Characterizing Face Recognition for Resource Efficient Deployment on Edge

Ayan Biswas[1], Sai Amrit Patnaik[1], A. H. Abdul Hafez[2] and Anoop M. Namboodiri[1]
[1]IIIT Hyderabad, India, [2]Hasan Kalyoncu University, Gaziantep, Turkey
{ayan.biswas, sai.patnaik}@research.iiit.ac.in, abdul.hafez@hku.edu.tr, anoop@iiit.ac.in

Abstract

Deployment of Face Recognition systems on the edge has seen significant growth due to advancements in hardware design and efficient neural architectures. However, tailoring SOTA Face Recognition solutions to a specific edge device is still not easy and is vastly unexplored. Although, benchmark data is available for some combinations of model, device, and framework, it is neither comprehensive nor scalable. We propose an approximation to determine the relationship between a model and its inference time in an edge deployment scenario. Using a small number of data points, we are able to predict the throughput of custom models in an explainable manner. The prediction errors are small enough to be considered noise in observations. We also analyze which approaches are most efficient and make better use of hardware in terms of accuracy and error rates to gain a better understanding of their behaviour. Related & necessary modules such as Face Anti-Spoofing are also analyzed. To the best of our knowledge, we are the first to tackle this issue directly. The data and code along with future updates to the models and hardware will be made available at https://github.com/AyanBiswas19/Resource_Efficient_FR.

1. Introduction

Recent years have seen massive growth in edge computing & smart technologies. This has led to massive interest in being able to perform previously difficult, compute intensive deep learning tasks on resource constrained end points such as edge devices. In particular, we focus on deploying Face Recognition on Edge [25, 18, 19], which is popular as it allows systems to intelligently respond to each user. There are numerous use cases [30, 24] for such a solution. For example, the future of authentication systems may lie in Edge AI, which is not only cost-effective & convenient but also offers significant privacy benefits by retaining sensor data to an edge device rather than uploading it to a cloud server. Face Recognition (FR) is extremely popular among the various forms of authentication & identification technologies. Due to its utility, face recognition & face anti-spoofing (FAS) methods are studied extensively, both in terms of accuracy & efficiency (smaller backbones). However, tailoring the solution for edge devices is not explored. While we choose Face Recognition as a usecase for classification, the analysis in this work may be directly extended to other computer vision tasks as well.

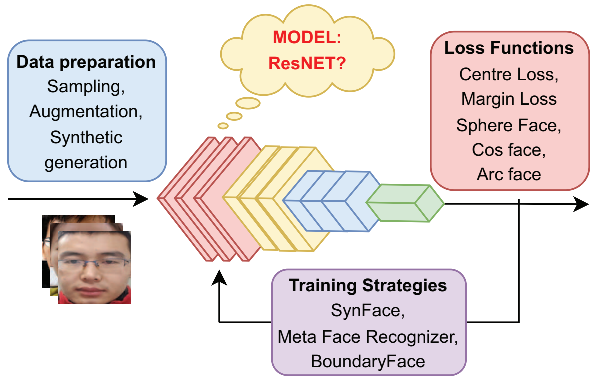

Figure 1. Methods employed to advance SOTA Face Recognition. Works involving models & architecture for Face Recognition are extremely rare. Most research is towards enhancing other components - Data, Training Strategy & Loss Functions. Tailoring these solutions to edge devices or use of backbones beyond common Object Recognition models is unexplored. Refer Table 1 for details.

As shown in Table 1, Solutions in Facial Recognition heavily draw from work done in image classification & object detection models, designed for the scale of ImageNet[6]. Further, most solutions use standard model configurations such as ResNet18 from the Resnet family [10], rather than custom or task-specific models, as used by MFR [9]. Figure 1 illustrates the current topology of SOTA Face Recognition research. Most works do not concern with the architecture or model crafting, rather it is treated as a blackbox & is barely explored in this domain. We believe this is partly due to design costs & the unavailability of benchmarks for custom architectures. In the same vein, we observe that solutions such as ShuffleNetV2 [23] & MnasNet [35] are vastly unexploited in the SOTA Face Recognition literature.

Method	Venue	Backbone
PatchNet [36]	CVPR '22	ResNet*18,34,50 MobileNetv2 ShuffleNetV2 0.5
AdaFace [17]	CVPR '22	ResNet*18, 50, 100
Magface [26]	CVPR '21	ResNet100,50
SynFace [32]	ICCV '21	LResNet50E-IR
MFR [9]	CVPR '20	28 Layer ResNet†
ElasticFace [3]	CVPR '22	ResNet100
PartialFC [1]	CVPR '22	ResNet
Zhang *et al*. [42]	ECCV '22	MobileNet
BoundaryFace [39]	ECCV '22	ResNet50
CoupleFace [22]	ECCV '22	ResNet, MobileNet
CFSM [20]	ECCV '22	ResNet50

Table 1. Architectures used by recent works in Face Recognition. Only PatchNet is a Face Anti-Spoofing solution. '*' refers to as modified by ArcFace [8], namely the removal of bottleneck structure. † - Width of 0.5

Generally, work on model architecture limits exploration to specific tasks, such as Object Classification/Detection on datasets like ImageNet [6]. These standard architectures are used as a backbone without significant modification. Few works such as MFR [9] in recognition design models that are task-specific. However, these works are less common than in FAS (Table 3). Further, even for models designed to be resource efficient, it is necessary to explore their performance on edge devices, as utilized in Facial Recognition & FAS. To this end, using custom models based on tried & tested architecture is of interest. From existing works, a model may be designed to consume the exact amount of resources demanded by the task, accuracy requirements, performance requirements & edge hardware.

At the same time, there has been rapid growth in the performance & variety of chips available for AI or Neural Network tasks. AI-Benchmark [14, 15], approaches this problem from an implementation standpoint. AI-Benchmark has detailed work on chips produced by specific manufacturers such as Qualcomm, MediaTek, & Samsung. It has discussed supported libraries, frameworks, operations, & hardware acceleration. A set of benchmarking tasks have been made, which assign a score of a device's capacity for AI tasks. Currently, they have examples of 700+ phones on their site. Similarly, MLCommons [33] provides benchmarking tools & data for various Machine Learning tasks, with heavy industry collaboration across multiple categories of devices.

Hence works such as AI-Benchmark, MLCommons provide insight on hardware & software support for deploying models to edge. Standard architectures such as [10, 11, 34, 12, 23, 35] have relatively better documentation. However, the data is sparsely distributed across combinations of models, hardware & frameworks. Further, using benchmarking data alone to obtain understanding of deploying to edge does not scale well & is infeasible. Therefore, we have proposed a mapping of custom models used in Face Recognition to their Throughput on Edge devices which uses very little data while retaining robust predictions.

Efficiency has been previously measured in terms of floating point operations (FLOPs), multiply-adds or batches/second [23] [34]. However, these metrics were chosen in an ad-hoc manner, specific to their particular novelty & task - generally ImageNet [6] classification. To address this, our work also attempts to standardize the procedure & protocols to analyze model inference on edge devices. While our work is centered on the use case of Face Recognition deployment, the framework may be extended to other domains.

We find a gap in the current literature - the mapping of architecture to the inference time of its models is not considered. To elaborate, there are extremely large no. of model configurations given an architecture. The following question may arise: Can these configurations be described using some number, say n, that maps to expected throughput/inference time relative to the standard configurations of the architecture? To the best of our knowledge, we are the first to address the problem of using benchmarks to estimate or analyze the performance of custom models.

The scope of this work is limited to deep learning solutions used in Face Recognition & related tasks such as FAS, for edge deployment. As Face Detection is identical to generic Object Detection, it is not explored in this work. Our objective is to analyze model inference; the problem of training on edge devices is outside the scope of this work. To this end, we formalize assumptions, protocols & metrics needed to describe & measure the model behavior on edge. However, there are concerns that are not related to deep learning, yet have a significant impact on model inference. One example is the data pipeline feeding input to the model, which is mostly implementation dependent - a software challenge. We address these empirically by highlighting their effect through data. Further, an analysis of how these factors interact differently based on architecture is carried out as well.

For architectures used in SOTA Face Recognition (like those in Table 1), we propose a mechanism to describe a model using its architecture. A novel analysis is done to understand the relation between a model & its inference time in an edge deployment scenario. We have been able to achieve high-quality estimates for the Throughput of custom models, from very limited data points in an explainable fashion. Further, we analyze which approaches are efficient or utilize hardware better regarding accuracy & error rates. This analysis enables decision-making for Face Recognition on edge along problems such as custom backbone design, less exploited architectures, operating across resolutions, trade-offs of throughput with accuracy, etc. We observe that FAS approaches tend to use custom models more frequently than Recognition. To this end, we have carried out a separate analysis highlighting the tradeoff of performance in terms of inference time & error rates for FAS models only.

To the best of our knowledge, this is the first attempt to map models with specific architectures to their relative throughput. We have structured the previously unstructured topology of Face Recognition for deployment in edge. Attaining these results has further required identifying measurable components, recognizing generalizable patterns (ex: backbone usage) & establishing the framework to analyse this specific domain. Our work makes the following contributions: 1) A novel analysis to understand the relation between model architecture & inference time in an edge deployment scenario for Face Recognition. 2) Accurate High-quality Throughput estimates for custom models from limited data in an explainable way. 3) Analysis of tradeoff between FAS model inference time & error rates.

To encourage an ongoing effort, a website has been prepared for this project. Newer & existing architectures, hardware & approaches would be gradually integrated. Information would be public & community contributions are welcome. It will be accessible at https://github.com/AyanBiswas19/Resource_Efficient_FR.

2. Architectural Analysis for Characterizing Models & Hardware on Edge.

In this section, we present our approach to measuring & comparing models on edge devices, the mapping of model architecture to the throughput for backbones in Table 1, & finally a separate analysis for the deployment of Face Anti-Spoofing (FAS) models.

It is necessary to formalize the characteristics of an edge device. Constraints, assumptions, deployment environment, etc affect the design choice of protocols used to measure required metrics. Section 2.1 presents this along with implementation details. Table 1 is indicative of the architecture used in SOTA Face Recognition (FR) approaches. The mapping of custom models of these architectures to throughput ($\mathcal{T}$) is discussed, in Section 2.2. We pose this as a learning problem & present a solution. However, unlike FR, solutions in FAS tend to use hand-crafted models for the task. Some of these approaches are listed in Table 3. The behaviour of backbone architectures alone is insufficient when characterizing the behaviour of $\mathcal{T}$ with respect to Error Rates in FAS models. Hence, Section 2.3 analyses the same for edge devices.

2.1. Framework Design for Analysis on Edge

We present in this section design & methodology of our experiments, for analysis of model inference on edge devices. This is done by 1. Choosing metrics & analysing their applicability, 2. Formalising modelling of the deployment & testing environment, 3. Outlining protocols to measure some of the metrics. Hence, we propose using -

Architecture $\mathcal{A}$: A family of models with similar design patterns in layers, blocks & operations. Example - ResNet18, ResNet50, etc follow ResNet architecture, MobileNetV2 (width = $x|x \in (0,1]$) follows MobileNet architecture. We use $\mathcal{A}$ to characterize models. In our experimentation, we have observed that if models are grouped by $\mathcal{A}$, then patterns against other metrics are simplified.

Throughput $(\mathcal{T}_{r,b})$: No. of inputs that can be inferred per unit time. If inputs are images & batch size $= 1$, then $\mathcal{T}$ can be measured in Frames Per Second (FPS), otherwise, Hz. r refers to the resolution. Unless specified, the default resolution is assumed to be $r = 224 \implies 224 \times 224$ resolution. b refers to batch size. Unless specified, $b = 1$. Although images at $b = 1$ (FPS) is the primary use case, face-embeddings, patches or batches per second are also relevant to the domain of Face Recognition. A similar metric is used in Section 3.1, 112×112 patches processed in parallel, per unit time.

Latency: Time taken by the network for inference of a single input. Latency $= {\mathcal{T}_{r,b=1}}^{-1}$.

Model GFLOPs $(\mathcal{F})$**:** We define $\mathcal{F}$ as no. of floating point operations (FLOPs) required for the inference of a single input, divided by 10^9. $\mathcal{F}$ varries with input size. FLOPs have been previously used to measure the complexity of a model, however, found to be an indirect approximation of metrics such as $\mathcal{T}$ [23]. In our experimentation, we find that $\mathcal{F}$ is useful if used in conjunction with $\mathcal{A}$.

No. of Parameters $(\mathcal{P})$: No. of parameters in the model. Experimentally, it has proven to be a good measure for characterizing the model, given architecture. It is preferred over $\mathcal{F}$ as it is independent of input size. Unlike $\mathcal{F}$, theoretical calculations or runtime profiling are not needed to measure $\mathcal{P}$. Ease of measurement is another advantage of $\mathcal{P}$.

We use $\mathcal{P}$ along with Architecture($\mathcal{A}$) primarily, to define a model. $\mathcal{F}$ is primarily used in drawing inferences only, due to the previously discussed limitations. $\mathcal{T}$ is favoured over Latency to estimate the speed of the model due to flexible usage & ease of comparison.

2.1.1 Deployment & Testing Environment

An edge device considered in this work is assumed to have limited compute resources - storage, memory, CPU, & GPU which are generally much slower than desktop or server counterparts & are relatively cheap in price. They are generally small, single-board computers that are deployed close to the “edge” - where sensors capture data. Examples include JetsonNano & Raspberry Pi. [21] considers Jetson TX2, a costlier (600$ in 2017), discontinued model in the

Jetson series, as a "representative mobile device". However, we find that Jetson Nano is a much better choice for a representative edge device due to its entry-level cost, availability of a standard CUDA GPU (128-core Maxwell), & general functionality. Hardware specifications are detailed in Section 3.

Factors such as capturing input from sensors, data pipeline, pre-processing, & post-processing are heavily implementation dependent. In general, the inference from the neural network is one of the most expensive steps. To this end, we time the inference/forward pass of the model. Inputs are preprocessed & loaded in memory prior to measuring inference, to mitigate the effect of the data pipeline. We do not consider post-inference operations. As an example - matching face embeddings in a database.

2.1.2 Measurement Protocol & Frameworks

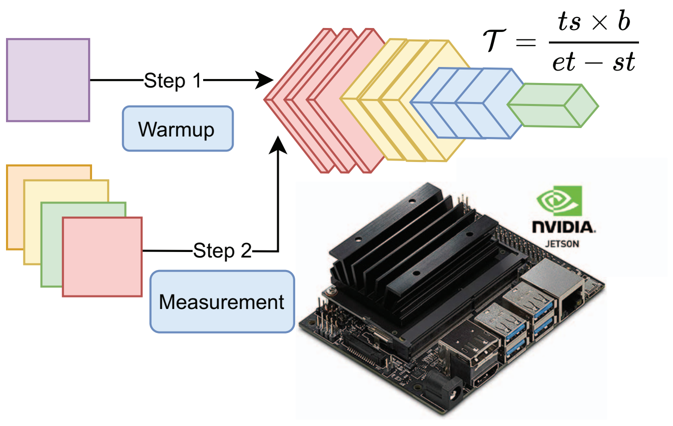

Figure 2. Deployment Environment: Nvidia Jetson Nano is the primary edge device used. Thread sync, preloading of models & data & warmup-run ensure consistent results. Process is repeated over PyTorch [31], TensorRT [28] with *FP32* & *FP16*. ts: no. of times experiment is repeated, b: Batch-Size, st, et: Start & End times.

Devices are run in headless mode in order to emulate a deployment scenario & save on excess computation, given limited resources. We measure $\mathcal{T}$ along 3 configurations - PyTorch [31], TensorRT [28], TensorRT with *FP16*. TensorRT is an SDK developed by Nvidia for inference. Tests have been carried out using Nvidia's torch2trt repository[29]. TensorRT & torch2trt have been used as they are first-party solutions on Jetson Nano - hence provide fair representation.

Figure 2 depicts our measurement protocol. It is independant of dataset - inputs are generated as random PyTorch Cuda Tensors, preloaded before timing the forward pass. Aside from the model & shape of input, ts = Trial Size, i.e no. of batches timed in the trial & batch size, are required.

2.2. Estimating the Throughput of Backbones used by SOTA Face Recognition

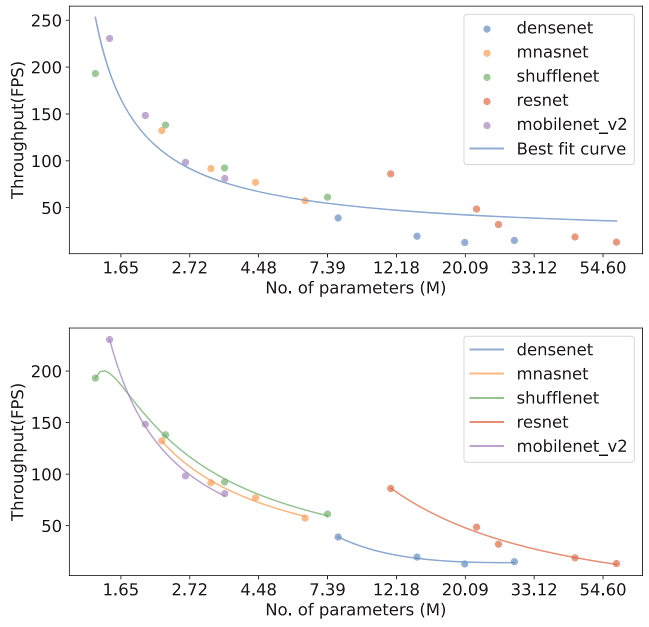

Figure 3. A plot of throughput vs standard model configurations, grouped by architecture. Points correspond to the observations. The naive approach (top) figure shows poor performance. However, the fit is very accurate when grouped by architecture (bottom). Mean absolute error reduces by 84%. Note: X-axis is on log scale.

In this section, we formulate a mapping from models of an architecture $(\mathcal{A})$ to their throughput $(\mathcal{T})$. We define the data-set used to train/fit the estimator function as Landmarks (L). L is device specific throughput data of models, typically of standard configurations such as ResNet18 of ResNet architecture. Consider -

$$\mathcal{T}(M) = Z_L(\mathcal{A}, \mathcal{P}) + \epsilon \tag{1}$$

where $\mathcal{T}(M)$ is $\mathcal{T}$ of Model M & ϵ is noise. $Z_L(M)$ is a function which estimates $\mathcal{T}$.

Ideally, M should be an ordered set of parameterized operations/architectural blocks, for ex: Convolutions, Residual Blocks, Fully Connected Layers, etc. However, it is difficult to get data across various configurations of $\mathcal{A}$, across devices & libraries. Therefore, it is necessary for Z to be computed from very limited L. Hence we substitute the ordered set of operations using $\mathcal{A}$ such that it A takes values {ResNet, MobileNet, DenseNet, ...}. We find that no. of parameters$(\mathcal{P})$ along with $\mathcal{A}$ captures sufficient information to predict the relative $\mathcal{T}$. Therefore, Z_L as $Z_L(M) = Z_L(\mathcal{A}, \mathcal{P})$.

Fig 3 (top) shows the distribution of $\mathcal{T}$ against $\mathcal{P}$. As model size increases with $\mathcal{P}$, there is an expected drop in $\mathcal{T}$. As $\mathcal{T}$ is inversely proportional to model size, a trend similar to $y = c/x$ is expected. But, we find this hyperbolic fit to show very poor approximations. However, when grouping points by $\mathcal{A}$, we are able to achieve fairly accurate predictions. Fig. 3 (bottom) shows the plot of Z_L, against $\mathcal{P}$, grouped by $\mathcal{A}$. Z_L is obtained by polynomial regression on $\mathcal{P}$, grouped by A. Hence,

$$Z_L(\mathcal{A}, \mathcal{P}) = \frac{C_1}{\mathcal{P}^2} + \frac{C_2}{\mathcal{P}} + I \tag{2}$$

where C_1 & C_2 are coeffiecints obtained from regression, I is the intercept. C_1, C_2 an I are computed separately for each $\mathcal{A}$.

Mean Absolute Error in $\mathcal{T}$ is reduced by 84% compared to Fig. 3 (top). Further, we verify the hypothesis through interpolation of non-standard configurations. Detailed results can be found in Table 2.

For testing, we use the following custom models: MobileNetV2 with varying widths, & ResNets of varying depths. The mean absolute errors are 3.4 FPS for MobileNetV2 & 3.08 for ResNets. When looking at individual points, the error in prediction is low enough to be considered noise in observations. Hence, we argue that the proposed framework of estimating T through the estimator function Z_L is reliable with limited data & does not overfit. Further, Z_L which estimates $\mathcal{T}$ is formulated such that L is used as training data, & $(\mathcal{A}, \mathcal{P})$ as features. Hence the mapping of custom models of an Architecture to their Throughput on a device is posed as a learning problem.

2.3. Deployability of FAS Models

Compared to Face Recognition approaches, as shown in Table 1, Anti-Spoofing approaches (Table 3) tend to use models handcrafted for the task. While the design of task-specific models is desired, we observe significant variation in Throughput ($\mathcal{T}$), for similar no. of parameters ($\mathcal{P}$) & Half Total Error Rate (HTER), a common metric used to measure error in FAS. Broadly, two observations of note are 1. Architectures making use of modified Convolution operations are underperforming 2. There is an odd distribution of error rates with respect to $\mathcal{T}$, wherein $\mathcal{T}$ is higher when the error is low, as observed in SSAN-R.

CDCN [41] & DC-CDN [40] models adopt an enhancement over the standard 2D Convolution operation, Central Difference Convolution (CDC). While these have shown improvements, it is important to note that these non-standard operations do not seem to be supported to the same extent as regular ones. These models suffer from extremely low $\mathcal{T}$. DC-CDN propose C-CDC convolution based on the CDC, which exploits sparse local features to improve the efficiency of the CDC, in theory. From claimed ACER values on Oulu-NPU [2] Protocol 1, DC-CDNs perform slightly worse than CDCNs. Further, no benefit in $\mathcal{T}$ is observed. However, as the deficiency in $\mathcal{T}$ is likely to be an issue of hardware/library implementations, the performance of CDC-based architectures in terms of $\mathcal{T}$ may not have been properly realised.

On the other hand, both SSAN [37] models are outliers in terms of $\mathcal{T}$ to HTER tradeoff. For some reason, SSAN-M [37] has extremely poor $\mathcal{T}$ despite having higher HTER, as compared to SSAN-R[37]. While it may be an implementation or hardware issue, the implementation used was official, to the best of our knowledge. There is a significant gap in $\mathcal{T}$ between SSAN-R & HierarchialFusionNetwork [4] (HFN). However, it is inconclusive as the model size increases at a faster rate when accuracy or errors saturate. SSAN-R shows significant improvement over DBM-Net [16]. On further investigation, SSAN-R has slightly better performance in terms of HTER, across all cross-database testing results. In conclusion, this study highlights that it is not possible to group Accuracy/Error Rates against $\mathcal{T}$, similar to Fig 6. Further, there are gaps in implementation for certain operations. This makes deploying potentially better-performing approaches infeasible.

Our analysis has been limited by the availability of implementations of works in this field. However, based on our experimentation, SSAN-R provides a good trade-off between error rates & $\mathcal{T}$.

3. Experimental Details

The experimentation work presented in this section, unless otherwise stated, is done on the Nvidia Jetson Nano 4GB. It has a 128-core Maxwell GPU, Quad-core ARM A57 @ 1.43 GHz processor & 4GB LPDDR4 shared memory. We consider it as a representative edge device, as argued in Section 2.1.1. As previously stated, experiments are run using PyTorch [31], TensorRT [28] & TensorRT at 16-bit Floating Point Precision (*FP16*).

3.1. Variation Across Resolutions

Resolution (r) is a key design component when deploying neural networks on edge devices. Requirements vary based on the task. For example, Face Detection in a crowded street would benefit from high r. Patch-based approaches such as PatchNet [36] generally operate at lower r. Hence, we analyze the relation between throughput ($\mathcal{T}$) & r, for standard backbones (Table 1). Table 4 shows the highest r achieved in our experimentation using TensorRT with *FP16*. For the sake of clarity, we define $r = k$ as $k \times k$ pixels. High r was possible only on the lightweight ResNet18 & MobileNetV2 models. Through RetinaFace [7], we verify that loss in $\mathcal{T}$ due to detection overhead is relatively low, evidenced by RetinaFace Mn25 - MobileNetV2 0.25 & Re50 - ResNet50.

Fig. 4 (left) shows the drop off of $\mathcal{T}$ against r. The increase in computation in Model GFLOPs ($\mathcal{F}$) is linear to no. of pixels. Therefore, a hyperbolic trend similar to

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
	Mean Absolute	1.07			Mean Absolute	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
	Mean Absolute	2.08			Mean Absolute	2.24

Table 2. Train & Test errors of the proposed estimation method. Absolute errors may be measured in **inferences/s**. Test models are custom models made from the same architecture. α refers to the width multiplier, as proposed in [11]. B and N denote the type of blocks used to make custom ResNet models - Basic Residual Block, Bottleneck Block.

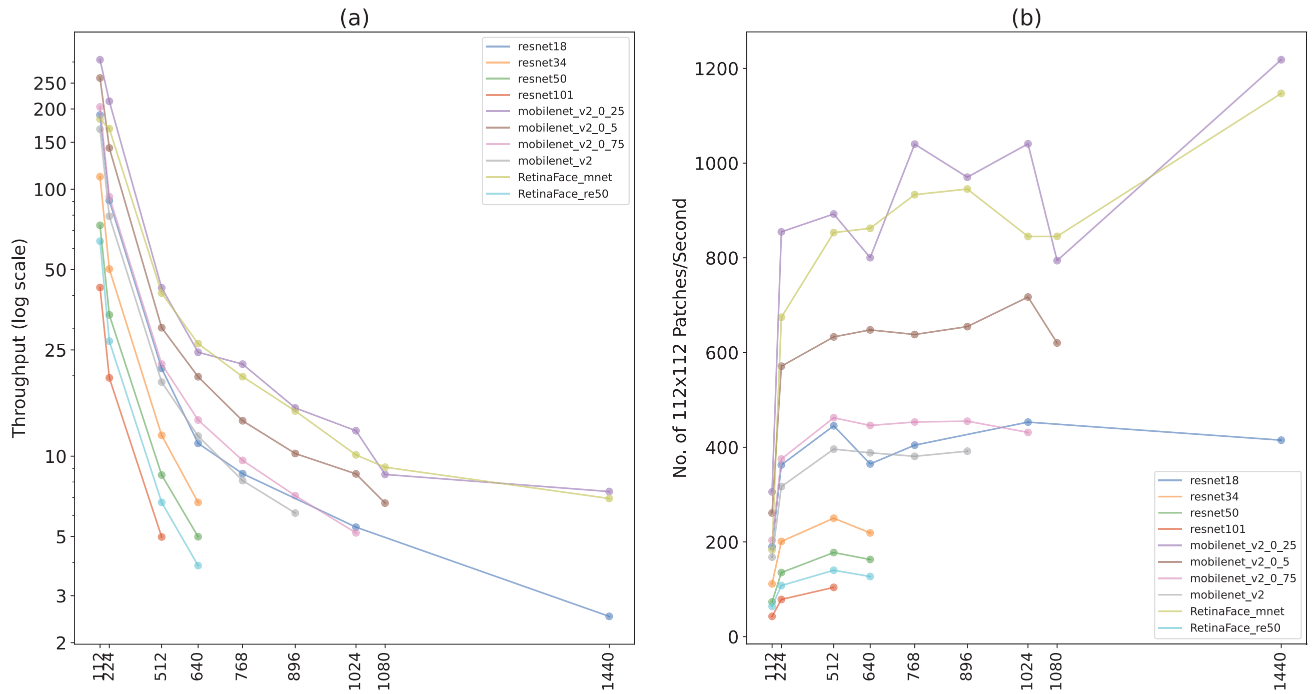

Figure 4. Left: Throughput ($\mathcal{T}$) (log scale) against resolution r on the X axis. $r = i \implies i \times i$ pixels. Right: 112×112 patches/second against r. This is not to be confused with batch size. No. of patches is used to estimate size of input processed per unit time, rather than no. of parallel inputs.

that explored in Section 2.2 is expected, but it proves to be incorrect. This is due to $\mathcal{T}$ being "clamped" at higher $\mathcal{T}$ or lower r. As expected, $\mathcal{T}$ is maximum at the minimum r of 112. However, although the computation cost $\mathcal{F}$ is linear in r, the drop-off in $\mathcal{T}$ is much faster. We believe this is due to the cost of other operations such as data pipeline, switching between runtimes for I/O, etc. Modelling of this overhead is beyond the scope of this work. However, let us consider "Pixels Processed / Time". In this case, Fig 4 shows the no.of $r = 112$ patches processed in parallel, as a function of resolution. We empirically prove the presence of this bottleneck as we see a large jump from r of 112 to 224, in

Model Name	$\mathcal{T}$	$\mathcal{P}$	HTER	ACER
FeatherNetA [43]	$213.5^\ddagger$	0.35	-	-
FeatherNetB [43]	$187.9^\ddagger$	0.35	-	-
CDCN* [41]	$2.5^\ddagger$	2.25	-	1
CDCNpp* [41]	$2.3^\ddagger$	2.26	-	0.2
C-CDN HV*[40]	$2.5^\ddagger$	1.25	-	0.6
C-CDN DG*[40]	$2.4^\ddagger$	1.25	-	0.7
DC-CDN* [40]	$1.2^\ddagger$	2.50	18.82	0.4
DBMNet [16]	$31.8^\dagger$	12.44	17.59	-
SSAN-M [37]	$2.5^\dagger$	8.79	19.51	-
SSAN-R [37]	$42.8^\dagger$	8.14	13.72	-
HFN* [4]	$12.5^\dagger$	62.12	12.4	-

Table 3. Performance analysis of some recent FAS solutions. * - Approximated Throughput $\mathcal{T}$, as model did not run on TensorRT. $\mathcal{P}$ - No. of Parameters. † - *FP16*, ‡ - *FP32*. HTER - computed from training on Replay attack [5], CASIA-MFSD [45], MSU-MFSD [38], testing on OULU-NPU [2]. ACER - on Protocol 1 of OULU-NPU. FeatherNets have accuracy data for CASIA SURF [44] only, leading to inconclusive comparisons with rest of the approaches.

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 4. Maximum Resolutions achieved in experimentation, using TensorRT with *FP16* Precision. Beyond these resolutions, we experience random crashes, & the process is killed. RetinaFace [7] is paired with Mn25 = MobileNetV2 0.25 & Re50 = ResNet50 backbones.

terms of no. of patches processsed, across all models.

Hence performance saturates at lower r, i.e. the rate of increase in $\mathcal{T}$ falls. It is preferable to avoid large r unless there is some specific benefit, due to high memory & resource consumption. This would make running other processes on devices harder, in deployment. However, it is interesting to note that high r proves to be better in terms of resource utilisation.

3.2. Impact of Floating Point Precision, Data Pipeline & Memory

Converting models from standard 32-bit Floating Point Precision (*FP32*) to 16-bit Precision (*FP16*) is standard practice for deploying models on limited resource environments. This is not a training time change. Indeed, pretrained models can be converted. We investigate in this

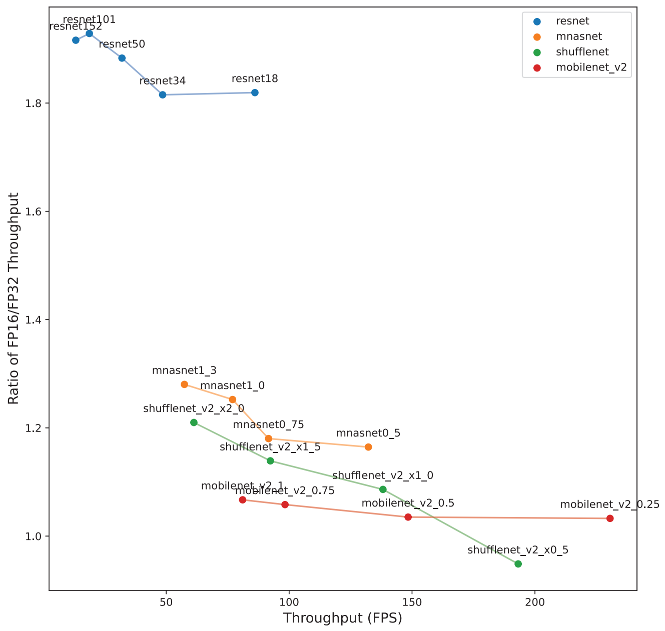

Figure 5. Distribution of the Ratio of Throughput on *FP32* to *FP16*, on TensorRT.

section which architectures have benefited the most from *FP16* conversion. However, both hardware & library need to support the use of fast *FP16* operations like TensorRT to observe the benefit.

Figure 5 plots the ratio R of the throughput $\mathcal{T}$ values of *FP16* to the regular *FP32* of TensorRT, while the throughput $\mathcal{T}$ is shown on X-axis. It can be observed that there is a clear separation of ResNets, for which R is higher from MobileNetV2, ShuffleNet, & MnasNet. This indicates that models designed for regular GPUs benefit significantly more from the reduction in memory footprint & parallelization due to *FP16*. Models designed to be resource-efficient show speedups as well. However, there are some outliers. MobileNetV2 seems to be very stable across $\mathcal{T}$ & $\mathcal{P}$. However, R being close to 1 indicates a limited speedup. Some models such as FeatherNetA, FeatherNetB (Table 3) & ShuffleNetV2 0.5x in Fig. 5 have $R < 1$, i.e they lose $\mathcal{T}$ when converting to *FP16*. While these observations are rare & may be specific to implementation, a common factor is that these models are designed to be lightweight & have high $\mathcal{T}$. It can be hypothesized, for some lightweight models, either hardware is unable to maintain efficiency when computation is reduced due to *FP16*, or the implementation is unable to support it.

3.2.1 Data Pipeline & Memory Bottleneck

As evidenced earlier in Section 3.1, there are other processes which consume or block resources, causing a bottleneck specifically at high $\mathcal{T}$. As an example, MobileNetV2 (width = 0.25) is able to operate at $\mathcal{T} = 7.37$, at 1440×1440

resolution, i.e, over 1200 patches of size 112×112 per second. At 112×112 resolution, the $\mathcal{T}$ achieved is merely 300 patches. Despite preloading inputs into memory, a massive bottleneck is observed in this experiment, as 300 inputs (tensors) have to be fed to the model, per second at $r = 120$ rather than 7 at $r = 1440$. Therefore, a person interested in deploying models on edge should consider performance across resolutions to estimate the impact of the data pipeline, & what could be a saturation point in optimization.

3.3. Accuracy vs Performance

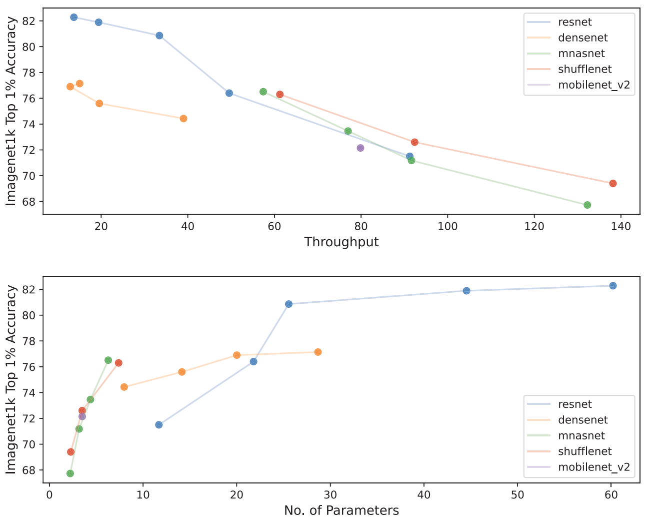

Figure 6. Top: Accuracy against Throughput, Bottom: Accuracy against No. of Parameters

We present in this section our analysis of the accuracy of models with respect to the performance, represented by the throughput $\mathcal{T}$, shown in Figure 6 (top). The analysis is also presented with respect to the model size represented by the number of parameters $\mathcal{P}$, shown in Figure 6 (bottom).

As it can be observed in Figure 1, most of the works on Face Recognition such as [17, 9, 32] try improving the performance by, for example, investigating the loss functions, training framework, data augmentation techniques, & data sampling. These are independent of the backbone models which are arbitrarily chosen. Furthermore, the accuracy margins reported by SOTA methods are very slim, often at 96 to 99% with $< 1\%$ variation on popular datasets such as LFW [13], AgeDB [27] etc. As such, it is hard to decouple the strength of the backbone from the approach, through the accuracy data of these works.

Hence, to estimate the "discriminating power" of the backbone model in hand, we have considered their Top 1 - Accuracy on Imagenet [6]. Top 1 Accuracy (acc) is considered over Top 5 due to simplicity & larger variation in scores. Further, all the backbone models have thorough experimentation on ImageNet. Aside from significantly larger models ResNets101 & ResNet152, only ResNet50 exceeds 80% acc. As acc gains are meagre compared to the drop in $\mathcal{T}$ for larger models, ResNet50 can be a very balanced choice at $\mathcal{T} = 33$. The ShuffleNetV2 models provide on average the highest accuracy at a given $\mathcal{T}$. The leftmost MnasNet model is on par with the leftmost ShuffleNetV2 model in Fig. 6 (Top), but the remaining other configurations perform worse.

However, another factor needs to be considered in the discussion of acc, no. of parameters $(\mathcal{P})$ which is related to model size, shown in Fig. 6 (bottom). MobileNetV2 & Mnasnet0_3 are close to ResNet18 in acc but far apart in $\mathcal{P}$, as shown in Fig. 6 (bottom). This allows us to design a trade-off between acc & $\mathcal{P}$, at constant $\mathcal{T}$. $\mathcal{P}$ is important when building applications to be run on edge devices with limited resources. This is of great use in situations where the size of the model has to be optimized, to run on devices with limited or slow storage, etc. However, $\mathcal{P}$ is insufficient to describe memory footprint at run time. As an example, The models Densenet121, & Densenet169 hugely outperform ResNet18 in acc to $\mathcal{P}$ ratio, but require significantly more memory access operations at runtime. However, ShuffleNetV2, MnasNet & MobileNetV2 architectures are expected to perform better on devices with lesser resources than Jetson Nano, as their $\mathcal{P}$ is low due to explicit design of memory efficiency.

4. Conclusion

We standardize inference on edge devices by analyzing the time & performance in deployment scenarios. We are the first to have created a mapping that estimates throughput based on the model's architecture. This has been done by structuring the unstructured topology of Face Recognition for deployment on Edge. The predictions obtained are accurate enough to be considered as noise in observations & require a very small no. of data points. Further, related challenges such as FAS are analyzed in detail. Deployment concerns such as impact of floating point precision, data pipeline, & memory operations on performance, as well as the accuracy of models in relation to their size have been emperically analyzed. The scale of the generic problem exceeds the scope of a single study. Something as simple as, performance variation among multiple units of the same device poses a challenge.

The data & code of the work will be available at https://github.com/AyanBiswas19/Resource_Efficient_FR. We hope this acts as a resource for hardware developers for furthering Face Recognition & Edge AI. While the work focuses on Face Recognition & FAS, the study is readily extensible to other vision based inference tasks on the edge.

References

[1] Xiang An, Jiankang Deng, Jia Guo, Ziyong Feng, XuHan Zhu, Jing Yang, and Tongliang 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)*, pages 4042–4051, June 2022. 2

[2] Zinelabinde Boulkenafet, Jukka Komulainen, Lei Li, Xiaoyi Feng, and Abdenour 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)*, pages 612–618. IEEE, 2017. 5, 7

[3] Fadi Boutros, Naser Damer, Florian Kirchbuchner, and Arjan Kuijper. Elasticface: Elastic margin loss for deep face recognition. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR) Workshops*, pages 1578–1587, June 2022. 2

[4] Rizhao Cai, Zhi Li, Renjie Wan, Haoliang Li, Yongjian Hu, and Alex C Kot. Learning meta pattern for face anti-spoofing. *IEEE Transactions on Information Forensics and Security*, 17:1201–1213, 2022. 5, 7

[5] Ivana Chingovska, André Anjos, and Sébastien 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)*, pages 1–7. IEEE, 2012. 7

[6] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*, pages 248–255. Ieee, 2009. 1, 2, 8

[7] Jiankang Deng, Jia Guo, Evangelos Ververas, Irene Kotsia, and Stefanos Zafeiriou. Retinaface: Single-shot multi-level face localisation in the wild. In *CVPR*, 2020. 5, 7

[8] Jiankang Deng, Jia Guo, Niannan Xue, and Stefanos Zafeiriou. Arcface: Additive angular margin loss for deep face recognition. In *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 4685–4694, 2019. 2

[9] Jianzhu Guo, Xiangyu Zhu, Chenxu Zhao, Dong Cao, Zhen Lei, and Stan 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. 1, 2, 8

[10] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016. 1, 2

[11] Andrew G Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *arXiv preprint arXiv:1704.04861*, 2017. 2, 6

[12] Gao Huang, Zhuang Liu, Laurens Van Der Maaten, and Kilian Q Weinberger. Densely connected convolutional networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4700–4708, 2017. 2

[13] Gary B Huang, Marwan Mattar, Tamara Berg, and Eric Learned-Miller. Labeled faces in the wild: A database forstudying face recognition in unconstrained environments. In *Workshop on faces in'Real-Life'Images: detection, alignment, and recognition*, 2008. 8

[14] Andrey Ignatov, Radu Timofte, William Chou, Ke Wang, Max Wu, Tim Hartley, and Luc Van Gool. Ai benchmark: Running deep neural networks on android smartphones. In *Proceedings of the European Conference on Computer Vision (ECCV) Workshops*, pages 0–0, 2018. 2

[15] Andrey Ignatov, Radu Timofte, Andrei Kulik, Seungsoo Yang, Ke Wang, Felix Baum, Max Wu, Lirong Xu, and Luc Van Gool. Ai benchmark: All about deep learning on smartphones in 2019. In *2019 IEEE/CVF International Conference on Computer Vision Workshop (ICCVW)*, pages 3617–3635. IEEE, 2019. 2

[16] Yunpei Jia, Jie Zhang, and Shiguang Shan. Dual-branch meta-learning network with distribution alignment for face anti-spoofing. *IEEE Transactions on Information Forensics and Security*, 17:138–151, 2021. 5, 7

[17] Minchul Kim, Anil K Jain, and Xiaoming Liu. Adaface: Quality adaptive margin for face recognition. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 18750–18759, 2022. 2, 8

[18] Anis Koubaa, Adel Ammar, Anas Kanhouch, and Yasser Al-Habashi. Cloud versus edge deployment strategies of real-time face recognition inference. *IEEE Transactions on Network Science and Engineering*, 9(1):143–160, 2021. 1

[19] Shen Li, Fang Liu, Jiayue Liang, Zhenhua Cai, and Zhiyao Liang. Optimization of face recognition system based on azure iot edge. *Computers, Materials & Continua*, 61(3), 2019. 1

[20] Feng Liu, Minchul Kim, Anil Jain, and Xiaoming Liu. Controllable and guided face synthesis for unconstrained face recognition. In *ECCV*, 2022. 2

[21] Jie Liu, Jiawen Liu, Wan Du, and Dong 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)*, pages 506–515. IEEE, 2019. 3

[22] Jiaheng Liu, Haoyu Qin, Yichao Wu, Jinyang Guo, Ding Liang, and Ke Xu. Coupleface: Relation matters for face recognition distillation, 2022. 2

[23] Ningning Ma, Xiangyu Zhang, Hai-Tao Zheng, and Jian Sun. Shufflenet v2: Practical guidelines for efficient cnn architecture design. In *Proceedings of the European Conference on Computer Vision (ECCV)*, September 2018. 1, 2, 3

[24] Sumaira Manzoor, Eun-Jin Kim, Sung-Hyeon Joo, Sang-Hyeon Bae, Gun-Gyo In, Kyeong-Jin Joo, Jun-Hyeon Choi, and Tae-Yong Kuc. Edge deployment framework of guardbot for optimized face mask recognition with real-time inference using deep learning. *Ieee Access*, 10:77898–77921, 2022. 1

[25] Yunlong Mao, Shanhe Yi, Qun Li, Jinghao Feng, Fengyuan Xu, and Sheng Zhong. A privacy-preserving deep learning approach for face recognition with edge computing. In *Proc. USENIX Workshop Hot Topics Edge Comput.(HotEdge)*, pages 1–6, 2018. 1

[26] Qiang Meng, Shichao Zhao, Zhida Huang, and Feng 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)*, pages 14225–14234, June 2021. 2

[27] Stylianos Moschoglou, Athanasios Papaioannou, Christos Sagonas, Jiankang Deng, Irene Kotsia, and Stefanos Zafeiriou. Agedb: the first manually collected, in-the-wild age database. In *proceedings of the IEEE conference on computer vision and pattern recognition workshops*, pages 51–59, 2017. 8

[28] Nvidia. NVIDIA TensorRT. https://developer.nvidia.com/tensorrt, 2023. [Accessed: April, 2023]. 4, 5

[29] NVIDIA-AI-IOT. torch2trt. https://github.com/NVIDIA-AI-IOT/torch2trt, 2023. [Accessed: April, 2023]. 4

[30] Manoranjan Parhi, Abhinandan Roul, Bravish Ghosh, and Abhilash Pati. Ioats: An intelligent online attendance tracking system based on facial recognition and edge computing. *International Journal of Intelligent Systems and Applications in Engineering*, 10(2):252–259, 2022. 1

[31] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. PyTorch: An Imperative Style, High-Performance Deep Learning Library. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc., 2019. 4, 5

[32] Haibo Qiu, Baosheng Yu, Dihong Gong, Zhifeng Li, Wei Liu, and Dacheng Tao. Synface: Face recognition with synthetic data. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 10880–10890, October 2021. 2, 8

[33] Vijay Janapa Reddi, Christine Cheng, David Kanter, Peter Mattson, Guenther Schmuelling, Carole-Jean Wu, Brian Anderson, Maximilien Breughe, Mark Charlebois, William Chou, et al. Mlperf inference benchmark. In *2020 ACM/IEEE 47th Annual International Symposium on Computer Architecture (ISCA)*, pages 446–459. IEEE, 2020. 2

[34] Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. Mobilenetv2: Inverted residuals and linear bottlenecks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4510–4520, 2018. 2

[35] Mingxing Tan, Bo Chen, Ruoming Pang, Vijay Vasudevan, Mark Sandler, Andrew Howard, and Quoc V Le. Mnasnet: Platform-aware neural architecture search for mobile. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 2820–2828, 2019. 1, 2

[36] Chien-Yi Wang, Yu-Ding Lu, Shang-Ta Yang, and Shang-Hong 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*, pages 20281–20290, 2022. 2, 5

[37] Zhuo Wang, Zezheng Wang, Zitong Yu, Weihong Deng, Jiahong Li, Tingting Gao, and Zhongyuan Wang. Domain generalization via shuffled style assembly for face anti-spoofing. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 4123–4133, 2022. 5, 7

[38] Di Wen, Hu Han, and Anil K Jain. Face spoof detection with image distortion analysis. *IEEE Transactions on Information Forensics and Security*, 10(4):746–761, 2015. 7

[39] Shijie Wu and Xun Gong. Boundaryface: A mining framework with noise label self-correction for face recognition, 2022. 2

[40] Zitong Yu, Yunxiao Qin, Hengshuang Zhao, Xiaobai Li, and Guoying Zhao. Dual-cross central difference network for face anti-spoofing. In *International Joint Conference on Artificial Intelligence*, 2021. 5, 7

[41] Zitong Yu, Chenxu Zhao, Zezheng Wang, Yunxiao Qin, Zhuo Su, Xiaobai Li, Feng Zhou, and Guoying Zhao. Searching central difference convolutional networks for face anti-spoofing. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 5295–5305, 2020. 5, 7

[42] Manyuan Zhang, Guanglu Song, Yu Liu, and Hongsheng 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*, page 720–736, Berlin, Heidelberg, 2022. Springer-Verlag. 2

[43] Peng Zhang, Fuhao Zou, Zhiwen Wu, Nengli Dai, Skarpness Mark, Michael Fu, Juan Zhao, and Kai 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*, pages 0–0, 2019. 7

[44] Shifeng Zhang, Xiaobo Wang, Ajian Liu, Chenxu Zhao, Jun Wan, Sergio Escalera, Hailin Shi, Zezheng Wang, and Stan 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*, pages 919–928, 2019. 7

[45] Zhiwei Zhang, Junjie Yan, Sifei Liu, Zhen Lei, Dong Yi, and Stan Z Li. A face antispoofing database with diverse attacks. In *2012 5th IAPR international conference on Biometrics (ICB)*, pages 26–31. IEEE, 2012. 7