Characterizing Face Recognition for Resource Efficient Deployment on Edge

Ayan Biswas¹, Sai Amrit Patnaik¹, A. H. Abdul Hafez² and Anoop M. Namboodiri¹
¹IIIT Hyderabad, India, ²Hasan 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.

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.
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 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.

<table>
<thead>
<tr>
<th>Method</th>
<th>Venue</th>
<th>Backbone</th>
</tr>
</thead>
<tbody>
<tr>
<td>PatchNet [36]</td>
<td>CVPR ’22</td>
<td>ResNet*18,34,50</td>
</tr>
<tr>
<td>AdaFace [17]</td>
<td>CVPR ’22</td>
<td>ResNet100,50</td>
</tr>
<tr>
<td>MagFace [26]</td>
<td>ICCV ’21</td>
<td>LResNet50E-IR</td>
</tr>
<tr>
<td>SynFace [32]</td>
<td>CVPR ’20</td>
<td>28 Layer ResNet†</td>
</tr>
<tr>
<td>MFR [9]</td>
<td>CVPR ’22</td>
<td>ResNet100</td>
</tr>
<tr>
<td>PartialFC [1]</td>
<td>CVPR ’22</td>
<td>ResNet50</td>
</tr>
<tr>
<td>Zhang et al. [42]</td>
<td>ECCV ’22</td>
<td>MobileNet</td>
</tr>
<tr>
<td>BoundaryFace [39]</td>
<td>ECCV ’22</td>
<td>ResNet50</td>
</tr>
<tr>
<td>CoupleFace [22]</td>
<td>ECCV ’22</td>
<td>ResNet, MobileNet</td>
</tr>
<tr>
<td>CFSSM [20]</td>
<td>ECCV ’22</td>
<td>ResNet50</td>
</tr>
</tbody>
</table>

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
tions, 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.


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 ($T$) 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 $T$ 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 $A$:** 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 $A$ to characterize models. In our experimentation, we have observed that if models are grouped by $A$, then patterns against other metrics are simplified.

**Throughput ($T_{r,b}$):** No. of inputs that can be inferred per unit time. If inputs are images & batch size $= 1$, then $T$ can be measured in Frames Per Second (FPS), otherwise, Hz. $r$ refers to the resolution. Unless specified, the default resolution is assumed to be $r = 224 \Rightarrow 224 \times 224$ 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, $112 \times 112$ patches processed in parallel, per unit time.

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

**Model GFLOPs ($F$):** We define $F$ as no. of floating point operations (FLOPs) required for the inference of a single input, divided by $10^9$. $F$ varies 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 $T$ [23]. In our experimentation, we find that $F$ is useful if used in conjunction with $A$.

**No. of Parameters ($P$):** 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 $F$ as it is independent of input size. Unlike $F$, theoretical calculations or runtime profiling are not needed to measure $P$. Ease of measurement is another advantage of $P$.

We use $P$ along with Architecture ($A$) primarily, to define a model. $F$ is primarily used in drawing inferences only, due to the previously discussed limitations. $T$ 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 $T$ 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 independent of dataset - inputs are generated as random PyTorch Cuda Tensors, preloaded before timing the forward pass. Aside from the model & shape of input, $ts = \text{Trial Size}$, 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 (top) shows the distribution of $T$ against $P$. As model size increases with $P$, there is an expected drop in $T$. As $T$ is inversely proportional to model size, a trend

\[
T(M) = Z_{L}(A, P) + \epsilon \tag{1}
\]

where $T(M)$ is $T$ of Model $M$ & $\epsilon$ is noise. $Z_{L}(M)$ is a function which estimates $T$.

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 $A$, 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 $A$ such that it takes values $\{\text{ResNet, MobileNet, DenseNet, ...}\}$. We find that no. of parameters($P$) along with $A$ captures sufficient information to predict the relative $T$. Therefore, $Z_{L}$ as $Z_{L}(M) = Z_{L}(A, P)$.

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

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

where \( C_1 \) & \( C_2 \) are coefficients obtained from regression, \( I \) is the intercept. \( C_1, C_2 \) an \( I \) are computed separately for each \( A \).

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 \( T \) is formulated such that \( L \) is used as training data, \& \( (A, \mathcal{P}) \) 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 & HierarchicalFusionNetwork [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 DBMNet [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 \( r = k \times k \) 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 Mm25 - 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
<table>
<thead>
<tr>
<th>MobileNetV2, Width</th>
<th>Predicted</th>
<th>Actual</th>
<th>MobileNetV2, Width</th>
<th>Predicted</th>
<th>Actual</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \alpha = 0.25 )</td>
<td>230.92</td>
<td>230.54</td>
<td>( \alpha = 0.2 )</td>
<td>245.39</td>
<td>239.46</td>
</tr>
<tr>
<td>( \alpha = 0.5 )</td>
<td>147.03</td>
<td>148.41</td>
<td>( \alpha = 0.6 )</td>
<td>123.45</td>
<td>116.82</td>
</tr>
<tr>
<td>( \alpha = 0.75 )</td>
<td>100.10</td>
<td>98.34</td>
<td>( \alpha = 0.8 )</td>
<td>94.57</td>
<td>95.17</td>
</tr>
<tr>
<td>( \alpha = 1 )</td>
<td>80.33</td>
<td>81.09</td>
<td>( \alpha = 0.9 )</td>
<td>85.87</td>
<td>85.14</td>
</tr>
<tr>
<td>Train Error</td>
<td>RMSE</td>
<td>1.20</td>
<td>Test Error</td>
<td>RMSE</td>
<td>4.47</td>
</tr>
<tr>
<td>Mean Absolute</td>
<td>1.07</td>
<td></td>
<td>Mean Absolute</td>
<td>3.47</td>
<td></td>
</tr>
</tbody>
</table>

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. \( \alpha \) 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](image)

Figure 4. Left: Throughput \((T)\) (log scale) against resolution \( r \) on the X axis. \( r = i \Rightarrow i \times i \) pixels. Right: \( 112 \times 112 \) 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 \( T \) being “clamped” at higher \( T \) or lower \( r \). As expected, \( T \) is maximum at the minimum \( r \) of 112. However, although the computation cost \( F \) is linear in \( r \), the drop-off in \( T \) 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
terms of no. of patches processed, across all models.

Hence performance saturates at lower \( r \), i.e. the rate of increase in \( T \) 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, pre-trained models can be converted. We investigate in this 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 \( T \) values of FP16 to the regular FP32 of TensorRT, while the throughput \( T \) 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 \( T \) & \( P \). 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 \( T \) 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 \( T \). 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 \( T \). As an example, MobileNetV2 (width = 0.25) is able to operate at \( T = 7.37 \), at 1440×1440

<table>
<thead>
<tr>
<th>Model Name</th>
<th>( T )</th>
<th>( P )</th>
<th>HTER</th>
<th>ACER</th>
</tr>
</thead>
<tbody>
<tr>
<td>FeatherNetA [43]</td>
<td>213.5†</td>
<td>0.35</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>FeatherNetB [43]</td>
<td>187.9†</td>
<td>0.35</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>CDCN* [41]</td>
<td>2.5‡</td>
<td>2.25</td>
<td>-</td>
<td>1</td>
</tr>
<tr>
<td>CDCNpp* [41]</td>
<td>2.3‡</td>
<td>2.26</td>
<td>-</td>
<td>0.2</td>
</tr>
<tr>
<td>C-CDN HV*[40]</td>
<td>2.5‡</td>
<td>1.25</td>
<td>-</td>
<td>0.6</td>
</tr>
<tr>
<td>C-CDN DG*[40]</td>
<td>2.4‡</td>
<td>1.25</td>
<td>-</td>
<td>0.7</td>
</tr>
<tr>
<td>DC-CDN* [40]</td>
<td>1.2‡</td>
<td>2.50</td>
<td>18.82</td>
<td>0.4</td>
</tr>
<tr>
<td>DBMNet [16]</td>
<td>31.8†</td>
<td>12.44</td>
<td>17.59</td>
<td>-</td>
</tr>
<tr>
<td>SSAN-M [37]</td>
<td>2.5‡</td>
<td>8.79</td>
<td>19.51</td>
<td>-</td>
</tr>
<tr>
<td>SSAN-R [37]</td>
<td>42.8†</td>
<td>8.14</td>
<td>13.72</td>
<td>-</td>
</tr>
<tr>
<td>HFN* [4]</td>
<td>12.5‡</td>
<td>62.12</td>
<td>12.4</td>
<td>-</td>
</tr>
</tbody>
</table>

Table 3. Performance analysis of some recent FAS solutions. * - Approximated Throughput \( T \), as model did not run on TensorRT. \( P \) - 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.

<table>
<thead>
<tr>
<th>Model Name</th>
<th>Max Resolution</th>
<th>Throughput</th>
</tr>
</thead>
<tbody>
<tr>
<td>ResNet18</td>
<td>1440</td>
<td>2.51</td>
</tr>
<tr>
<td>ResNet34</td>
<td>640</td>
<td>6.71</td>
</tr>
<tr>
<td>ResNet50</td>
<td>640</td>
<td>4.99</td>
</tr>
<tr>
<td>ResNet101</td>
<td>512</td>
<td>4.98</td>
</tr>
<tr>
<td>MobileNetV2 0.25</td>
<td>1440</td>
<td>7.37</td>
</tr>
<tr>
<td>MobileNetV2 0.5</td>
<td>1080</td>
<td>6.67</td>
</tr>
<tr>
<td>MobileNetV2 0.75</td>
<td>1024</td>
<td>5.16</td>
</tr>
<tr>
<td>MobileNetV2</td>
<td>896</td>
<td>6.12</td>
</tr>
<tr>
<td>RetinaFace Mn25</td>
<td>1440</td>
<td>6.94</td>
</tr>
<tr>
<td>RetinaFace Re50</td>
<td>640</td>
<td>3.89</td>
</tr>
</tbody>
</table>

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.

Figure 5. Distribution of the Ratio of Throughput on FP32 to FP16, on TensorRT.
resolution, i.e., over 1200 patches of size $112 \times 112$ per second. At $112 \times 112$ resolution, the $T$ 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

![Accuracy vs Performance Graph](image_url)

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 $T$, shown in Figure 6 (top). The analysis is also presented with respect to the model size represented by the number of parameters $P$, 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 $T$ for larger models, ResNet50 can be a very balanced choice at $T = 33$. The ShuffleNetV2 models provide on average the highest accuracy at a given $T$. 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 ($P$) which is related to model size, shown in Fig. 6 (bottom). MobileNetV2 & MnasNet0.3 are close to ResNet18 in $acc$ but far apart in $P$, as shown in Fig. 6 (bottom). This allows us to design a trade-off between $acc$ & $P$, at constant $T$. $P$ 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, $P$ is insufficient to describe memory footprint at run time. As an example, the models Densenet121, & Densenet169 hugely outperform ResNet18 in $acc$ to $P$ 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 $P$ 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 empirically 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](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


[40] Zitong Yu, Yuxiao 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.


