PETAL*face*: Parameter Efficient Transfer Learning for Low-resolution Face Recognition Supplementary Document

Kartik Narayan[1], Nithin Gopalakrishnan Nair[1], Jennifer Xu[2], Rama Chellappa[1], Vishal M. Patel[1]

[1]Johns Hopkins University, [2]Systems and Technology Research

{knaraya4, ngopala2, rchella4, vpatel36}@jhu.edu, jennifer.xu@str.us

The supplementary document is organized into the following sections. First, we discuss additional implementation details. Next, we present the results on other evaluation settings of the IJB-S dataset. Also, we present a gradient analysis of PETAL*face* and compare it to full fine-tuning to highlight that the proposed approach leads to stable convergence. Finally, we provide a failure case analysis of PETAL*face*.

A. Implementation Details

All deployment codes were implemented in PyTorch framework and executed it on eight A5000 GPUs, each equipped with 24GB of memory.The models are trained using the AdamW optimizer and a polynomial learning rate (LR) scheduler, with an initial learning rate of $5e^{-4}$ and a weight decay set to 0.1. We fine-tuned for 40 epochs on TinyFace [1] dataset , utilizing a warm-up of 2 epochs and a batch size of 8. For the BRIAR [2] dataset, we fine-tuned for 10 epochs with one warm-up epoch, also using a batch size of 8. We utilized a low-rank decomposition of 8 for the TinyFace dataset and 32 for the BRIAR dataset. We employed CNN-IQA [4] as our NR-IQA network to assign weightages to the LoRA modules. We present the implementation code modules for the adaptive weight estimated and Adaptive LoRA in the below code fragments. The weightage for the twin LoRA modules is calculated using `generate_alpha`. The final output is calculated as shown in `adaptive_lora`. The complete PETAL*face* training framwork for a single layer is outlined in Algorithm 1.

Algorithm 1 PETAL*face* Training Framework for a single layer

Given: Pre-trained weight matrix $W_0 \in \mathbb{R}^{m\times n}$, LoRA blocks $W_1 \in \mathbb{R}^{m\times n}$ and $W_2 \in \mathbb{R}^{m\times n}$, Input images $X = \{x_i \mid 0 \leq i < p\}$, Image quality estimator $\phi(x)$
for each dataset **do**
  Sample a random l number of samples $x_1, x_2, \ldots, x_l$
  Calculate the mean μ and standard deviation σ of the quality scores:

$$\mu = \frac{1}{l}\sum_{i=1}^{l}\phi(x_i), \quad \sigma = \sqrt{\frac{1}{l}\sum_{i=1}^{l}(\phi(x_i)-\mu)^2}$$

end for
Set the threshold $t = \mu + \sigma$
for each sample x_i in X **do**
  $q_i = \phi(x_i)$
  The weightage α_i is calculated using q_i by:

$$\alpha_i = \begin{cases} 0.5 & \text{if } q_i = t \\ 0.5 - (t - q_i) & \text{if } q_i < t \\ 0.5 + (q_i - t) & \text{if } q_i > t \end{cases}$$

end for
We obtain image quality scores $Q = \{q_i \mid 0 \leq i < p \ni q_i = \phi(x_i), \forall\, 0 \leq i < p\}$
The final output x^i_{out} is calculated as:

$$x^i_{out} = W_0(x_i) + \alpha_i W_1(x_i) + (1-\alpha_i)W_2(x_i)$$

Image quality based weight assignment

```
!pip install pyiqa
iqa = pyiqa.create_metric('cnniqa').cuda()

def generate_alpha(img, iqa, threshold):
    device = img.device
    BS, C, H, W = img.shape
    alpha = torch.zeros((BS, 1), dtype=torch.float32, device=device)

    score = iqa(img)
    for i in range(BS):
        if score[i] == threshold:
            alpha[i] = 0.5
        elif score[i] < threshold:
            alpha[i] = 0.5 - (threshold - score[i])
        else:
            alpha[i] = 0.5 + (score[i] - threshold)
    return alpha
```

Adaptive LoRA

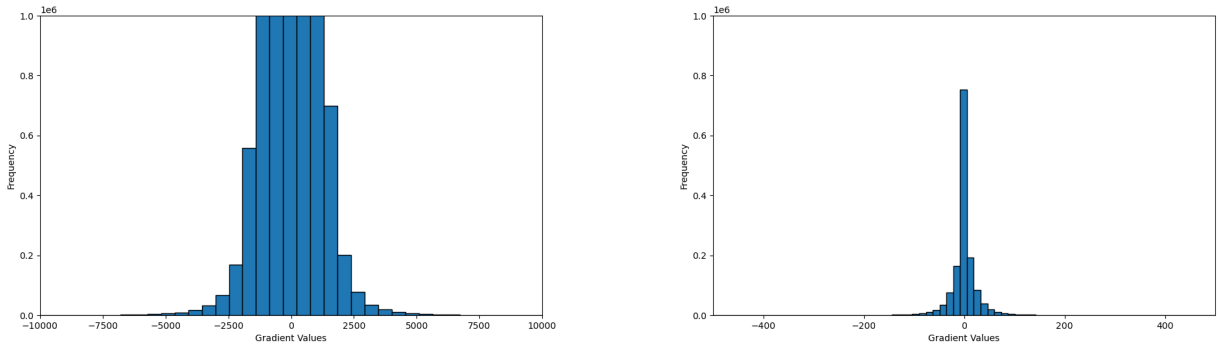
```
class AdaptiveLoRA(nn.Linear):
    def __init__(self, in_features: int, out_features: int,  r: int, scale: int, bias: bool=True) -> None:
        super().__init__(in_features, out_features, bias)
        # LoRA 1
        self.r_1 = r
        self.scale_1 = scale
        self.trainable_lora_down_1 = nn.Linear(in_features, self.r_1, bias=False)
        self.dropout_1 = nn.Dropout(0.1)
        self.trainable_lora_up_1 = nn.Linear(self.r_1, out_features, bias=False)
        self.selector_1 = nn.Identity()
        nn.init.normal_(self.trainable_lora_down_1.weight, std=1/self.r_1)
        nn.init.zeros_(self.trainable_lora_up_1.weight)

        # LoRA 2
        self.r_2 = r
        self.trainable_lora_down_2 = nn.Linear(in_features, self.r_2, bias=False)
        self.dropout_2 = nn.Dropout(0.1)
        self.trainable_lora_up_2 = nn.Linear(self.r_2, out_features, bias=False)
        self.scale_2 = scale
        self.selector_2 = nn.Identity()

        nn.init.normal_(self.trainable_lora_down_2.weight, std=1/self.r_2)
        nn.init.zeros_(self.trainable_lora_up_2.weight)

    def forward(self, x, alpha):
        out = F.linear(x, self.weight, self.bias)
        lora_adjustment_1 = self.scale_1*self.dropout_1(self.trainable_lora_up_1(
        ↪  self.selector_1(self.trainable_lora_down_1(x))))
        lora_adjustment_2 = self.scale_2*self.dropout_2(self.trainable_lora_up_2(
        ↪  self.selector_2(self.trainable_lora_down_2(x))))
        out = out + (1 - alpha)*lora_adjustment_1 + alpha*lora_adjustment_2
        return  out

```



(a) Full fine-tuning

(b) PETAL*face*

Figure 1. Comparison of initial gradients when (a) Full fine-tuning a model and using (b) PETAL*face* fine-tuning approach. We can see that PETAL*face* has small initial gradients which results in stable and gradual convergence. **NOTE:** The scale of the 'Gradient Values' axis for Full fine-tuning and PETAL*face* is different.

B. IJB-S Results

Training	Dataset	Arch.	IJB-S (Surveillance to Single) [3]			IJB-S (Surveillance to Booking) [3]		
			Rank-1	Rank-5	Rank-10	Rank-1	Rank-5	Rank-10
Pre-trained	WBF4M [5]	R50	32.01	45.72	51.25	43.82	55.75	61.28
Pre-trained	WBF4M [5]	Swin-B	33.23	49.85	57.63	46.22	59.40	64.93
Full-FT	WBF4M [5]	Swin-B	4.20	10.95	16.64	5.39	13.31	19.84
PETAL*face*	WBF4M [5]	Swin-B	**37.12**	**51.07**	**57.60**	**43.63**	**59.85**	**66.15**
PETAL*face*	WBF12M [5]	Swin-B	44.40	57.84	63.87	51.09	64.67	70.30

Table 1. Results on IJB-S [3] dataset in *Surveillance-to-single* and *Surveillance-to-booking* settings. The models are fine-tuned on the BRIAR train set. We report the closed-set rank retrieval (Rank-1, Rank-5 and Rank-10). [**BLUE**] indicates the best results for models trained on WebFace4M [5].

The results on the IJB-S dataset in the *Surveillance-to-single* and *Surveillance-to-booking* settings are shown in Table 1. In the *Surveillance-to-single* setting, gallery images are single high-quality images. Similarly, in *Surveillance-to-booking*, we have high-quality gallery images from different angles. The probes are of surveillance quality in both settings. This setup highlights the importance of having two proxy encoders for different resolutions within the same backbone, which are weighted based on input image quality. PETAL*face* shows improved performance with rank-1, rank-5, and rank-10 retrieval accuracies of 44.40, 57.84, and 63.87, respectively, in the *Surveillance-to-single* setting. We see similar improvements in the *Surveillance-to-booking* setting for rank-5 and rank-10 accuracies, with increases of 0.45% and 1.22%. The results demonstrate the generalization capability of the proposed fine-tuning approach. Although the model is fine-tuned on the BRIAR dataset, the knowledge of low-resolution data gained from that can be translated to other datasets such as IJB-S. Additionally, we observe a significant drop in performance when we fully fine-tune the model. We discussed the causes in the main paper and want to reiterate here. Face recognition models are pre-trained on large datasets with high-resolution images. When fine-tuning on low-resolution datasets, the model encounters a domain difference, which leads to large gradient updates initially. This deviates the model from the original pre-trained state abruptly, leading to poor convergence. We provide a gradient analysis in Section C to validate our claims.

C. Gradient Analysis

We analyze the gradients of the model backbone when fully fine-tuning the model versus when using PETAL*face* to fine-tune the model. We plot the frequency of gradient values for the first iteration of training. As shown in the Figure 1, we see that when fully fine-tuning the model, the initial gradients are very large, and even after clipping the gradients, there will be a large number of parameters that will change significantly. This is due to the domain difference between pre-trained and fine-tuned data, leading to an abrupt deviation from pre-trained weights when fully fine-tuning the model. The initial value of gradients when using the PETAL*face* fine-tuning approach results in relatively smaller gradients initially, leading to more stable and gradual convergence and improved performance. Moreover, the original weights remain frozen thereby preserving all information learned during large scale training. This demonstrates the superiority of our approach in efficiently adapting to low-resolution data.

D. Failure Case Analysis

Figure 2. Failure Case Analysis of PETAL*face* on the BRIAR dataset. All the subjects are consented for publication.

We conducted a failure case analysis of the probe videos, as summarized in Figure 2, to examine the limitations of our model. We found that it struggled to recognize faces that were very low in resolution and featured extreme head poses. It also failed in cases of heavy occlusion, where faces were obscured by items like caps, masks, or sunglasses. Additionally, the model performed poorly when faces were degraded by atmospheric turbulence, making recognition difficult. Furthermore, the model failed with probe videos lacking frontal face views, as it could not identify individuals without clear frontal visibility throughout the video.

References

[1] Zhiyi Cheng, Xiatian Zhu, and Shaogang Gong. Low-resolution face recognition. In *Computer Vision–ACCV 2018: 14th Asian Conference on Computer Vision, Perth, Australia, December 2–6, 2018, Revised Selected Papers, Part III 14*, pages 605–621. Springer, 2019. 1

[2] David Cornett, Joel Brogan, Nell Barber, Deniz Aykac, Seth Baird, Nicholas Burchfield, Carl Dukes, Andrew Duncan, Regina Ferrell, Jim Goddard, et al. Expanding accurate person recognition to new altitudes and ranges: The briar dataset. In *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision*, pages 593–602, 2023. 1

[3] Nathan D Kalka, Brianna Maze, James A Duncan, Kevin O'Connor, Stephen Elliott, Kaleb Hebert, Julia Bryan, and Anil K Jain. Ijb–s: Iarpa janus surveillance video benchmark. In *2018 IEEE 9th international conference on biometrics theory, applications and systems (BTAS)*, pages 1–9. IEEE, 2018. 3

[4] Le Kang, Peng Ye, Yi Li, and David Doermann. Convolutional neural networks for no-reference image quality assessment. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1733–1740, 2014. 1

[5] Zheng Zhu, Guan Huang, Jiankang Deng, Yun Ye, Junjie Huang, Xinze Chen, Jiagang Zhu, Tian Yang, Jiwen Lu, Dalong Du, et al. Webface260m: A benchmark unveiling the power of million-scale deep face recognition. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 10492–10502, 2021. 3