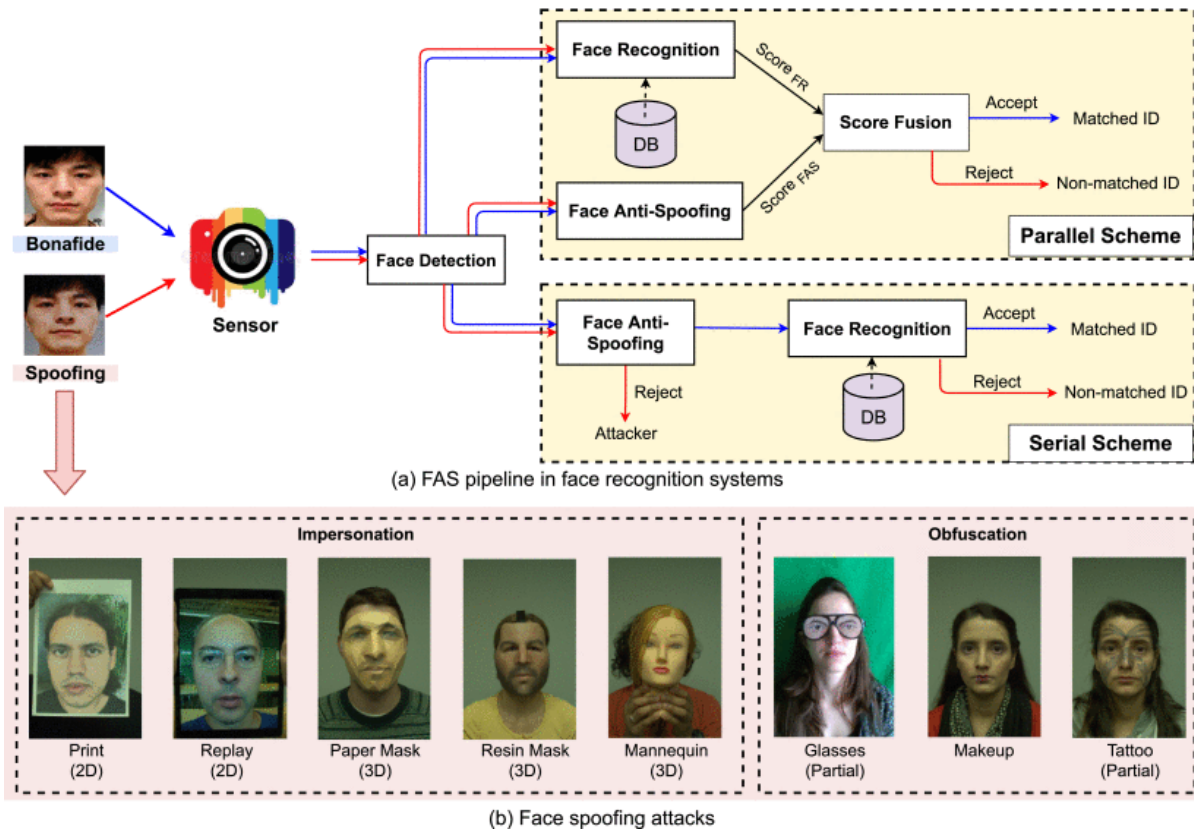


Figure 4.1 Typical face spoofing attacks and the FAS integration pipeline. The FAS module acts as a critical gatekeeper (often in a serial scheme), meaning its inference latency directly impacts the total throughput of the authentication system. This thesis analyzes the edge-inference performance of CNNs designed to detect such presentation attacks. Adapted from Yu et al. [42].

like TensorRT. Thus, improvements in benchmark accuracy do not necessarily translate to deployable performance.

Accordingly, the focus of this chapter is not to explore the algorithmic intricacies or domain generalization of FAS methods, but to evaluate their behavior under strict deployment constraints. Specifically, it analyzes the relationship between model design, inference throughput, and standardized error metrics, highlighting the trade-offs that arise when translating theoretical FAS performance into practical edge deployment.

4.2 Datasets and Metrics in FAS

4.2.1 Metrics

Unlike generic computer vision tasks such as object detection or image classification, Face Anti-Spoofing (FAS) commonly formalized in standardization literature as Presentation Attack Detection (PAD) relies on domain-specific datasets and evaluation metrics. To contextualize the throughput-accuracy trade-offs analyzed in this chapter, it is necessary to define the primary error metrics used to quantify a model’s ability to distinguish *bona fide* (authentic) inputs from spoofed artifacts.

This thesis primarily references the Half Total Error Rate (HTER) and the Average Classification Error Rate (ACER) to measure the prediction performance of Face Anti-Spoofing (FAS) models, and uses these metrics in conjunction with throughput measurements to evaluate trade-offs between predictive performance and deployability on edge hardware.

HTER is a legacy biometric error metric that balances the False Acceptance Rate (FAR) and the False Rejection Rate (FRR):

$$HTER = \frac{FAR + FRR}{2}$$

where

$$FAR = \frac{\text{False Acceptances}}{\text{Total Attack Samples}}, \quad FRR = \frac{\text{False Rejections}}{\text{Total Genuine Samples}}$$

- **False Acceptance Rate (FAR):** The proportion of spoof (attack) samples incorrectly accepted as genuine.
- **False Rejection Rate (FRR):** The proportion of genuine samples incorrectly rejected as spoof.

While HTER provides a singular numerical score that balances false positives and false negatives, it is highly sensitive to the choice of classification threshold.

In modern FAS evaluation, ACER is widely adopted as a standard metric. It is the average of the Attack Presentation Classification Error Rate (APCER) and the Bona Fide Presentation Classification Error Rate (BPCER):

$$ACER = \frac{APCER + BPCER}{2}$$

where

$$APCER = \frac{\text{Attack Samples Misclassified as Bona Fide}}{\text{Total Attack Samples}}, \quad BPCER = \frac{\text{Bona Fide Samples Misclassified as Attacks}}{\text{Total Bona Fide Samples}}$$

- **APCER:** The proportion of attack samples misclassified as *bona fide* (a failure to detect a spoof).
- **BPCER:** The proportion of *bona fide* samples misclassified as attacks (a false rejection of a legitimate user).

Although HTER and ACER share a similar formulation as averages of two error rates, they are defined within different evaluation protocols and serve distinct roles. HTER is a general biometric error metric, whereas ACER is specifically defined for Presentation Attack Detection (PAD) tasks. ACER is formally recommended by international standards such as ISO/IEC 30107-3 [43] for assessing the resilience of biometric systems against spoofing.

These metrics quantify the predictive performance of a model. Evaluating deployability on edge devices, however, requires analysis of inference throughput ($\mathcal{T}$) and model size - no. of parameters ($\mathcal{P}$). Consequently, these factors are used to characterize inference behavior on edge devices, as presented in Section 4.3.

4.2.2 Datasets

Several benchmark datasets are routinely employed to evaluate Face Anti-Spoofing (FAS) systems. Each introduces specific sources of variability such as illumination, device type, and attack modality intended to stress the generalization capacity of PAD algorithms. While this thesis does not focus on solving the generalization problem itself, establishing these standardized benchmarks is critical for evaluating whether a model’s theoretical accuracy justifies its computational cost.

Oulu-NPU [44] is one of the most extensively used benchmarks for mobile-domain FAS. It consists of 5,940 video recordings from 55 participants, collected over three sessions using six different front-facing smartphone cameras. Because it utilizes mobile hardware, it serves as a representative benchmark for the real-world edge authentication scenarios targeted in this thesis. The dataset contains 1,980 *bona fide* access attempts and 3,960 attack samples produced via printed photographs and video replays. To facilitate systematic evaluation, Oulu-NPU defines four protocols:

- **Protocol I:** Evaluates robustness under unseen environmental conditions (illumination and background changes).
- **Protocol II:** Tests generalization to previously unseen presentation attack instruments (PAIs).
- **Protocol III:** Investigates cross-device performance using leave-one-camera-out training and testing splits.

- **Protocol IV:** Combines all three axes of variability, yielding the most challenging assessment setting.

Earlier datasets, though smaller in scale, remain influential for cross-dataset evaluation. **Replay-Attack** contains approximately 1,300 video sequences recorded under controlled and adverse lighting conditions. **CASIA-FASD** [45] extends variability by incorporating different attack modalities across 600 videos, while **MSU-MFSD** introduces additional device diversity through recordings on both smartphones and laptops. These datasets are frequently utilized in cross-domain studies, where models are trained on legacy sources and evaluated on target datasets like Oulu-NPU to test domain generalization.

More recent resources, such as **CASIA-SURF** [46], have expanded the evaluation paradigm to multi-modal sensing, providing aligned RGB, depth, and infrared streams. However, as established in Section 4.1, edge devices often lack the hardware required to capture multi-modal data. Consequently, architectures that report State-of-the-art (SOTA) results exclusively on CASIA-SURF (such as FeatherNets [47]) cannot be directly compared to RGB-only deployments due to this modality mismatch.

By outlining the commonly used datasets and ACER/HTER metrics, a standardized reference for evaluating predictive performance in FAS is established within the scope of this thesis. However, comparisons across models are not always directly valid, as reported performance is inherently tied to the choice of dataset, evaluation protocol, and sensing modality. Furthermore, predictive error rates alone are insufficient to characterize inference throughput on edge devices. This motivates the joint evaluation of predictive performance with inference throughput and model size in the subsequent section.

4.3 Observed Edge Behavior and Trade-offs

Standard Face Recognition (FR) pipelines typically rely on generalized backbones (as established in Section 2.2). In contrast, FAS approaches tend to utilize highly specialized, task-specific architectures, as detailed in Table 4.1. While the development of task-specific models aligns with the broader objective of optimizing networks for edge environments, the practical deployability of these handcrafted FAS architectures warrants critical evaluation. To this end, this chapter presents a joint evaluation of their predictive error rates (measured via HTER and ACER) and hardware deployability - measured via inference throughput, $\mathcal{T}$.

Crucially, for custom models containing operations without native TensorRT support, the throughput estimation methodology developed in Chapter 3 could not be applied due to the lack of standardized architectural data and insufficient data points. Instead, the throughput for these specific models is empirically approximated by temporarily bypassing the unsupported layers within the PyTorch graph, allowing the compiler to measure the baseline $\mathcal{T}$ of the remaining supported network structure.

Task-specific FAS architectures are designed to capture a broad spectrum of subtle spoofing artifacts, frequently incorporating inductive biases to mine localized features such as texture anomalies, color distortions, and liveness cues. While these specialized designs consistently demonstrate State-of-the-art (SOTA) predictive performance on standardized protocols, the relatively small scale of FAS datasets in-

Model Name	$\mathcal{T}$	$\mathcal{P}$	HTER	ACER
FeatherNetA [47]	213.5 [‡]	0.35	-	-
FeatherNetB [47]	187.9 [‡]	0.35	-	-
CDCN* [48]	2.5 [‡]	2.25	-	1
CDCNpp* [48]	2.3 [‡]	2.26	-	0.2
C-CDN HV* [49]	2.5 [‡]	1.25	-	0.6
C-CDN DG* [49]	2.4 [‡]	1.25	-	0.7
DC-CDN* [49]	1.2 [‡]	2.50	18.82	0.4
DBMNet [50]	31.8 [†]	12.44	17.59	-
SSAN-M [51]	2.5 [†]	8.79	19.51	-
SSAN-R [51]	42.8 [†]	8.14	13.72	-
HFN* [52]	12.5 [†]	62.12	12.4	-

Table 4.1 Performance analysis of selected FAS models. * - Approximated throughput $\mathcal{T}$, as models did not execute fully on TensorRT. $\mathcal{P}$ - number of parameters (in millions). [†] - FP16, [‡] - FP32. HTER is computed under cross-dataset evaluation (trained on Replay-Attack [53], CASIA-MFSD [45], MSU-MFSD [54], tested on OULU-NPU [44]). ACER is reported on Protocol 1 of OULU-NPU. FeatherNet models report accuracy only on CASIA-SURF [46], and are therefore not directly comparable.

troduces severe domain mismatch risks. Unlike generalized FR systems whose performance metrics can generally be trusted across diverse real-world conditions due to massive training corpora FAS models frequently overfit to the specific environmental constraints or capture devices of their benchmarks. Furthermore, in an adversarial deployment scenario, malicious actors continuously engineer novel attacks that fall entirely outside this narrow training distribution. Consequently, the theoretical accuracy metrics (HTER and ACER) achieved by these handcrafted architectures cannot be taken at the same face value as standard FR benchmarks; high benchmark accuracy does not guarantee robustness or generalization capability in unpredictable, real-world deployments.

Beyond these generalization risks, evaluating the edge-inference behavior of these task-specific models reveals significant structural bottlenecks. An analysis of Table 4.1 highlights a significant disconnect between the number of parameters ($\mathcal{P}$), predictive accuracy, and actual hardware throughput ($\mathcal{T}$). For instance, models with comparable parameter counts can exhibit throughputs that vary by more than an order of magnitude (e.g., SSAN-M at 2.5 Frames Per Second (FPS) versus SSAN-R at 42.8 FPS under the evaluated configurations in Table 4.1). Broadly, two critical empirical observations emerge. First, architectures utilizing modified convolution operations severely underperform in throughput relative to their parameter count. Second, there is an anomalous distribution of error rates with respect to $\mathcal{T}$; counter-intuitively, higher throughput is observed alongside lower error rates, as demonstrated by SSAN-R.

This throughput deficiency is most apparent in CDCN [48] and DC-CDN [49]. These models replace standard 2D convolutions with Central Difference Convolution (CDC) to capture local gradient features through directional pixel differences, theoretically enhancing sensitivity to fine-grained spoofing artifacts without increasing the overall parameter count. As formally defined by Yu et al. [48], the CDC operation is formulated as:

$$y(p_0) = \underbrace{\sum_{p_n \in \mathcal{R}} w(p_n) \cdot x(p_0 + p_n)}_{\text{vanilla convolution}} + \theta \cdot \underbrace{\left(-x(p_0) \cdot \sum_{p_n \in \mathcal{R}} w(p_n) \right)}_{\text{central difference term}} \quad (4.1)$$

where:

- p_0 : The current pixel location in the input feature map.
- $p_n \in \mathcal{R}$: The offset within the convolution kernel’s receptive field.
- $x(p_0 + p_n)$: The input feature value at the index corresponding to the center p_0 plus the offset p_n .
- $w(p_n)$: The convolution kernel weight at offset p_n .
- θ : A balancing hyperparameter for the central difference term.

This additional term introduces additional operations that deviate from standard 2D convolutions. As a result, these formulations may not be efficiently supported by existing edge deployment frameworks such as TensorRT. Empirically, this is reflected in the significantly low throughput observed for these models. Although DC-CDN proposes Cross Central Difference Convolution (C-CDC) to improve theoretical efficiency by exploiting sparse local features, no corresponding improvement in $\mathcal{T}$ is observed in practice, and the reported ACER on Oulu-NPU Protocol 1 is slightly worse than that of CDCN. This suggests that the observed throughput limitations are influenced, at least in part, by the compatibility of these operations with current hardware and software stacks, and that their effective performance may not be fully realized under the evaluated deployment conditions.

Conversely, architectures like SSAN [51] exhibit an outlier relationship between throughput and predictive accuracy. For instance, SSAN-M demonstrates notably low throughput alongside a higher HTER compared to SSAN-R, despite utilizing official implementations. While a significant throughput gap exists between SSAN-R and Hierarchical Fusion Network (HFN) [52], drawing direct architectural conclusions is challenging, as model sizes often increase disproportionately as error rates saturate. However, across comprehensive cross-database testing, SSAN-R consistently exhibits lower HTER and a significant performance improvement over baselines like DBMNet [50].

The findings of this empirical study indicate that a linear correlation cannot be assumed between theoretical predictive accuracy and edge throughput. The deployment bottlenecks introduced by unsupported or inefficiently mapped custom operations can render several SOTA FAS approaches impractical

for edge deployment, although this may partly reflect limitations in current software and hardware implementations. Given the constraints of hardware acceleration, experimental results indicate that SSAN-R offers the most favorable empirical balance between standardized error rates and inference throughput ($\mathcal{T}$), positioning it as a strong candidate for deployment on the evaluated edge setup.

These findings highlight the disconnect between theoretical predictive performance and practical deployability of FAS models on edge hardware. Building on this, the subsequent chapter extends the analysis to FR backbones, examining system-level factors that govern inference throughput and performance trade-offs.

Chapter 5

Experimentation and Empirical Analysis

This chapter empirically characterizes throughput scaling and hardware utilization constraints in edge inference. By evaluating the backbone architectures used in State-of-the-art (SOTA) Face Recognition (FR) research on the target NVIDIA Jetson Nano using TensorRT, the analysis demonstrates the practical divergence between theoretical arithmetic efficiency and sustained real-world performance. Section 5.1 analyzes the scaling relationship between input resolution and inference throughput, identifying hardware saturation patterns. Section 5.2 then examines hardware utilization bottlenecks, specifically the efficacy of reduced-precision execution and the dominant impact of data pipeline overheads. Finally, Section 5.3 integrates these observations to evaluate the trade-off between architectural discriminative capacity and its viability for edge deployment.

5.1 Throughput Scaling Across Input Resolution

Model Name	Max Resolution	Throughput
ResNet18	1440	2.51
ResNet34	640	6.71
ResNet50	640	4.99
ResNet101	512	4.98
MobileNetV2 0.25	1440	7.37
MobileNetV2 0.5	1080	6.67
MobileNetV2 0.75	1024	5.16
MobileNetV2	896	6.12
RetinaFace Mn25	1440	6.94
RetinaFace Re50	640	3.89

Table 5.1 Maximum Resolutions achieved in experimentation, using TensorRT with FP16 Precision. Beyond these resolutions, random crashes occur, & the process is killed. RetinaFace [55] is paired with Mn25 = MobileNetV2 0.25 & Re50 = ResNet50 backbones.

Input resolution (r) is a critical architectural parameter when deploying convolutional neural networks on resource-constrained edge devices. Functional requirements vary significantly based on the

deployment task; for instance, crowd-level Face Detection necessitates high-resolution inputs to capture sufficient pixel density for individual faces, whereas patch-based Face Anti-Spoofing (FAS) approaches, such as PatchNet [17], typically operate at much lower resolutions. Consequently, characterizing the scaling relationship between inference throughput ($\mathcal{T}$) and r across common backbone families is necessary for efficient deployment.

Table 5.1 details the maximum resolution achieved during experimentation before memory overflow triggered process termination, utilizing TensorRT with FP16 precision. For clarity, resolution is defined as $r = k$, representing a $k \times k$ pixel input. The empirical data demonstrates that sustained operation at high resolutions is feasible primarily for lightweight models, notably ResNet18 and MobileNetV2. To isolate the impact of the backbone from auxiliary task overhead, RetinaFace [55] models are incorporated into the experimentation. The results indicate that the detection-specific overhead introduces only a modest throughput penalty when compared to the standalone backbones (Table 5.1). Specifically, at an $r = 1440$ resolution, the baseline MobileNetV2-0.25 achieves 7.37 Frames Per Second (FPS), which decreases to 6.94 FPS when deployed within the RetinaFace architecture. Similarly, the heavier ResNet50 backbone achieves 4.99 FPS at $r = 640$, dropping to 3.89 FPS under the RetinaFace implementation.

Figure 5.1 plots the decay of $\mathcal{T}$ relative to r . The X-axis corresponds to the total number of input pixels, reflecting a linear progression in spatial volume. The computational cost of convolutional models, measured in 10^9 Floating-Point Operations (GFLOPs), scales approximately linearly with the total number of input pixels. Consequently, under the simplifying assumption where inference latency is proportional to $\mathcal{F}$ and hardware utilization is constant, throughput is expected to follow a hyperbolic decay with increasing resolution.

However, empirical observations demonstrate a deviation from this idealized curve, exhibiting a significantly steeper drop-off at higher resolutions, and conversely, a "clamped" behavior at lower resolutions. While $\mathcal{T}$ reaches its absolute maximum at the minimum tested resolution of 112, the growth in throughput with reduced resolution is less than expected as compared to the decrease in Floating-Point Operations (FLOPs). This suggests that this constraint arises from fixed system-level overheads such as data pipeline staging, Input/Output (I/O) bound runtime switching, and memory transfer latencies which dominate the execution time when the computational payload is small.

The effective input volume processed per unit time has been evaluated to quantify the computational efficiency independently of the latency of a model. For interpretability, the metric is expressed as the number of equivalent 112×112 spatial patches processed per second. Figure 5.2 illustrates this metric across varying resolutions. Note that this metric measures the number of non-overlapping patches or tiles within a single input image and does not imply batching along the input dimension.

The data reveals a very significant jump in patches processed across all models when transitioning from $r = 112$ to $r = 224$. This indicates that at very low resolutions, the computational units of the edge hardware are underutilized, limited primarily by pipeline overhead rather than arithmetic capacity. While increasing r improves raw hardware utilization, it incurs a substantial latency penalty. An

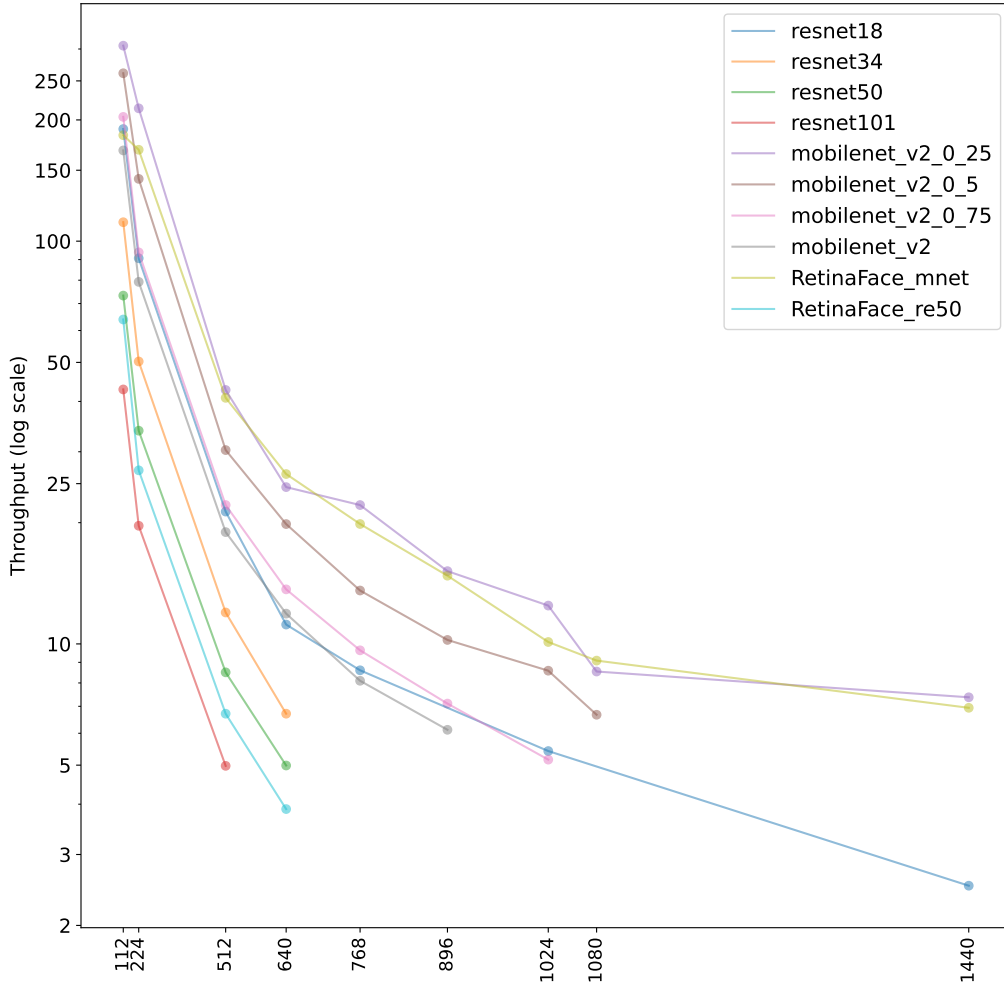


Figure 5.1 Throughput ($\mathcal{T}$) (log scale) against resolution r on the X-axis, where $r = i \implies i \times i$ pixels. The results demonstrate substantial throughput degradation with increasing resolution, while throughput at lower resolutions approaches a performance ceiling rather than continuing to scale proportionally. Measured on an NVIDIA Jetson Nano using TensorRT at FP16 precision.

individual high-resolution stream may block device response for a prolonged duration; for example, the RetinaFace Re50 model operating at $r = 640$ achieves only 3.89 FPS (Table 5.1), equating to an inference latency exceeding 250 milliseconds. Such latency compromises the real-time requirements of interactive edge applications and limits concurrent execution on-device.

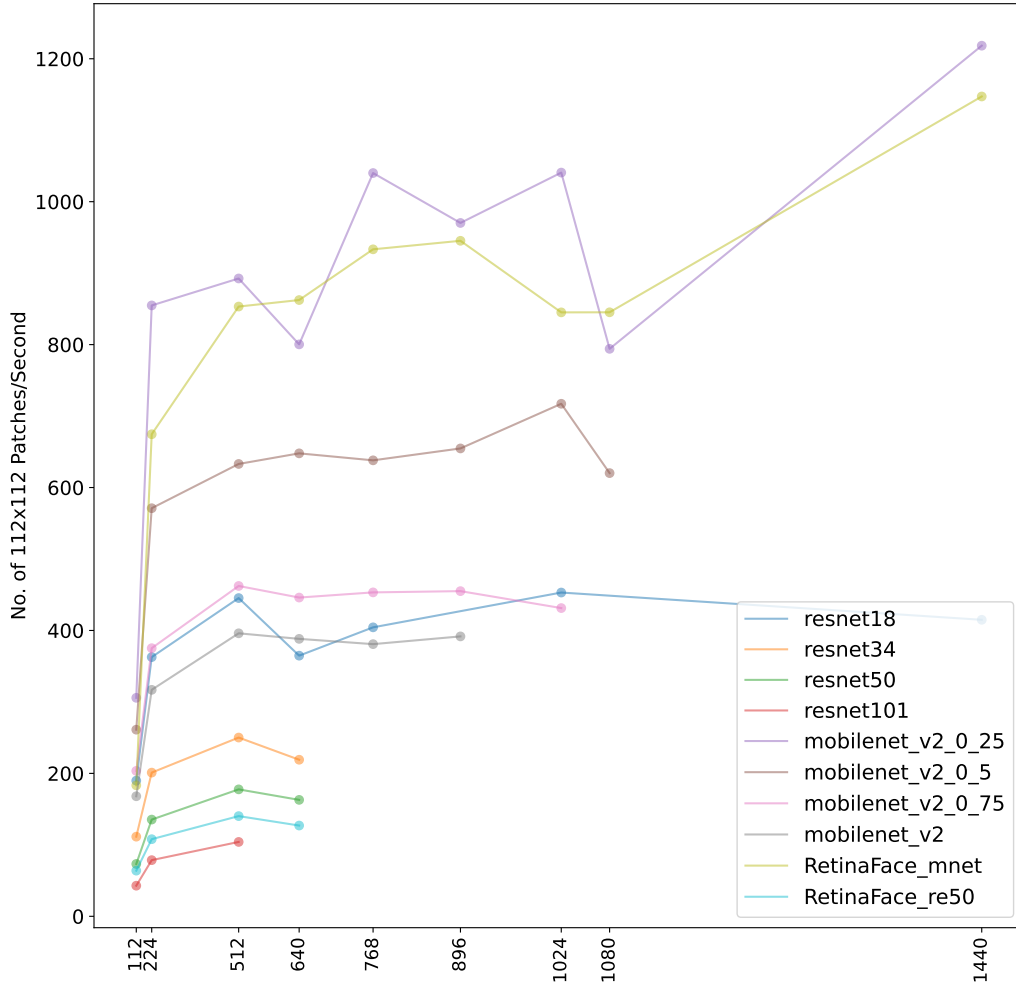


Figure 5.2 Number of equivalent 112×112 patches processed per second against r . This metric estimates the volume of input processed per unit time rather than parallel inputs (batch size). The results demonstrate improved effective hardware utilization at higher resolutions despite declining raw throughput. Measured on an NVIDIA Jetson Nano using TensorRT at FP16 precision.

5.2 Hardware Utilization Constraints

5.2.1 Efficacy of FP16 Precision Reduction

Reducing model precision from a standard 32-bit Floating Point Precision (FP32) to 16-bit Floating Point Precision (FP16) is a standard optimization for resource-constrained deployment. This precision conversion decreases the parameter memory footprint and accelerates arithmetic operations without

requiring the retraining of pre-trained weights (the conversion process specific to this thesis has been presented in Section 3.1.3). However, the extent of inference acceleration depends strongly on the underlying hardware architecture and software support for efficient half-precision execution, such as the optimized FP16 kernels provided by TensorRT.

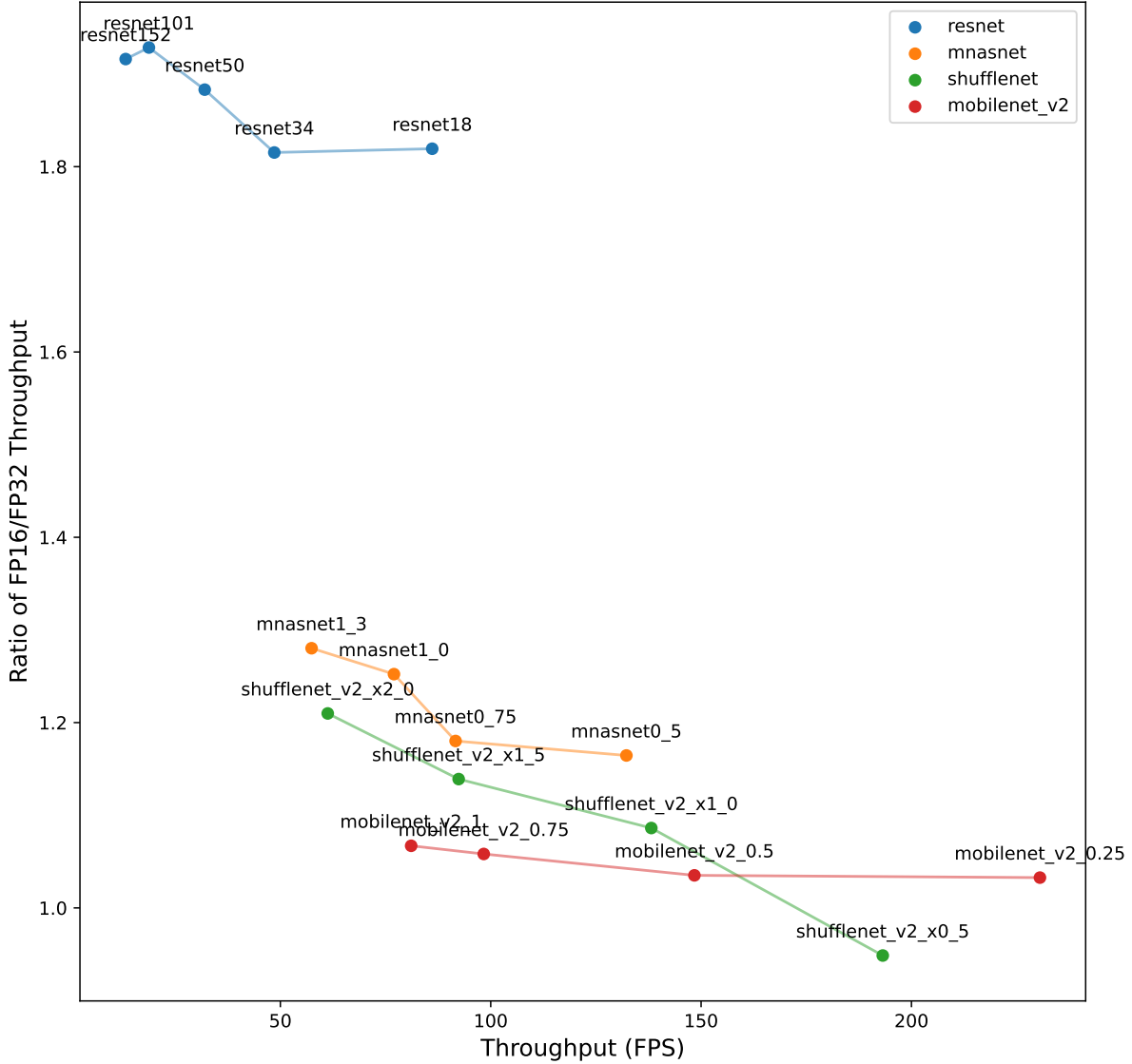


Figure 5.3 Ratio of inference throughput achieved using FP16 relative to FP32 precision on TensorRT, plotted against absolute FP16 throughput. The data demonstrates a distinct architectural separation: heavier, compute-bound models (e.g., ResNets) exhibit substantial speed-up ratios approaching $2\times$ due to improved memory bandwidth utilization and parallel execution efficiency. Conversely, lightweight architectures demonstrate limited gains, indicating a shift toward memory or pipeline-bound operational regimes at high baseline throughputs.

Figure 5.3 illustrates the speed-up ratio S , defined as the inference throughput achieved using FP16 relative to FP32, plotted against the absolute FP16 throughput. As established previously, throughput

generally decreases as parameter count $\mathcal{P}$ increases. Consequently, for most architecture families, S tends to be lowest for the smallest model variants that already operate at a high baseline throughput.

A distinct architectural separation emerges in the data: heavy models, such as ResNets, consistently exhibit higher speed-up ratios under FP16 compared to lightweight families like MobileNetV2, ShuffleNet, and MnasNet. This indicates that models originally designed for full-scale GPU execution benefit substantially from reduced precision, likely due to improved memory bandwidth utilization and parallel execution efficiency. In contrast, compact models demonstrate limited marginal gains. MobileNetV2, in particular, remains quite stable, maintaining an S ratio close to unity with minimal variance across its configurations. MobileNetV2’s specific scaling strategy, which modulates the channel width (layer thickness) rather than overall network depth might be a contributing factor.

In select cases such as FeatherNetA, FeatherNetB (Table 4.1), and ShuffleNetV2 0.5x a slight reduction in throughput ($S < 1$) is observed post-conversion. A common characteristic among these anomalies is their extremely high baseline throughput and lightweight design. This thesis hypothesizes that at very low computational workloads, either the hardware is unable to sustain efficient core utilization when the arithmetic payload per forward pass is further reduced, or the underlying framework implementations are not fully optimized for 16-bit Floating Point Precision (FP16) inference on ultra-lightweight graphs.

5.2.2 Data Pipeline and Memory Transfer Bottlenecks

As demonstrated in Section 5.1, raw computational capacity is not the sole determinant of edge inference throughput. At high-throughput regimes, the operational bottleneck tends to shift from compute to the data pipeline.

Consider the MobileNetV2 (width = 0.25) architecture. Operating at a spatial resolution of 1440×1440 , the model achieves a throughput of $\mathcal{T}=7.37$, which corresponds to effectively processing over 1,200 individual 112×112 patches per second. However, when the model is evaluated directly at a native 112×112 resolution, the observed throughput drops drastically to approximately 300 patches per second, despite the inputs being preloaded directly into memory to bypass storage Input/Output (I/O).

This performance degradation indicates that the computational cost of the model is no longer the limiting factor. Instead, the fixed overhead associated with staging and feeding a large volume of small, discrete input tensors to the GPU saturates the pipeline. The impact of reduced precision as observed in Section 5.2.1 corroborates this. These observations reinforce the conclusions drawn regarding resolution scaling: estimating edge performance solely based on a model’s computational workload such as the Floating-Point Operations (FLOPs) required for a forward pass is fundamentally incomplete. Because the throughput ceiling is ultimately dictated by the host system’s infrastructure rather than the model’s arithmetic complexity, deployment strategies must explicitly account for data pipeline latency and empirically evaluate performance to identify true hardware saturation points, especially at high throughput.

5.3 Trade-off Between Discriminative Capacity and Edge Inference

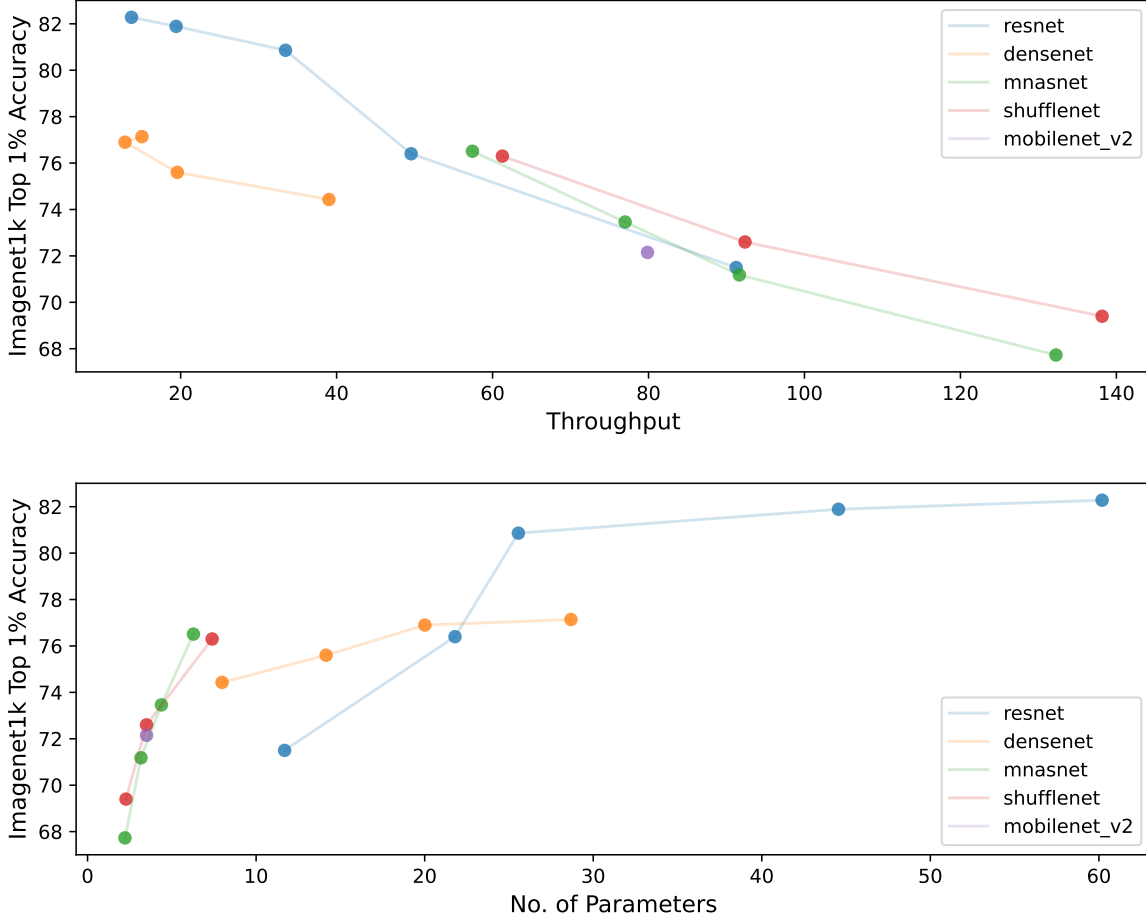


Figure 5.4 Top: ImageNet Top-1 Accuracy against Throughput ($\mathcal{T}$). Bottom: Accuracy against Number of Parameters ($\mathcal{P}$). The results highlight the trade-off between architectural discriminative capacity and edge deployment efficiency.

This section analyzes the trade-off between the discriminative capacity of an architecture and edge inference performance. Figure 5.4 illustrates this relationship, plotting top-1 accuracy against both throughput ($\mathcal{T}$) and parameter count ($\mathcal{P}$).

As illustrated in Section 2.2 and Table 1.1, contemporary advancements in Face Recognition (FR) primarily stem from optimizing loss functions, training frameworks, and data augmentation techniques. These improvements are generally orthogonal to the choice of the backbone architecture, which is often selected independently of the specific training methodology. Furthermore, benchmark accuracy on standard FR datasets, such as LFW [56] and AgeDB [57], is heavily saturated, with State-of-the-art (SOTA) models routinely reporting margins between 96% and 99% and an intra-model variance of $< 1\%$. This

saturation obscures underlying architectural efficacy, making it difficult to decouple the intrinsic strength of the backbone from the specific training methodology.

To isolate and evaluate the raw discriminative capacity of the backbones, this thesis utilizes Top-1 classification accuracy on ImageNet [6] as a proxy metric. Top-1 accuracy provides a stricter, higher-variance baseline than Top-5, effectively exposing the feature extraction capabilities of the architecture independent of specific FR innovation.

Analyzing the empirical data, only the ResNet50 model exceeds an 80% Top-1 accuracy threshold among the standard edge-viable architectures, excluding the significantly heavier ResNet101 and ResNet152. Because accuracy gains plateau while throughput drops exponentially for these larger models, ResNet50 emerges as a highly balanced baseline choice operating at $\mathcal{T}=33$. Conversely, ShuffleNetV2 variants consistently yield the highest accuracy per unit of throughput. While the lightest MnasNet configuration matches the baseline ShuffleNetV2 performance (Figure 5.4, Top), its heavier variants scale poorly by comparison.

Parameter count ($\mathcal{P}$) serves as an additional critical constraint for edge deployment due to strict storage limitations. As shown in Figure 5.4 (Bottom), architectures such as MobileNetV2 and MnasNet-0.3 achieve accuracy comparable to ResNet18 but operate with a drastically reduced $\mathcal{P}$. This data facilitates a targeted trade-off between accuracy and storage footprint at a constant throughput.

However, relying solely on $\mathcal{P}$ is insufficient for evaluating runtime memory footprints. For example, DenseNet121 and DenseNet169 exhibit vastly superior accuracy-to-parameter ratios compared to ResNet18, yet their dense skip-connection topology incurs high memory bandwidth overhead during intermediate feature map allocation at runtime. Therefore, explicitly memory-efficient architectures like ShuffleNetV2, MnasNet, and MobileNetV2 remain preferable for extreme edge environments possessing tighter constraints than the Jetson Nano.

To the best of the author’s knowledge, this thesis is the first work to address the mapping of models of architectures used in Face Recognition (FR) to their inference throughput in an edge deployment setting. It further empirically characterizes the deployability of FAS models, alongside the trade-offs between throughput, resolution, precision, and discriminative capacity in FR backbones. It is hoped that this work serves to reduce the barrier and encourage innovation in the field of FR - specifically in adoption of custom and edge efficient architecture.

Chapter 6

Conclusion

This thesis has examined the deployment of Face Recognition (FR) systems on resource-constrained edge devices, with a specific focus on the relationship between neural architecture and practical inference throughput. While benchmark data exists for selected combinations of models, hardware, and deployment frameworks, scalable evaluation of custom architectures remains infeasible due to the fragmented design space spanning model variants, runtimes, and hardware configurations. Furthermore, consistent with prior observations in hardware-aware CNN literature, this work empirically demonstrates that theoretical metrics such as Floating-Point Operations (FLOPs) are insufficient to characterize real-world deployability, as inference throughput depends not only on arithmetic complexity, but also on architectural execution behavior and system-level factors such as memory movement and pipeline overheads.

Through an analysis of contemporary FR literature, this thesis has identified an underlying architectural structure governing modern FR systems, where the majority of recent innovation is concentrated in training methodology, loss design, and data augmentation, while the backbone architecture is frequently treated as a fixed abstraction [14, 19, 20]. Building on this observation, it is demonstrated that inference throughput of these backbones follow predictable trends when models are grouped into architectural families. This has enabled the development of an explainable, architecture-aware framework capable of estimating the throughput of custom FR backbone models from a sparse set of empirical measurements. The resulting predictions exhibit errors comparable to noise in observations, substantially reducing the need for exhaustive benchmarking on physical hardware. The analysis further indicated that intra-device performance variation may itself depend on architectural behavior, reinforcing the need for localized, hardware-specific throughput estimation.

Beyond predictive modeling, this thesis evaluated Face Anti-Spoofing (FAS) as a necessary auxiliary component within practical FR systems, revealing quantifiable trade-offs between throughput and error rates, along with deployment limitations arising from specialized operations that do not map efficiently to optimized runtimes such as TensorRT [58]. In parallel, empirical experimentation on FR backbones characterized the relationship between discriminative power and inference throughput, while also exposing scaling behavior and throughput saturation caused by data movement and system-level execution constraints.

Collectively, these findings establish that deployability of FR systems on edge hardware cannot be understood solely through theoretical arithmetic complexity or isolated model benchmarks. Instead, deployability emerges from structured interactions between architectural design, runtime behavior, and hardware-level execution constraints. By establishing an explainable mapping between architecture and throughput, this thesis provides a practical framework for hardware-aware exploration of custom FR architectures without exhaustive deployment-time benchmarking.

While this work focuses specifically on FR and FAS, the analytical framework and deployment methodology are readily extensible to broader computer vision workloads operating under edge constraints. Directions for future work include:

- Extending the study across a wider range of edge hardware platforms and deployment frameworks.
- Estimating the throughput of arbitrary models as a composition of their layers or modules.
- Designing backbone architectures optimized specifically for FR embedding tasks rather than generic object classification.

To the best of the author’s knowledge, this is the first work to address scalable throughput estimation of custom FR models through architecture-aware performance modeling on edge devices. The complete benchmarking suite and source code are publicly available at https://github.com/AyanBiswas19/Resource_Efficient_FR.

Bibliography

- [1] Y. Mao, S. Yi, Q. Li, J. Feng, F. Xu, and S. Zhong, “A privacy-preserving deep learning approach for face recognition with edge computing,” in *Proc. USENIX Workshop Hot Topics Edge Comput.(HotEdge)*, 2018, pp. 1–6.
- [2] A. Koubaa, A. Ammar, A. Kanhouch, and Y. AlHabashi, “Cloud versus edge deployment strategies of real-time face recognition inference,” *IEEE Transactions on Network Science and Engineering*, vol. 9, no. 1, pp. 143–160, 2021.
- [3] S. Li, F. Liu, J. Liang, Z. Cai, and Z. Liang, “Optimization of face recognition system based on azure iot edge,” *Computers, Materials & Continua*, vol. 61, no. 3, 2019.
- [4] M. Parhi, A. Roul, B. Ghosh, and A. Pati, “Ioats: An intelligent online attendance tracking system based on facial recognition and edge computing,” *International Journal of Intelligent Systems and Applications in Engineering*, vol. 10, no. 2, pp. 252–259, 2022.
- [5] S. Manzoor, E.-J. Kim, S.-H. Joo, S.-H. Bae, G.-G. In, K.-J. Joo, J.-H. Choi, and T.-Y. Kuc, “Edge deployment framework of guardbot for optimized face mask recognition with real-time inference using deep learning,” *Ieee Access*, vol. 10, pp. 77 898–77 921, 2022.
- [6] 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*. Ieee, 2009, pp. 248–255.
- [7] K. He, X. Zhang, S. Ren, and J. Sun, “Deep residual learning for image recognition,” in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 770–778.
- [8] J. Guo, X. Zhu, C. Zhao, D. Cao, Z. Lei, and S. Z. Li, “Learning meta face recognition in unseen domains,” in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2020.
- [9] 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)*, September 2018.

- [10] 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*, 2019, pp. 2820–2828.
- [11] 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 *Proceedings of the European Conference on Computer Vision (ECCV) Workshops*, 2018, pp. 0–0.
- [12] 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,” in *2019 IEEE/CVF International Conference on Computer Vision Workshop (ICCVW)*. IEEE, 2019, pp. 3617–3635.
- [13] V. J. Reddi, C. Cheng, D. Kanter, P. Mattson, G. Schmuelling, C.-J. Wu, B. Anderson, M. Breughe, M. Charlebois, W. Chou *et al.*, “Mlperf inference benchmark,” in *2020 ACM/IEEE 47th Annual International Symposium on Computer Architecture (ISCA)*. IEEE, 2020, pp. 446–459.
- [14] J. Deng, J. Guo, N. Xue, and S. Zafeiriou, “Arcface: Additive angular margin loss for deep face recognition,” in *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2019, pp. 4685–4694.
- [15] Q. Meng, S. Zhao, Z. Huang, and F. Zhou, “Magface: A universal representation for face recognition and quality assessment,” in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2021, pp. 14 225–14 234.
- [16] H. Qiu, B. Yu, D. Gong, Z. Li, W. Liu, and D. Tao, “Synface: Face recognition with synthetic data,” in *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, October 2021, pp. 10 880–10 890.
- [17] C.-Y. Wang, Y.-D. Lu, S.-T. Yang, and S.-H. Lai, “Patchnet: A simple face anti-spoofing framework via fine-grained patch recognition,” in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2022, pp. 20 281–20 290.
- [18] M. Kim, A. K. Jain, and X. Liu, “Adaface: Quality adaptive margin for face recognition,” in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2022, pp. 18 750–18 759.
- [19] F. Boutros, N. Damer, F. Kirchbuchner, and A. Kuijper, “Elasticface: Elastic margin loss for deep face recognition,” in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR) Workshops*, June 2022, pp. 1578–1587.
- [20] X. An, J. Deng, J. Guo, Z. Feng, X. Zhu, J. Yang, and T. Liu, “Killing two birds with one stone: Efficient and robust training of face recognition cnns by partial fc,” in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2022, pp. 4042–4051.

- [21] S. Wu and X. Gong, “Boundaryface: A mining framework with noise label self-correction for face recognition,” 2022.
- [22] J. Liu, H. Qin, Y. Wu, J. Guo, D. Liang, and K. Xu, “Coupleface: Relation matters for face recognition distillation,” 2022.
- [23] F. Liu, M. Kim, A. Jain, and X. Liu, “Controllable and guided face synthesis for unconstrained face recognition,” in *ECCV*, 2022.
- [24] M. Zhang, G. Song, Y. Liu, and H. Li, “Towards robust face recognition with comprehensive search,” in *Computer Vision – ECCV 2022: 17th European Conference, Tel Aviv, Israel, October 23–27, 2022, Proceedings, Part XII*. Berlin, Heidelberg: Springer-Verlag, 2022, p. 720–736. [Online]. Available: https://doi.org/10.1007/978-3-031-19775-8_42
- [25] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam, “Mobilenets: Efficient convolutional neural networks for mobile vision applications,” *arXiv preprint arXiv:1704.04861*, 2017.
- [26] 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*, 2018, pp. 4510–4520.
- [27] G. Huang, Z. Liu, L. Van Der Maaten, and K. Q. Weinberger, “Densely connected convolutional networks,” in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2017, pp. 4700–4708.
- [28] A. Krizhevsky, I. Sutskever, and G. E. Hinton, “Imagenet classification with deep convolutional neural networks,” *Advances in neural information processing systems*, vol. 25, 2012.
- [29] K. Simonyan and A. Zisserman, “Very deep convolutional networks for large-scale image recognition,” *arXiv preprint arXiv:1409.1556*, 2014.
- [30] 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*, 2015, pp. 1–9.
- [31] 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*, 2018, pp. 6848–6856.
- [32] V. Janapa Reddi, D. Kanter, P. Mattson, J. Duke, T. Nguyen, R. Chukka, K. Shiring, K.-S. Tan, M. Charlebois, W. Chou *et al.*, “Mlperf mobile inference benchmark: An industry-standard open-source machine learning benchmark for on-device ai,” *Proceedings of Machine Learning and Systems*, vol. 4, pp. 352–369, 2022.

- [33] C. Banbury, V. J. Reddi, P. Torelli, J. Holleman, N. Jeffries, C. Kiraly, P. Montino, D. Kanter, S. Ahmed, D. Pau *et al.*, “Mlperf tiny benchmark,” *arXiv preprint arXiv:2106.07597*, 2021.
- [34] Nvidia, “Jetson Nano Developer Kit,” 2023, [Accessed: April 8, 2023]. [Online]. Available: <https://developer.nvidia.com/embedded/jetson-nano-developer-kit>
- [35] NVIDIA Corporation, “NVIDIA Jetson Nano Developer Kit – Technical Specifications,” <https://developer.nvidia.com/embedded/jetson-nano-developer-kit>, 2019, accessed: 2026-01-02.
- [36] Nvidia, “NVIDIA TensorRT,” <https://developer.nvidia.com/tensorrt>, 2023, [Accessed: April, 2023]. [Online]. Available: <https://developer.nvidia.com/tensorrt>
- [37] J. Liu, J. Liu, W. Du, and D. Li, “Performance analysis and characterization of training deep learning models on mobile device,” in *2019 IEEE 25th International Conference on Parallel and Distributed Systems (ICPADS)*. IEEE, 2019, pp. 506–515.
- [38] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala, “PyTorch: An Imperative Style, High-Performance Deep Learning Library,” in *Advances in Neural Information Processing Systems 32*, H. Wallach, H. Larochelle, A. Beygelzimer, F. d’Alché Buc, E. Fox, and R. Garnett, Eds. Curran Associates, Inc., 2019, pp. 8024–8035. [Online]. Available: <http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf>
- [39] NVIDIA-AI-IOT, “torch2trt,” <https://github.com/NVIDIA-AI-IOT/torch2trt>, 2023, [Accessed: April, 2023]. [Online]. Available: <https://github.com/NVIDIA-AI-IOT/torch2trt>
- [40] P. Sinha, A. Guliani, R. Jain, B. Tran, M. D. Sinclair, and S. Venkataraman, “Not all gpus are created equal: characterizing variability in large-scale, accelerator-rich systems,” in *SC22: International Conference for High Performance Computing, Networking, Storage and Analysis*. IEEE, 2022, pp. 01–15.
- [41] F. Schroff, D. Kalenichenko, and J. Philbin, “Facenet: A unified embedding for face recognition and clustering,” in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2015, pp. 815–823.
- [42] Z. Yu, Y. Qin, X. Li, C. Zhao, Z. Lei, and G. Zhao, “Deep learning for face anti-spoofing: A survey,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 45, no. 5, pp. 5609–5631, 2023.
- [43] “Information technology – biometric presentation attack detection – part 3: Testing and reporting,” ISO/IEC, 2023, standard. [Online]. Available: <https://www.iso.org/standard/79520.html>

- [44] Z. Boulkenafet, J. Komulainen, L. Li, X. Feng, and A. Hadid, “Oulu-npu: A mobile face presentation attack database with real-world variations,” in *2017 12th IEEE international conference on automatic face & gesture recognition (FG 2017)*. IEEE, 2017, pp. 612–618.
- [45] Z. Zhang, J. Yan, S. Liu, Z. Lei, D. Yi, and S. Z. Li, “A face antispoofing database with diverse attacks,” in *2012 5th IAPR international conference on Biometrics (ICB)*. IEEE, 2012, pp. 26–31.
- [46] S. Zhang, X. Wang, A. Liu, C. Zhao, J. Wan, S. Escalera, H. Shi, Z. Wang, and S. Z. Li, “A dataset and benchmark for large-scale multi-modal face anti-spoofing,” in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2019, pp. 919–928.
- [47] P. Zhang, F. Zou, Z. Wu, N. Dai, S. Mark, M. Fu, J. Zhao, and K. Li, “Feathernets: Convolutional neural networks as light as feather for face anti-spoofing,” in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops*, 2019, pp. 0–0.
- [48] Z. Yu, C. Zhao, Z. Wang, Y. Qin, Z. Su, X. Li, F. Zhou, and G. Zhao, “Searching central difference convolutional networks for face anti-spoofing,” in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2020, pp. 5295–5305.
- [49] Z. Yu, Y. Qin, H. Zhao, X. Li, and G. Zhao, “Dual-cross central difference network for face anti-spoofing,” in *International Joint Conference on Artificial Intelligence*, 2021.
- [50] Y. Jia, J. Zhang, and S. Shan, “Dual-branch meta-learning network with distribution alignment for face anti-spoofing,” *IEEE Transactions on Information Forensics and Security*, vol. 17, pp. 138–151, 2021.
- [51] Z. Wang, Z. Wang, Z. Yu, W. Deng, J. Li, T. Gao, and Z. Wang, “Domain generalization via shuffled style assembly for face anti-spoofing,” in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2022, pp. 4123–4133.
- [52] R. Cai, Z. Li, R. Wan, H. Li, Y. Hu, and A. C. Kot, “Learning meta pattern for face anti-spoofing,” *IEEE Transactions on Information Forensics and Security*, vol. 17, pp. 1201–1213, 2022.
- [53] I. Chingovska, A. Anjos, and S. Marcel, “On the effectiveness of local binary patterns in face anti-spoofing,” in *2012 BIOSIG-proceedings of the international conference of biometrics special interest group (BIOSIG)*. IEEE, 2012, pp. 1–7.
- [54] D. Wen, H. Han, and A. K. Jain, “Face spoof detection with image distortion analysis,” *IEEE Transactions on Information Forensics and Security*, vol. 10, no. 4, pp. 746–761, 2015.
- [55] J. Deng, J. Guo, E. Ververas, I. Kotsia, and S. Zafeiriou, “Retinaface: Single-shot multi-level face localisation in the wild,” in *CVPR*, 2020.

- [56] G. B. Huang, M. Mattar, T. Berg, and E. Learned-Miller, “Labeled faces in the wild: A database for studying face recognition in unconstrained environments,” in *Workshop on faces in 'Real-Life' Images: detection, alignment, and recognition*, 2008.
- [57] S. Moschoglou, A. Papaioannou, C. Sagonas, J. Deng, I. Kotsia, and S. Zafeiriou, “Agedb: the first manually collected, in-the-wild age database,” in *proceedings of the IEEE conference on computer vision and pattern recognition workshops*, 2017, pp. 51–59.
- [58] NVIDIA Corporation, “NVIDIA JetPack SDK,” <https://developer.nvidia.com/embedded/jetpack>, 2023, accessed: 2026-01-02.