Supplementary Materials

FaceLiVT: Face Recognition using Linear Vision Transformer with Structural Reparameterization

In this supplementary material, we present implementation details and additional quality comparisons to further demonstrate the effectiveness and quality of our proposed method in comparison to other existing approaches. By including these extended evaluations, we aim to provide a more comprehensive analysis and highlight the advantages of our method over alternative techniques.

A. Implementation Details of FaceLiVT

The advantages of FaceLiVT are using RepMixer, which can be reparameterized while inferring the model to speed up. Below is the implementation of RepMixer in PyTorch-like code:

```
class RepConv(nn.Module):
    def __init__(self, inc, ouc, ks=1, stride=1,
                 pad=0, groups=1):
        super().__init__()
        self.conv = nn.Conv2d(inc, ouc, ks, stride,
                              pad, groups=groups)
        self.repconv = nn.Conv2d(inc, ouc, ks//2,
                              stride, pad//2, groups=groups)
        self.bn = nn.BatchNorm2d(ouc)

    def forward(self, x):
        xr = self.conv(x) + self.repconv(x)
        return self.bn(xr)

    def forward_deploy(self, x):
        return self.conv(x)

    @torch.no_grad()
    def reparam(self):
        conv = self.conv
        repconv=self.repconv; self.__delattr__('repconv')
        kw, kh = (conv.weight.shape[2]-
                  repconv.weight.shape[2])//2,
                  (conv.weight.shape[3]-
                  repconv.weight.shape[3])//2
        repconv_w = nn.functional.pad(repconv.weight,
                    [kh,kh,kw,kw])
        repconv_b = repconv.bias

        final_conv_w = conv.weight + repconv_w
        final_conv_b = conv.bias + repconv_b

        conv.weight.data.copy_(final_conv_w)
        conv.bias.data.copy_(final_conv_b)

        bn = self.bn
        w = bn.weight / (bn.running_var + bn.eps)**0.5
        w = conv.weight * w[:, None, None, None]
        b = bn.bias + (conv.bias - bn.running_mean) *
            bn.weight / (bn.running_var + bn.eps)**0.5
        self.__delattr__('bn')

        conv.weight.data.copy_(w)
        conv.bias.data.copy_(b)

        self.forward = self.forward_deploy
        self.conv = conv
        return self
```

The other advantages of FaceLiVT are using Multi-Head Linear Attention (MHLA), which can reduce the complexity of conventional Multi-Head Self Attention from quadratic $\mathcal{O}(N^2)$ into linear $\mathcal{O}(Nr)$. Below is the implementation of MHLA in PyTorch-like code:

```
class MultiHeadLinearAttention(torch.nn.Module):
    def __init__(self, dim, resolution,
                 ratio=4, act_layer=nn.ReLU):
        super().__init__()
        self.n_head = 16
        self.res=resolution**2
        self.dim=dim
        lin1=[]
        lin2=[]
        self.norm=nn.GroupNorm(1, dim)
        self.act_layer = act_layer()
        for i in range(self.n_head):
            lin1.append(nn.Linear(self.res, self.res*ratio))
            lin2.append(nn.Linear(self.res*ratio, self.res))
        self.lin1 = torch.nn.ModuleList(lin1)
        self.lin2 = torch.nn.ModuleList(lin2)

    def forward(self, x):
        B,C,H,W= x.shape
        x = self.norm(x).reshape(-1, self.dim, self.res)
        x = x.chunk(self.n_head, dim=1)
        x_out = []
        for i in range(self.n_head):
            feat = self.lin1[i](x[i])
            feat = self.act_layer(feat)
            feat = self.lin2[i](feat)
            x_out.append(feat)
        x = torch.cat(x_out, dim=-1)
        x = x.reshape(B,C,H,W)
        return x
```

We employ a similar training methodology as [1]–[3] on Glint360K [4]. The AdamW optimizer [5] and a polynomial decay learning rate schedule are utilized. The starting learning rate is 6×10^{-3} with a minimum of 1×10^{-5}. The total batch size amounts to 384 using $3\times$ NVIDIA RTX-A6000 GPUs, and the weight decay is 1×10^{-4}. The model is trained for 20 epochs with a pre-processing resolution of (112×112).

TABLE I
FACELIVT TRAINING HYPERPARAMETERS ON GLINT360K DATASETS

Hyperparameters	Config
optimizer	AdamW
learning	rate 0.006
batch	size 384
LR schedule	polynomial
warmup epochs	0
training epochs	20
weight decay	0.0001
embedding size	512

B. Detailed Result On Benchmark Test

Table II shows the extension detail of testing in the IJB-B and IJB-C datasets with TAR at different FAR values from $1e^{-5}$ to $1e^{-3}$. Our proposed model is slightly lower than EdgeFace, since it was trained using a larger dataset, WebFace240M, and 80 epochs longer than ours. Due to the limited computation hardware, we trained using Glint360K. With the similar datasets, our proposed FaceLiVT can outperform the CNN-based face recognition model.

Table III presents the inference speed of our proposed FaceLiVT. Although EdgeFace, a Hybrid-ViT with reduced

TABLE II
COMPARISON OF FACELIVT VARIANT WITH STATE-OF-THE-ART ON IJB-B AND IJB-C BENCHMARK DATASET.

Model	IJB-B			IJB-C		
	$1e^{-5}$	$1e^{-4}$	$1e^{-3}$	$1e^{-5}$	$1e^{-4}$	$1e^{-3}$
MobileFaceNetV1 [2]	85.7	92.0	94.7	90.4	93.9	95.9
MobileFaceNet [2]	87.9	92.8	95.6	92.2	94.7	96.6
ShuffleFaceNet-1.5 [2]	86.5	92.3	95.2	91.3	94.3	96.3
EdgeFace-S(0.5) [1]	-	93.6	-	-	95.6	-
ResNet50-ArcFace [6]	80.5	89.9	94.5	86.1	92.1	96.0
FaceLiVT-M	88.4	93.4	95.6	91.3	95.0	96.7
FaceLiVT-M-(LA)	83.7	92.5	95.1	89.25	94.1	96.3
ShuffleFaceNet-0.5 [2]	-	-	-	-	-	-
EdgeFace-XS(0.6) [1]	-	92.7	-	-	94.8	-
FaceLiVT-S	68.8	89.1	95.0	66.7	89.7	96.0
FaceLiVT-S-(LA)	58.4	83.4	94.7	56.3	82.5	94.63

parameters and FLOPs, presents a bottleneck in the mobile latency inference test. In comparison to EdgeFace-XS, our proposed FaceLiVT-M(LA) is 8.6 times faster, despite having higher FLOPs, due to effective reparameterization.

TABLE III
COMPARISON OF FACELIVT VARIANT WITH STATE-OF-THE-ART ON IPHONE 15 PRO MOBILE LATENCY TEST.

Model	Type	Param (M)	FLOP (M)	Latency (ms)
MobileFaceNetV1 [2]	CNN	3.4	1100	0.81
MobileFaceNet [2]	CNN	2.0	933	0.77
ShuffleFaceNet-1.5× [2]	CNN	2.6	577	0.69
EdgeFace-S(0.5) [1]	Hybrid ViT	3.6	306	10.21
FaceLiVT-M	Hybrid ViT	14.3	569	1.11
FaceLiVT-M-(LA)	Hybrid ViT	9.75	386	0.67
ShuffleFaceNet-0.5 [2]	CNN	1.4	66.9	0.45
EdgeFace-XS(0.6) [1]	Hybrid ViT	1.77	154	5.82
FaceLiVT-S	Hybrid ViT	5.89	237	0.61
FaceLiVT-S-(LA)	Hybrid ViT	5.05	160	0.47

REFERENCES

[1] A. George, C. Ecabert, H. O. Shahreza, K. Kotwal, and S. Marcel, "Edgeface: Efficient face recognition model for edge devices," *IEEE Transactions on Biometrics, Behavior, and Identity Science*, 2024.

[2] Y. Martinez-Diaz, M. Nicolas-Diaz, H. Mendez-Vazquez, L. S. Luevano, L. Chang, M. Gonzalez-Mendoza, and L. E. Sucar, "Benchmarking lightweight face architectures on specific face recognition scenarios," *Artificial Intelligence Review*, vol. 54, no. 8, pp. 6201–6244, 2021.

[3] Y. Martindez-Diaz, L. S. Luevano, H. Mendez-Vazquez, M. Nicolas-Diaz, L. Chang, and M. Gonzalez-Mendoza, "Shufflefacenet: A lightweight face architecture for efficient and highly-accurate face recognition," in *Proceedings of the IEEE/CVF international conference on computer vision workshops*, pp. 0–0, 2019.

[4] X. An, X. Zhu, Y. Gao, Y. Xiao, Y. Zhao, Z. Feng, L. Wu, B. Qin, M. Zhang, D. Zhang, *et al.*, "Partial fc: Training 10 million identities on a single machine," in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pp. 1445–1449, 2021.

[5] I. Loshchilov, "Decoupled weight decay regularization," *arXiv preprint arXiv:1711.05101*, 2017.

[6] J. Deng, J. Guo, N. Xue, and S. Zafeiriou, "Arcface: Additive angular margin loss for deep face recognition," in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pp. 4690–4699, 2019.