EffBin: Efficient Face Recognition via Binary Neural Networks

Susim Roy
University at Buffalo
susimmuk@buffalo.edu

Bharat Yalavarthi
University at Buffalo
byalavar@buffalo.edu

Nalini Ratha
University at Buffalo
nratha@buffalo.edu

***Abstract*—Face recognition models have achieved remarkable advances through deep learning, with many techniques matching or surpassing human-level recognition performance under diverse environmental conditions. However, while prior research has predominantly focused on improving recognition accuracy, little attention has been paid to improving computational efficiency and reducing memory usage. These aspects are critical for deploying face recognition systems, efficiently processing large-scale data, and enabling fast inference. To address this gap, we propose an approach that employs 1-bit activations and weights in widely used face recognition models, such as AdaFace, while preserving high recognition accuracy. Additionally, we significantly accelerate inference by using a custom CUDA kernel tailored to our specific convolutional requirements. Finally, we demonstrate the generalizability of our method, achieving promising results across five standard face recognition datasets. This work paves the way for more efficient and scalable face recognition solutions without compromising performance.**

***Index Terms*—Face Recognition, Binary Networks, Computational Efficiency**

I. Introduction

In recent years, the rapid advancement of deep learning techniques has led to the remarkable success of convolutional neural networks (CNNs) in the field of face recognition (FR). Advanced network architectures [11], [26], [28] and discriminative learning approaches [23], [27], [31] have significantly boosted the performance of FR models to unprecedented levels. Face recognition can generally be categorized into two tasks: face identification and face verification [13]. Face identification classifies an image to a specific identity, while face verification determines whether a pair of images belongs to the same identity. In both cases, a typical face recognition pipeline involves detecting and aligning a face [34], mapping it into a feature vector (embedding) [7], [8], and comparing the embeddings of different images to measure identity similarity. To achieve this, state-of-the-art solutions train deep neural networks using either embedding-focused approaches, such as Triplet loss [23], or identity-classification methods [4]. Despite significant improvements in recognition accuracy, state-of-the-art FR models often lack constraints on memory and computational efficiency. These models typically involve large numbers of parameters, requiring extensive memory and substantial computational resources, which pose challenges for deploying them on mobile platforms, robots, or embedded systems. While recent efforts have explored lightweight CNN architectures, such as MobileNets [5], ShuffleNet [19], and MixNets [1], as well as hybrid Transformer-CNN networks [9], these methods still rely on full-precision 32-bit model weights during inference, limiting their deployment efficiency. In this work, we introduce a Binary Neural Network (BNN)-based face recognition approach inspired by recent advancements in both BNNs and FR models, specifically AdaBin [29] and AdaFace [14]. Our method employs 1-bit activations and weights, significantly reducing memory consumption and inference time while maintaining high recognition accuracy across five widely used datasets. Binary neural networks are particularly appealing due to their tiny storage requirements and efficient inference [6], [17], [30], [32], [38], achieved through binarizing both weights and activations and utilizing bitwise operations for efficient convolutions. Our experiments demonstrate that the proposed BNN-based approach achieves recognition performance comparable to state-of-the-art FR models, paving the way for scalable, efficient deployment of face recognition systems. As the demand for compact and high-performance FR models continues to grow, our work provides a practical solution for enabling lightweight, efficient and effective face recognition.

II. Related Work

Our work relates to the rich prior work on increasing the efficiency of deep networks. Several techniques like binary networks, quantization, and pruning were explored to make decrease the inference latency and the memory footprint of large networks. In this work we use the techniques of binary neural networks to speedup the face recognition. [6] first introduced the concept of binary networks to replace the costly multiply and accumulate operations with simple accumulations. Rastegari et al. [22] introduces XNOR-Net which performs efficient CNN approximations using binary weights, activations and operations that achieve up to 32× memory savings and 58× faster operations, enabling real-time performance on CPUs with competitive accuracy on ImageNet. Following XNOR-Net, authors of [2] introduced an updated version called XNOR-Net++ which learns discriminative scaling factors via backpropagation, achieving up to 6% higher accuracy on ImageNet compared to XNOR-Net under the same computational budget. Tu et al. [29] introduces AdaBin, an adaptive binarization method for binary neural networks that optimizes binary sets per layer, achieving state-of-the-art performance, including 66.4% Top-1 accuracy on ImageNet

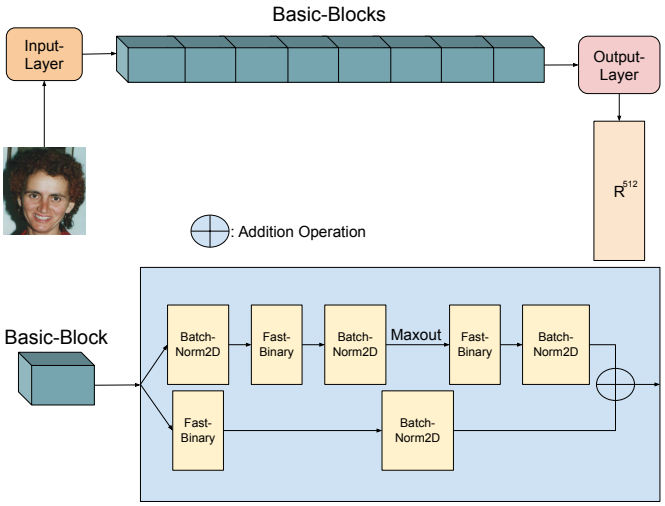

Fig. 1. The above figure shows our architecture for the proposed approach. We introduce the Fast-Binary layer which can be easily inserted in place of any convolution layer and trained. The overall architecture follows the structure of resnet-18 while making changes to the activation function to Maxout and replacement of convolution layer. The input and output layer are same as the original architecture.

with ResNet-18. We use the training methodology of AdaBin in our work. [35] introduces techniques to jointly optimize efficiency and accuracy, allowing to trade-off between the both objectives. [16] propose INSTA-BNN, which dynamically adjusts quantization thresholds for improved accuracy, and [33] propose ReCU, which revives "dead weights" to enhance training efficiency and performance. Prior work has also explored quantization to reduce the inference latency of which binary networks are special case. [25] presents SPARQ, a sparsity-aware quantization method that dynamically leverages activation sparsity to achieve efficient 4-bit quantization with minimal accuracy degradation, enabling practical hardware implementation. [15] shows that quantization generally outperforms pruning for neural network compression while [18] introduces Adaptive Floating-Point (AFP), a novel format that enhances compression rates, eliminates de-quantization, and reduces energy consumption by 11.2× with minimal accuracy loss.

III. Methodology

In this part, we first describe how to binarize weights and activations. Then, we describe the trends followed by state-of-the-art face recognition models like Adaface in better recognition abilities followed by finally the method via which we merged the two techniques through a hardware implementation on the CUDA kernel.

A. Binary-Weight Networks

In order to constrain a convolutional neural network, to have binary weights, we estimate the real-value weight filter W using a binary filter $B \in \{+1, -1\}^{c \times w \times h}$ and a scaling factor $\alpha \in \mathbb{R}^+$ such that $W \approx \alpha B$. A convolutional operation can be approximated by:

$$I * W \approx (I \oplus B)\alpha \tag{1}$$

where $\oplus$ indicates a convolution without any multiplication. Since the weight values are binary, we can implement the convolution with additions and subtractions. The binary weight filters reduce memory usage by a factor of $\sim 32\times$ compared to single precision filters. To find the optimal weight filters, we solve the following optimization:

$$\begin{aligned} J(B, \alpha) &= \|W - \alpha B\|^2 \\ \alpha^*, B^* &= \arg\min_{\alpha, B} J(B, \alpha) \end{aligned} \tag{2}$$

B. XNOR-Networks

While we can deduce from the above section that we can devise binary weights, we are not fully utilizing the bitwise operation like XNOR until we have binarized the activations also. Now, we explain how to binarize both weights and inputs, so convolutions can be implemented efficiently using XNOR and bitcounting operations. A convolution consist of repeating a shift operation and a dot product. Shift operation moves the weight filter over the input and the dot product performs element-wise multiplications between the values of the weight filter and the corresponding part of the input. If we express dot product in terms of binary operations, convolution can be approximated using binary operations. To approximate the dot product between $X, W \in \mathbb{R}^n$ such that $X^\top W \approx \beta H^\top \alpha B$, where $H, B \in \{+1, -1\}^n$ and $\beta, \alpha \in \mathbb{R}^+$, we solve the following optimization:

$$\alpha^*, B^*, \beta^*, H^* = \arg\min_{\alpha, B, \beta, H} \|X \odot W - \beta\alpha H \odot B\| \tag{3}$$

where $\odot$ indicates element-wise product.

C. Adaptive Binary-Sets

While XNOR-Net [22] minimizes the mean squared error (MSE) by multiplying a scale factor whereas IR-Net [21] obtains the maximum information entropy by weight reshaping and then conduct the same operation as XNOR-Net. However, these methods can not get accurate quantization error between binarized data and real-valued data due to the following limitations. Firstly, the center position of previous binarized values $\{-1, +1\}$ is always 0, which is not aligned with the center of original real-valued weights. Secondly, MSE is a simple metric to evaluate the quantization error but do not consider the distribution similarity between binarized data and real-valued data. Therefore, the weights are binarized using the following oepration:

$$w_b = \mathcal{B}(w) = \begin{cases} w_{b1} = \beta_w - \alpha_w, & w < \beta_w, \\ w_{b2} = \beta_w + \alpha_w, & w \geq \beta_w \end{cases} \tag{4}$$

where α_w and β_w are distance and center of binarized weights, the binary elements of w_b in the forward is $\beta_w - \alpha_w$ and

$\beta_w + \alpha_w$. Further, to find the center of the distribution and minimize the KLD, we need β_w to be the following:

$$\beta_w = \mathbb{E}(w) \approx \frac{1}{c \times k \times k} \sum_{m=0}^{c-1} \sum_{j=0}^{k-1} \sum_{i=0}^{k-1} w_{m,j,i} \tag{5}$$

We also assume that the weights follow a bell-shaped distribution and approximate the α_w to be the standard deviation as shown below:

$$\alpha_w = \frac{||w - \beta_w||_2}{\sqrt{c \times k \times k}} \tag{6}$$

where $|| \cdot ||_2$ is the l_2 norm. These two variables are updated in every training step channel-wise along with the real-valued weights which are in turn updated via backpropagation.
In a similar manner, we also frame the activations to be binary weight matrices. Previous works such as [3] have already shown that activation quantization is a challenging task with low bit-width, and has much more impacts to the final performance than weight. Therefore, to keep in line with the weights, we also binarize the activations using:

$$a_b = \mathcal{B}(a) = \begin{cases} a_{b1} = \beta_a - \alpha_a, & a < \beta_a, \\ a_{b2} = \beta_a + \alpha_a, & a \geq \beta_a \end{cases} \tag{7}$$

Here, β_a and α_a are learnable parameters which are updated via gradient-descent in each training iteration. These values can be interpreted as the center and distance values respectively. However, to prevent gradient-explosion, we apply the following transformation function on the activation values:

$$a_b = \alpha_a \times \text{Sign}(\text{Htanh}(\frac{a - \beta_a}{\alpha_a})) + \beta_a \tag{8}$$

Accumulating all the transformations, we can write the overall convolution operation as follows:

$$\alpha_a \alpha_w (a_b \odot w_b) + \alpha_a \beta_w (I \odot a_b) + g(w) \tag{9}$$

Here, $\odot$ is the convolution operator and $g(w)$ is a precomputed value which can be found during training and stored and then used offline during inference. Finally, we use the Maxout [10] activation function to enhance the positive nature of the feature maps instead of PReLU [12].

D. Quality Adaptive Margin

AdaFace introduced quality adaptive margin to address issues with unidentifiable images and images with low quality, an adaptive margin function based on the feature norm was proposed where it was shown that different margin functions emphasize sample difficulty and that the feature norm effectively identifies low-quality images. Combining these insights, a new loss for face recognition (FR) that incorporates an Image Quality Indicator (IQI) was introduced as follows:

$$\hat{\|z_i\|} = \left\lfloor \frac{\|z_i\| - \mu_z}{\sigma_z / h} \right\rfloor_{-1}^{1} \tag{10}$$

AdaFace uses this indicator to determine if a hard sample needs to be emphasized unlike the prior face recognition techniques which emphasized all hard samples leading to lower performance. A hard sample is prioritized if it is high quality and de-emphasized if its a low quality image ensuring balanced gradient scaling near the decision boundary.

E. GPU Implementation:

Here, we explain our main ideas to obtain the speed up of binary convolution by referring to certain functions:

a) Binary Encoding: We first perform an encoding of the 32-bit floating numbers of activations and filter weights to binary values of 0 and 1. This significantly reduces memory consumption and accelerates the computation because binary arithmetic is much faster compared to floating-point arithmetic. To do this, we use a kernel which converts floating-point values into their binary representation, combining multiple bits into one group,i.e., 16 bits per group and storing them as integers. Next, we take care of the encoding of a single row, where it goes through an array and creates a bitmask for every element. It does this either for the filter or for the input, depending on the setting of a flag variable.

b) Matrix Multiplication (Binary GEMM): The primary computational kernel is the binary GEMM. A binary multiplication is achieved with bitwise operations (XOR) between the packed binary input and filter values. This binary matrix multiplication is performed by the kernel, $binary\_gemm\_kernel$, where it calculates the Hamming distance between the input and filter bit patterns. This is done in CUDA parallelism, where every thread computes the XOR between corresponding bits of input and filter and then counts the number of differing bits using the *__popc* intrinsic, which gives the population count. This binary GEMM operation computes results in a matrix that contains the binary convolution output.

c) Result Aggregation: Once the binary GEMM computation has been done, the result gets aggregated in $final\_kernel$. The contributions coming out of every single one of those packed bits in encoding will contribute towards giving the final result of every element of this convolution output by summing up all contributions. The aggregation calculates the final values by summing up all the contributions of the packed bits in $output\_sum$ and correcting them with a formula, $k - 2 * output\_sum$, where k denotes the number of bits packed into each integer. This step is crucial for converting the binary results back to an appropriate value range.

F. Fast Binary Layer:

Our Fast Binary layer essentially performs Equation 9 where the convolution operation is performed via our CUDA implementation. We stored the $g(w)$ matrix with each convolution layer weight since $g(w) = \alpha_w \beta_a (w_b \odot I) + \beta_w \beta_a (I)$ which is independent of the input.

IV. Experiments and Discussion

In this section, we will provide all the necessary details used in our experiments including the datasets, models and subsequent results.

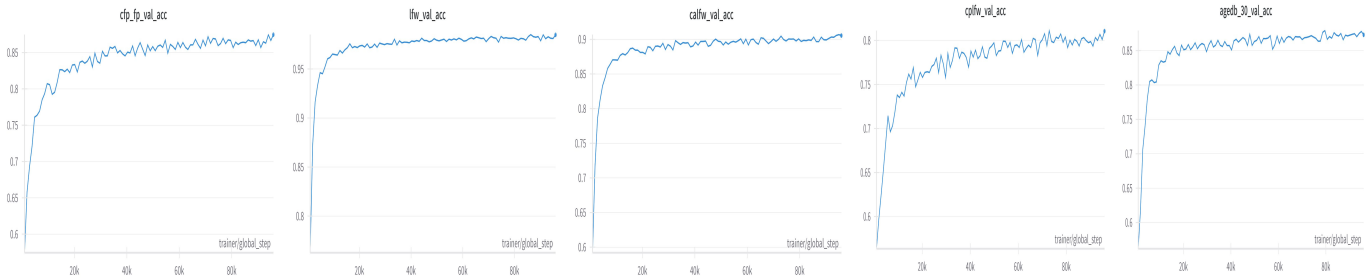

Fig. 2. The above five line plots show the increase in validation accuracy of the five datasets over 100 epochs.

A. Datasets and Models:

We use WebFace4M [39] as our training datasets. It contains 4.2M facial images. We test on high-quality image datasets which include LFW [13], CPLFW [36], CFP-FP [24], AgeDB [20], and CALFW [37]. Each of these datasets have 12000 samples for validation and testing. These datasets are popular benchmarks for FR in controlled environmental settings. Although, they have variations in pose, lighting and age, they are still considered to be high in quality. The backbone model in our case is the standard resnet-18 with modifications as in [7].

B. Preprocessing and Hyperparameter Settings:

We first preprocess the dataset by cropping and aligning the facial images, maintaining the procedure followed in [7] which results in images of size 112×112. We used the SGD optimizer with a weight decay of $1e-5$. We also used the Cosine Annealing scheduler for the learning rate with the total number of epochs being 100. Finally, we set the scale parameter(s) to 64, margin m to 0.4 and quality indicator h to 0.33. All training and validation were performed on a single NVIDIA-H100 GPU.

C. Evaluation on different benchmarks:

a) Training: BNNs are known to be unstable during training due to the 1-bit constraint. However, here we observe from Figure 3 that the loss curve is smooth over the 100 epochs and gradually decreases while flattening out in the last few epochs, indicating that it saturates. Even if we observe the train loss per step, we notice a lot of small variations around the mean loss. However, these never have any abnormal jumps and have a gradual decrease to a fixed point. This can be further strengthened by the validation accuracies in Figure 2, which shows that the validation accuracy increases over the epochs with slight variations. As we can observe in the case of LFW and AgeDB30, the saturation had already begun at a very early epoch and there was only an incremental increase in the validation accuracy.

b) Test Accuracy: We show the results of our proposed approach on the five datasets in Table I. As we can observe, we reach almost the same SOTA accuracy of Adaface in LFW dataset with $< 1\%$ error and $\approx 4\%$ error in the overall average test accuracy over the 5 datasets. Our key objective here was to show that we can reach near the best accuracy during inference even while reducing the memory storage from 32 bits to 1-bit.

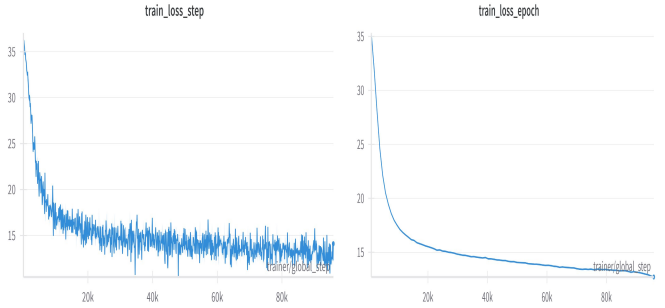

Fig. 3. The left plot shows the train loss variations with each step while the right plot shows the smooth descent of the training loss over epochs.

TABLE I
TEST ACCURACY ON FIVE VALIDATION DATASETS OF OUR PROPOSED FACE RECOGNITION APPROACH COMPARED WITH ORIGINAL ADAFACE.

Dataset	LFW	CFP-FP	CPLFW	AgeDB	CALFW	Avg Acc
Adaface	99.13%	92.59%	87.00%	92.72%	92.65%	**92.82%**
Ours	98.48%	87.47%	81.06%	87.28%	90.55%	**88.97%**

c) Speedup in Inference: Modern GPU systems where pytorch's convolution class has been implemented, there has been a lot of optimization. For example, inside a loop, it can be faster due to optimized batch processing, memory locality, and hardware acceleration. When multiple operations are grouped together, the underlying framework and hardware can better utilize parallelism and perform fewer memory accesses. Additionally, running convolution in a loop may lead to more efficient use of caching and less overhead from multiple individual function calls. Due to this, if we compare the optimized convolution with our binary convolution it would be unfair.

Thus, compared our approach to pytorch's $F.conv2d$ function at each convolution stage of the model. In Figure 4, the blue bars indicate the time taken by our binary convolution and the yellow lines represent the normal convolution time. As we can observe, the average time for each binary convolution is in the order of $10^{-3}s$ while a normal convolution takes around $2s$ to do the same operation. This shows around $1000\times$

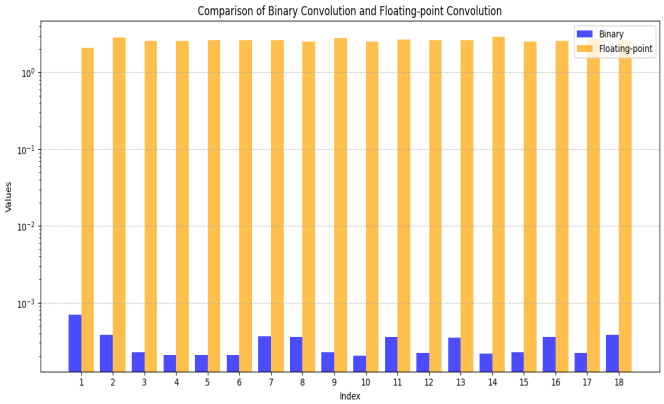

Fig. 4. The figure shows the comparsion of the running time of each convolution layer when using floating point numbers(shown in yellow) and 1-bit convolution(shown in blue). In both the cases, both the input and weight matrices are identical with same stride,dilation,padding and groups values.

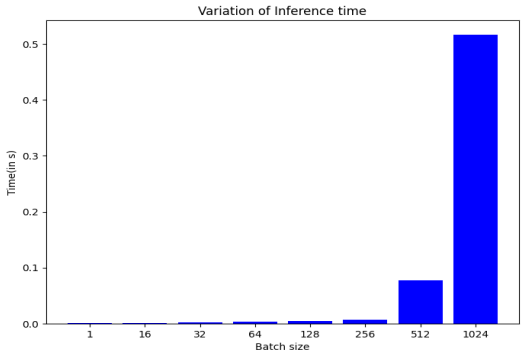

Fig. 5. In the above figure, we varied the batch size while keeping the image size, in-channels, filters and other parameters like stride fixed and same as the 18th conv layer of resnet-18.

improvement over a normal convolution. However, we need to note here that our compared pytorch version did not use any caching or other optimization methods as described previously and was initialized everytime we needed to find the outputs from a layer.

D. Ablation Studies:

In this part, we take a closer look on how varying the parameters of our implemented binary convolution module changes the running time during inference.

1) Effect of Batch Size: Most FR models are known to work when we have a mixed quality of images. Thus, a large batch size is important for proper learning of the network parameters. In Figure 5, we observe that as long as batch-size$\leq$ 256, we have negligible amount of running time for the 18th convolution layer. However, as soon as the batch size reaches 1024, it takes around $0.5s$ to calculate the filter response. This shows that increasing batch size takes more time to operate on. This is because the CUDA kernel works on the basis of encoding rows and columns of each batch

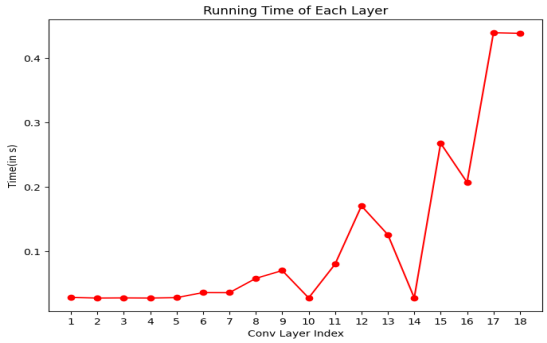

Fig. 6. In this figure, we kept the batch size fixed to 512 and calculated the time needed by each convolution layer of the network with the other parameters same as the parameters of the convolution layers of resnet18.

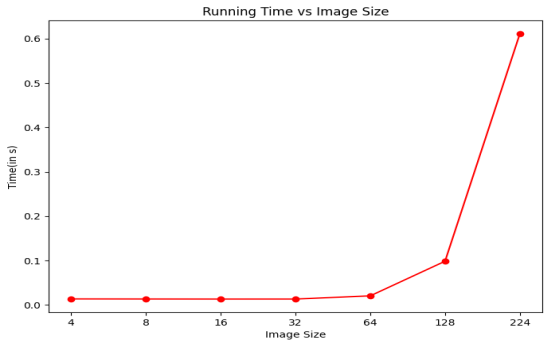

Fig. 7. In this figure, we kept the batch size fixed to 512 and calculated the time needed by the 3rd convolution layer of resnet-18 to calculate the response by varying the input activation map dimensions. Here, each map is of dimension $N \times N$.

individually and hence, it grows as and when the batch size grows.

2) Effect of Filter-Size: In Figure 6, we observe that the running time of each convolution layer increases as we go deeper into the network. This suggests that a larger batch size can affect the running time of deep binary networks. This is because we have more number of filters to process in the deeper layers which increases the number of bitwise signed integers to be processed. Also note that there are dips in certain parts. This is because those are the downsampling layers which decrease the filters. Hence, we can say that since number of output filters were decreased in points, the runtime also decreased in those layers.

3) Effect of Feature Map Size: In Figure 7, we observe that on increasing the image size in the powers of 2 does not have much effect till it reaches 128 after which it increases rapdily. This can be explained due to various reasons:

Increase in Threads and Blocks: The chunk size is represented by $1 + \frac{m-1}{ENCODE\_BITS}$ where m is the number of encoded columns. Since m increases, it takes a toll on the number of threads which can be processed leading to higher execution time.

Warp Size: The warp size in most NVIDIA GPUs is 32 and in our case it's 32 for the GEMM operations and 8 for encoding the matrices. Since, we have more size than the allowed, some operations have to wait for others to finish. Additionally, in case of feature dimensions of 224×224, it is not a multiple of the warp size which makes some threads idle and reduces the execution time.

V. Conclusion

We proposed a novel approach to use binarized neural networks for faster inference of large datasets by reducing computation overhead of floating points and reduced the memory needed to store weights from 32 bit to 1-bit. Additionally, we also presented a GPU kernel which can take any binary input and filter and calculate the filter response. Whilst this is the first of it's kind implementation for efficient face recognition, we hope to see more hardware-friendly work in this area, gradually shifting towards more energy efficient methods.

References

[1] F. Boutros et al. Mixfacenets: Extremely efficient face recognition networks. In *2021 IEEE International Joint Conference on Biometrics (IJCB)*, pages 1–8, 2021.

[2] A. Bulat and G. Tzimiropoulos. Xnor-net++: Improved binary neural networks, 2019.

[3] Z. Cai et al. Deep learning with low precision by half-wave Gaussian quantization. In *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 5406–5414, 2017.

[4] Q. Cao, L. Shen, W. Xie, O. M. Parkhi, and A. Zisserman. Vggface2: A dataset for recognising faces across pose and age. In *2018 13th IEEE International Conference on Automatic Face Gesture Recognition (FG 2018)*, pages 67–74, 2018.

[5] S. Chen et al. Mobilefacenets: Efficient cnns for accurate real-time face verification on mobile devices. *ArXiv*, 1804:07573, 2018. abs : n. pag.

[6] M. Courbariaux, Y. Bengio, and J.-P. David. Binaryconnect: Training deep neural networks with binary weights during propagations. *Advances in neural information processing systems*, 28, 2015.

[7] J. Deng, J. Guo, J. Yang, N. Xue, I. Kotsia, and S. Zafeiriou. Arcface: Additive angular margin loss for deep face recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 44(10):5962–5979, Oct. 2022.

[8] H. Fang, W. Deng, Y. Zhong, and J. Hu. Generate to adapt: Resolution adaption network for surveillance face recognition. In *Computer Vision – ECCV 2020: 16th European Conference, Glasgow, UK, August 23–28, 2020, Proceedings, Part XV*, page 741–758, Berlin, Heidelberg, 2020. Springer-Verlag.

[9] A. George et al. Edgeface: Efficient face recognition model for edge devices. *IEEE Transactions on Biometrics, Behavior, and Identity Science*, 6:158–168, 2023.

[10] I. J. Goodfellow, D. Warde-Farley, M. Mirza, A. Courville, and Y. Bengio. Maxout networks, 2013.

[11] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition, 2015.

[12] K. He, X. Zhang, S. Ren, and J. Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *2015 IEEE International Conference on Computer Vision (ICCV)*, pages 1026–1034, 2015.

[13] G. Huang, M. Mattar, T. Berg, and E. Learned-Miller. Labeled faces in the wild: A database forstudying face recognition in unconstrained environments. *Tech. rep.*, 10 2008.

[14] M. Kim et al. Adaface: Quality adaptive margin for face recognition. In *2022 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 18729–18738, 2022.

[15] A. Kuzmin, M. Nagel, M. V. Baalen, A. Behboodi, and T. Blankevoort. Pruning vs quantization: Which is better? In *Thirty-seventh Conference on Neural Information Processing Systems*, 2023.

[16] C. Lee, H. Kim, E. Park, and J.-J. Kim. Insta-bnn: Binary neural network with instance-aware threshold. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 17325–17334, 2023.

[17] C. Liu et al. Circulant binary convolutional networks: Enhancing the performance of 1-bit dcnns with circulant back propagation. In *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2686–2694, 2019.

[18] F. Liu, W. Zhao, Z. He, Y. Wang, Z. Wang, C. Dai, X. Liang, and L. Jiang. Improving Neural Network Efficiency via Post-training Quantization with Adaptive Floating-Point . In *2021 IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 5261–5270, 2021.

[19] Y. Martnez-Daz et al. Shufflefacenet: A lightweight face architecture for efficient and highly-accurate face recognition. In *2019 IEEE/CVF International Conference on Computer Vision Workshop (ICCVW)*, pages 2721–2728, 2019.

[20] S. Moschoglou, A. Papaioannou, C. Sagonas, J. Deng, I. Kotsia, and S. Zafeiriou. Agedb: The first manually collected, in-the-wild age database. pages 1997–2005, 07 2017.

[21] H. Qin, R. Gong, X. Liu, M. Shen, Z. Wei, F. Yu, and J. Song. Forward and backward information retention for accurate binary neural networks, 2020.

[22] M. Rastegari, V. Ordonez, J. Redmon, and A. Farhadi. Xnor-net: Imagenet classification using binary convolutional neural networks, 2016.

[23] F. Schroff, D. Kalenichenko, and J. Philbin. Facenet: A unified embedding for face recognition and clustering. In *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 815–823, 2015.

[24] S. Sengupta, J.-C. Chen, C. Castillo, V. M. Patel, R. Chellappa, and D. W. Jacobs. Frontal to profile face verification in the wild. In *2016 IEEE Winter Conference on Applications of Computer Vision (WACV)*, pages 1–9, 2016.

[25] G. Shomron, F. Gabbay, S. Kurzum, and U. Weiser. Post-training sparsity-aware quantization. In A. Beygelzimer, Y. Dauphin, P. Liang, and J. W. Vaughan, editors, *Advances in Neural Information Processing Systems*, 2021.

[26] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition, 2015.

[27] Y. Sun, X. Wang, and X. Tang. Deep learning face representation by joint identification-verification, 2014.

[28] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich. Going deeper with convolutions, 2014.

[29] Z. Tu et al. Adabin: Improving binary neural networks with adaptive binary sets. In *European Conference on Computer Vision*, 2022.

[30] Z. Wang, J. Lu, and J. Zhou. Learning channel-wise interactions for binary convolutional neural networks. *IEEE Trans Pattern Anal Mach Intell*, 43(10):3432–3445, October 2021. doi:10.1109/TPAMI.2020.2988262. PMID: 32324540. Epub 2021 Sep 2.

[31] Y. Wen, K. Zhang, Z. Li, and Y. Qiao. A discriminative feature learning approach for deep face recognition. In *European Conference on Computer Vision*, 2016.

[32] Y. Xu et al. A main/subsidiary network framework for simplifying binary neural networks. In *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 7147–7155, 2018.

[33] Z. Xu, M. Lin, J. Liu, J. Chen, L. Shao, Y. Gao, Y. Tian, and R. Ji. Recu: Reviving the dead weights in binary neural networks. In *Proceedings of the IEEE/CVF international conference on computer vision*, pages 5198–5208, 2021.

[34] K. Zhang, Z. Zhang, Z. Li, and Y. Qiao. Joint face detection and alignment using multitask cascaded convolutional networks. *IEEE Signal Processing Letters*, 23(10):1499–1503, 2016.

[35] Y. Zhang, Z. Zhang, and L. Lew. Pokebnn: A binary pursuit of lightweight accuracy. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 12475–12485, 2022.

[36] T. Zheng and W. Deng. Cross-pose lfw : A database for studying cross-pose face recognition in unconstrained environments. 2018.

[37] T. Zheng, W. Deng, and J. Hu. Cross-age lfw: A database for studying cross-age face recognition in unconstrained environments, 2017.

[38] S. Zhu, X. Dong, and H. Su. Binary ensemble neural network: More bits per network or more networks per bit? In *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), Long Beach, CA, USA*, pages 4918–4927, 2019. doi:10.1109/CVPR.2019.00506.

[39] Z. Zhu, G. Huang, J. Deng, Y. Ye, J. Huang, X. Chen, J. Zhu, T. Yang, J. Lu, D. Du, and J. Zhou. Webface260m: A benchmark unveiling the power of million-scale deep face recognition, 2021.