复现论文题目：DECA: Detailed Expression Capture and Animation

1. 研究背景

1.1 发展现状

三维人脸重建是计算机图形学领域的一个重要研究方向，其目标是从单张或多张图像中恢复出高精度的三维面部几何外观，自 Vetter 和 Blanz[1999]首次提出基于单张图像的 3D 面部重建方法(3DMM)以来，该领域取得了显著的进展。

随着深度学习技术的兴起，三维人脸重建方法在精度、效率和鲁棒性方面得到了显著提升，推动了其在虚拟现实、增强现实、视频编辑、图像合成、人脸识别等领域的广泛应用。然而，虽然现有的方法可以重建出面部形状，但是在捕捉表情依赖的细节（如皱纹，眼球等）方面仍然存在挑战。这些细节对于提升真实情感和支持情感分析至关重要。

因此，本文提出了 DECA（Detailed Expression Capture and Animation），该方法在不需要 2D 图像到 3D 模型的监督下，可以从广泛的图像中学习可变化的位移模型（Animatable displacement model），在仅需一张图像的情况下还原特定人脸的表情皱纹等细节。

1.2 DECA 简介

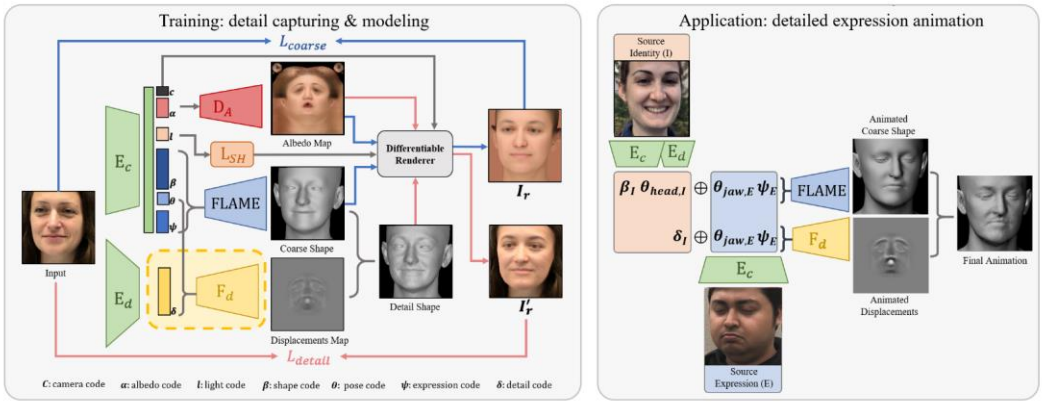


图 1.1 DECA 结构图

DECA 从广泛地训练图像中学习一个带有几何信息的参数化脸部模型，如图 1.1 左。训练完成后，DECA 可以从单张图像 I 中重建出具有详细面部几何的 3D 头部，如图 1.1 右。通过将学习到的细节参数话，DECA 可以控制 FLAME 模型的表情和下颌参数，并且可以使其动画化。

DECA 的核心思想是，同一个人的面部会因为表情变化而展现出不同的细节（如皱纹），但其他形状属性保持不变，因此，面部细节应该被分解为静态的个人特征细节和动态的表情依赖细节。如图 1.1，DECA 学习了两种解码器 Ec（Encoder Coarse）和 Ed（Encoder Detail），分别用于人脸的粗略重建和面部表情的细节重建。

2. 方法框架

2.1 粗略重建

2.1.1 损失函数定义

DECA 使用 FLAME 模型对人脸进行粗略重建，训练 Corase Encoder 解码器。

首先使用 ResNet50[He at al.2016]网络，将一个包含人脸的 2D 图像作为网络的输入，来得到一个低维隐变量，之后解码来合成一张 2D 图像 Ir，并最小化合成图像和输入图像之间的差异。隐变量包含 FLAME 模型的参数 β, ψ, θ 来重建几何信息，反射率系数 α，相机 c 和光照参数 l。最终解码器E_c预测输出 256 维的隐变量。

在训练时，给定一个人脸的多幅图像，对应的身份标签c_i和 68 个人脸关键点k_i，粗略重建最小化以下损失函数。

$$L_{coarse} = L_{lmk} + L_{eye} + L_{pho} + L_{id} + L_{sc} + L_{reg}$$

L_{lmk}为标记点重投影损失函数，用于测量输入的人脸图片中标记点k_i和 FLAME 模型表面相对应的标记点投影到图像后的差异，该项损失函数可以用于预测相机参数。L_{lmk}定义如下：

$$L_{lmk} = \sum_{i=1}^{68} \left|\left| k_i - s\Pi(M_i) + t \right|\right|_1$$

标记点损失具体实现如下

```
def landmark_loss(predicted_landmarks, landmarks_gt, weight=1.):
    # (predicted_theta, predicted_verts, predicted_landmarks) = ringnet_outputs[-1]
    if torch.is_tensor(landmarks_gt) is not True:
        real_2d = torch.cat(landmarks_gt).cuda()
    else:
        real_2d = torch.cat( tensors: [landmarks_gt, torch.ones((landmarks_gt.shape[0], 68, 1)).cuda()], dim=-1)
    # real_2d = torch.cat(landmarks_gt).cuda()

    loss_lmk_2d = batch_kp_2d_l1_loss(real_2d, predicted_landmarks)
    return loss_lmk_2d * weight
```

L_{eye}为眼睛闭合损失，计算上眼皮和下眼皮上的关键点k_i和k_j之间的相对偏移与 FLAME 模型表面对应点M_i和Mj投影到图像后的差异。定义如下，其中 E 是上下眼皮标记点对的集合，相比于L_{lmk}，L_{eye}具有平移不变性，即投影后上下眼皮标记点之间的相对距离是不变的，因此所以增加此损失项后眼部重建效果更好。作者在文中也给出了消融实验来证明。

$$L_{eye} = \sum_{(i,j)\in E} \left|\left| k_i - k_j - s\Pi\left(M_i - M_j\right) \right|\right|_1$$

眼部闭合损失具体实现如下

```
def eyed_loss(predicted_landmarks, landmarks_gt, weight=1.):
    if torch.is_tensor(landmarks_gt) is not True:
        real_2d = torch.cat(landmarks_gt).cuda()
    else:
        real_2d = torch.cat( tensors: [landmarks_gt, torch.ones((landmarks_gt.shape[0], 68, 1)).cuda()], dim=-1)
    pred_eyed = eye_dis(predicted_landmarks[:,:,:2])
    gt_eyed = eye_dis(real_2d[:,:,:2])

    loss = (pred_eyed - gt_eyed).abs().mean()
    return loss
```

L_{pho}为光照度损失，用于计算输入图像和渲染图像之间的差异，定义如下。其中V_I是面部掩码，在面部皮肤区域值为 1，其他区域为 0，该掩码通过面部分割方法[Nirkin et al.2018]获得。用于仅计算面部区域的损失，避免头发、衣物、眼镜等遮挡物造成的影响。作者在文中也给出了消融实验证明。

$$L_{pho} = \left|\left|V_I \odot (I - I_r)\right|\right|_{1,1}$$

```
losses['photo_detail'] = (
                                uv_texture_patch * uv_vis_mask_patch - uv_texture_gt_patch * uv_vis_mask_patch).abs().mean() * self.cfg.loss.ph
losses['photo_detail_mrf'] = self.mrf_loss(uv_texture_patch * uv_vis_mask_patch,
                                           uv_texture_gt_patch * uv_vis_mask_patch) * self.cfg.loss.photo_D * self.cfg.loss.mrf
```

L_{id}为个体身份损失，用于确保渲染图像与输入图像中的人看起来是同一个人。该损失函数在[Deng et al. 2019;Gecer et al. 2019]中证明可以生成更真实的面部形状。公式如下，该损失鼓励渲染图像捕捉到人物身份的基本特征。图 2 使用消融实验证明此项损失有效。

$$L_{id} = 1 - \frac{f(I) \cdot f(I_r)}{|f(I)|_2 \cdot |f(I_r)|_2}$$

L_{sc}为形状一致性损失，给定一个人的两张图像，编码器E_c应当输出相同的形状参数，即$\beta_i = \beta_j$。Sanyal et al.2019 这篇论文中强制两个系数之间的差异小于不同人对应的形状系数的差异来鼓励形状一致性，但是实际中这个边界并不好确定，所以 DECA 中使用了相比于[Sanyal et al. 2019]中不同的策略，即使用β_j替换β_i，同时保持其他所有参数不变。如果方法正确预测了同一个人两张图像中的面部形状，那么图像之间只交换形状系数β后渲染的图像应该是一样的。

$$L_{sc} = L_{coarse}\left(I_i, R\left(M\left(\beta_j, \theta_i, \psi_i\right), B\left(\alpha_i, l_i, N_{uv,i}\right), c_i\right)\right)$$

2.2 细节重建

2.2.1 Detail Decoder

细节重建通过输出一张位移贴图（Displacement Map）来增强粗略重建得到的几何形状，和粗略重建相同，细节重建训练了一个 Encoder Detail 来将图片编码为 128 维的潜变量δ，来学习特定的细节，该潜变量随后和 FLAME 的表情参数ψ以及和下颌姿态参数θ_{jaw}一起输入到F_d来解码生成位移贴图。

F_d的定义为如下：

$$D = F_d\left(\delta, \psi, \theta_{Jaw}\right)$$

该 Decoder 可以生成具有个人特征的细节，此处的$\psi \in R^{50}$和$\theta_{jaw} \in R^3$来自粗略重建，可以捕捉到皱纹细节。

```
self.D_detail = Generator(
```

```python
class Generator(nn.Module):
    ⚠ YadiraF
    def __init__(self, latent_dim=100, out_channels=1, out_scale=0.01, sample_mode = 'bilinear'):
        super(Generator, self).__init__()
        self.out_scale = out_scale

        self.init_size = 32 // 4  # Initial size before upsampling
        self.l1 = nn.Sequential(nn.Linear(latent_dim, 128 * self.init_size ** 2))
        self.conv_blocks = nn.Sequential(
            nn.BatchNorm2d(128),
            nn.Upsample(scale_factor=2, mode=sample_mode), #16
            nn.Conv2d( in_channels: 128,  out_channels: 128,  kernel_size: 3, stride=1, padding=1),
            nn.BatchNorm2d( num_features: 128,  eps: 0.8),
            nn.LeakyReLU( negative_slope: 0.2, inplace=True),
            nn.Upsample(scale_factor=2, mode=sample_mode), #32
            nn.Conv2d( in_channels: 128,  out_channels: 64,  kernel_size: 3, stride=1, padding=1),
            nn.BatchNorm2d( num_features: 64,  eps: 0.8),
            nn.LeakyReLU( negative_slope: 0.2, inplace=True),
            nn.Upsample(scale_factor=2, mode=sample_mode), #64
            nn.Conv2d( in_channels: 64,  out_channels: 64,  kernel_size: 3, stride=1, padding=1),
            nn.BatchNorm2d( num_features: 64,  eps: 0.8),
            nn.LeakyReLU( negative_slope: 0.2, inplace=True),
            nn.Upsample(scale_factor=2, mode=sample_mode), #128
            nn.Conv2d( in_channels: 64,  out_channels: 32,  kernel_size: 3, stride=1, padding=1),
            nn.BatchNorm2d( num_features: 32,  eps: 0.8),
            nn.LeakyReLU( negative_slope: 0.2, inplace=True),
            nn.Upsample(scale_factor=2, mode=sample_mode), #256
            nn.Conv2d( in_channels: 32,  out_channels: 16,  kernel_size: 3, stride=1, padding=1),
            nn.BatchNorm2d( num_features: 16,  eps: 0.8),
            nn.LeakyReLU( negative_slope: 0.2, inplace=True),
```

最后通过可微渲染的方式输出图像。

2.2.2 损失函数定义

细节重建整体的损失函数定义如下

$$L_{detail} = L_{phoD} + L_{mrf} + L_{sym} + L_{dc} + L_{regD}$$

L_{phoD}为光照度损失，此处定义和粗略重建中的光照度损失相同，不再赘述。

L_{mrf}为隐式多样化马尔可夫随机场（ID-MRF）[Wang et al. 2018]，该损失函数可以从训练网络的不同层中提取特征块，最小化两幅图像的对应最近邻特征块之间的差异，该损失可以鼓励 DECA 捕捉高频细节。

L_{sym}为软对称损失，该损失用来对不可见的面部部分（如眼镜下的皮肤）进行正则化，可以增强自遮挡的鲁棒性，定义如下，其中 V 代表 UV 空间中的面部皮肤掩码，flip 为水平翻转操作，论文中使用消融实验证明该损失可以减少遮挡区域边界伪影的情况。

$$L_{sym} = |V_{uv} \odot (D - flip(D))|_{1,1}$$

L_{regD}为正则化项，用来减少噪声。

L_{dc}为细节一致性损失。该损失引入用来解耦细节。DECA 方法中主要的一大贡献就是将表情和人脸分离，为了能达到此目的，就需要将人物的**自身特定细节**（如痣，毛孔，眉毛以及和表情无关的皱纹）与表情相关的皱纹（即随着面部表情变化而产生的皱纹）分离。人物的特定细节由δ控制，而表情相关的皱纹由 FLAME 模型的表情参数ψ以及和下颌姿态参数θ_{jaw}控制。因此，同一个人的不同照片，应当有相似的几何形状和特定细节，即交换同一个人两张图片的δ参数应当对渲染的图像没有影响。这一思想十分先进，是 DECA 的关键所在。论文中也使用消融实验证明了如果没有该损失，DECA 则无法解耦细节。

给定的两张图像I_i和I_j，细节一致性损失定义如下，其中$\boldsymbol{\delta_j}$参数来自图片 J。

$$L_{dc} = L_{detail}\left(I_i, \mathcal{R}\left(M(\beta_i, \theta_i, \psi_i), A(\alpha_i), F_d\left(\delta_j, \psi_i, \theta_{jaw,i}\right), I_i, c_i\right)\right)$$

细节一致性损失对应的实现如下，其中 codedict 为使用编码器 Encoder 后得到的参数。具体实现如下，codedict['detail']即δ参数，这里打乱了一个批次中图片的顺序，例如一个人有 3 张不同角度的图片，然后将 3 个图片的顺序打乱得到 detailcode_new，然后将两组 detailcode 合并为新的 detail 参数。在代码的最下方可以看到图片和标记点 lmk 也被合并成原来的两倍，这样就可以将原本的图片和打乱后的 detail 细节参数计算损失。

```python
if self.cfg.loss.shape_consistency or self.cfg.loss.detail_consistency:
    '''
    make sure s0, s1 is something to make shape close
    the difference from ||so - s1|| is
    the later encourage s0, s1 is cloase in l2 space, but not really ensure shape will be close
    '''
    new_order = np.array([np.random.permutation(self.K) + i*self.K for i in range(self.batch_size)])
    new_order = new_order.flatten()
    shapecode = codedict['shape']
    if self.train_detail:
        detailcode = codedict['detail']
        detailcode_new = detailcode[new_order]
        codedict['detail'] = torch.cat( tensors: [detailcode, detailcode_new], dim=0)
        codedict['shape'] = torch.cat( tensors: [shapecode, shapecode], dim=0)
    else:
        shapecode_new = shapecode[new_order]
        codedict['shape'] = torch.cat( tensors: [shapecode, shapecode_new], dim=0)
    for key in ['tex', 'exp', 'pose', 'cam', 'light', 'images']:
        code = codedict[key]
        codedict[key] = torch.cat( tensors: [code, code], dim=0)
    ## append gt
    images = torch.cat( tensors: [images, images], dim=0)# images = images.view(-1, images.shape[-3], images.shape[-2], images.shape[-1])
    lmk = torch.cat( tensors: [lmk, lmk], dim=0) #lmk = lmk.view(-1, lmk.shape[-2], lmk.shape[-1])
    # masks = torch.cat([masks, masks], dim=0)
```

3. 实验结果

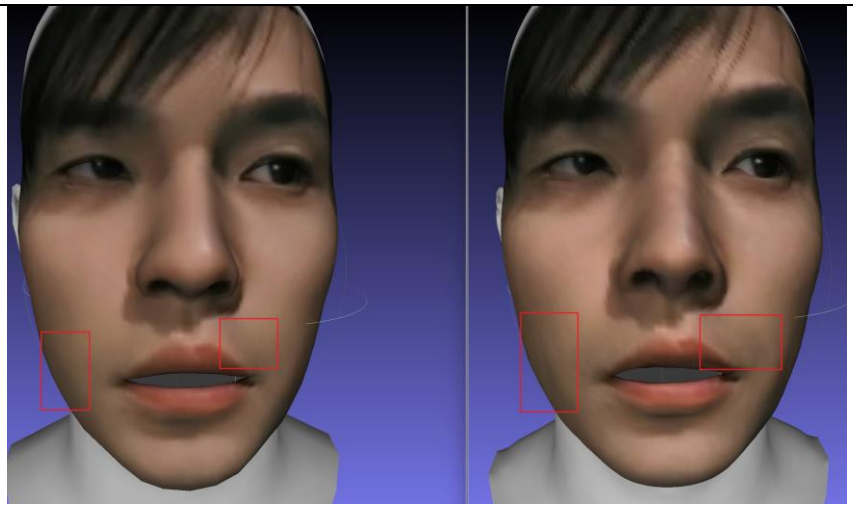
3.1 重建

3.1.1 面部重建

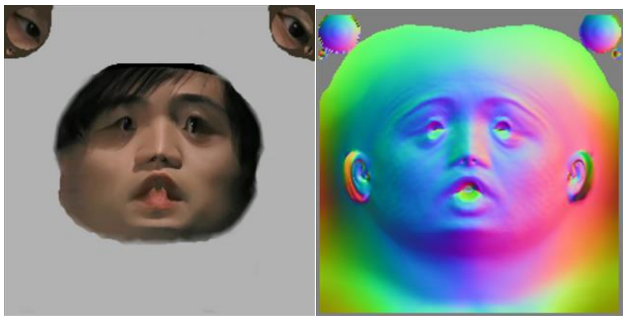
重建的命令为：python demos/demo_reconstruct.py -i TestSamples/examples2 --saveDepth True --saveObj True --rasterizer_type pytorch3d

输入图片为如下左图，粗略重建如右图

下面对比图中，右图为细节重建，左图为粗略重建，具体差异可以看红框部分，细节图片增加了一些脸部细节，可以让重建后的结果更真实。



下左图为原始的纹理贴图，右图为细节重建生成的位移贴图 Displacement Map 转换后的法线贴图 Normal Map



3.1.2 表情迁移

表情迁移命令为： python demos/demo_transfer.py -e TestSamples/examples/exp.png -i TestSamples/examples/img_1.png --rasterizer_type pytorch3d

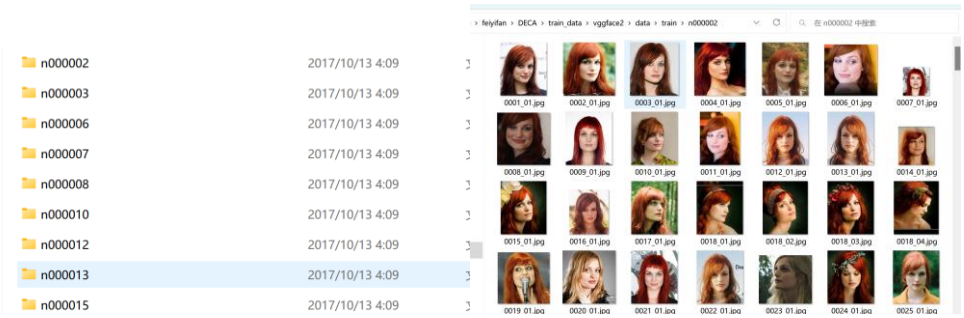
左图为原始图片，右图为需要迁移的表情。

最终结果如下，根据右图最右边的小图可以看出，迁移表情后，人脸明显增加了很多和表情相关的皱纹细节。

3.2 修改数据集，训练

由于 DECA 的项目主页说明信息不足，导致 DECA 训练过程十分困难，作者只给出了训练的代码，并没有说明训练数据集的处理方式，官方 Issue 中有许多人遇到了相同的训练问题但作者也没有回复。因此花费了巨量的时间来跑通整个训练代码。

本次复现使用的训练数据集为 vggface2 人脸数据集，原论文中还使用了 VoxCeleb2，BUPT-Balancedface。Vggface2 数据集包含 n 个文件夹，每个文件夹中都有 n 个同一个人的人脸图片，如下图所示。

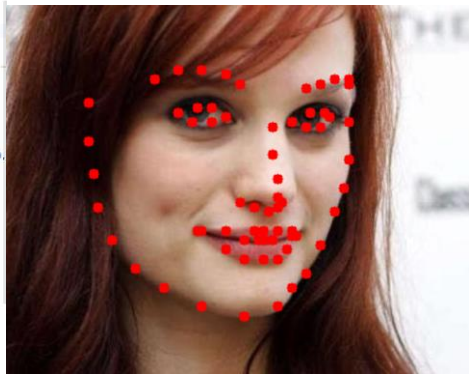


除了训练所需要的图片外，DECA 还需要每个图片的 LandMark 标记点文件和面部区域分割文件，用于计算标记点损失和光照度损失，为了简化训练过程，此处只使用 landmark 文件。如下图所示，修改 datasets 的 vggface 文件，定义图片文件夹和 landmark kpt 文件。

由于项目中说明十分缺失，我选择用其他替代方案来生成 landmark 文件，这里使用了一个 face aligment 库(1adrianb/face-alignment: :fire: 2D and 3D Face alignment library build

using pytorch)来获取人脸的 landmark，最终人脸标记点如下右图所示

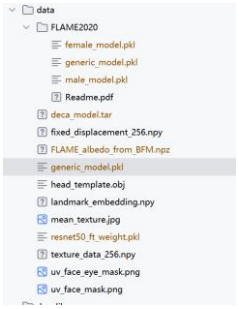
```python
import face_alignment

1 usage
class FaceDetect:...

1 usage
def execute(path, filename):
    file_path = os.path.join(path, filename)
    img = cv2.cvtColor(cv2.imread(str(file_path)), cv2.COLOR_BGR2RGB)
    fd = FaceDetect(device='cpu', detector='sfd')  # device = 'cpu' or 'cuda', detector = 'dlib' or 'sfd'
    face_info = fd.align(img)
    if face_info is not None:
        image_align, landmarks_align = face_info

        for i in range(landmarks_align.shape[0]):
            cv2.circle(image_align, center: (int(landmarks_align[i][0]), int(landmarks_align[i][1])), radius: 5, color: (255, 0,
            print(int(landmarks_align[i][0]), int(landmarks_align[i][1]))

        name, ext = os.path.splitext(filename)
        save_path = os.path.join(path, name + ".npy")
        np.save(str(save_path), landmarks_align)
        # torch.save(landmarks_align, '0001_01.npy')
        # cv2.imwrite(r'0001_01.png', cv2.cvtColor(image_align, cv2.COLOR_RGB2BGR))


if __name__ == '__main__':
    img_path = "../../DECA/train_data/vggface2/data/train_sample_2"
    for subdir, _, files in os.walk(img_path):
        for file in files:
            execute(subdir, file)
            pass
```



至此，训练所需要的数据准备完毕，最终数据文件如下，每一个人的每一张图片包含一个 npy 文件来记录该人脸的标记点坐标。

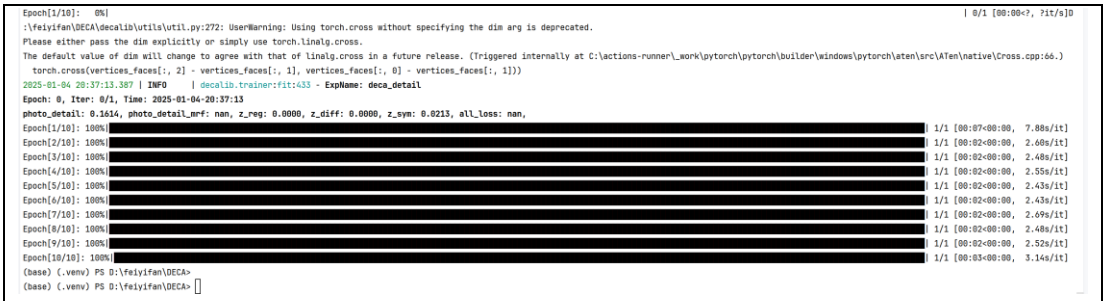
名称	修改日期	类型	大小
0001_01.jpg	2017/4/27 21:51	JPG 文件	40 KB
0001_01.npy	2025/1/3 19:58	NPY 文件	1 KB
0002_01.jpg	2017/4/27 21:51	JPG 文件	25 KB
0002_01.npy	2025/1/3 19:58	NPY 文件	1 KB
0003_01.jpg	2017/4/27 21:51	JPG 文件	48 KB
0003_01.npy	2025/1/3 19:58	NPY 文件	1 KB
0004_01.jpg	2017/4/27 21:51	JPG 文件	14 KB

训练时，每一个 Batch，DECA 都会随机选择 K 个图片。除了训练数据集外，还需要 resnet50_ft_weight.pkl 预训练模型来识别人脸，FLAME_albedo_from_BFM.npz 文件来从 BFM 模型转化为 FLAME 模型，FLAME2020 模型，所有需要的模型文件放在 data 文件夹下。



最终训练过程如下：

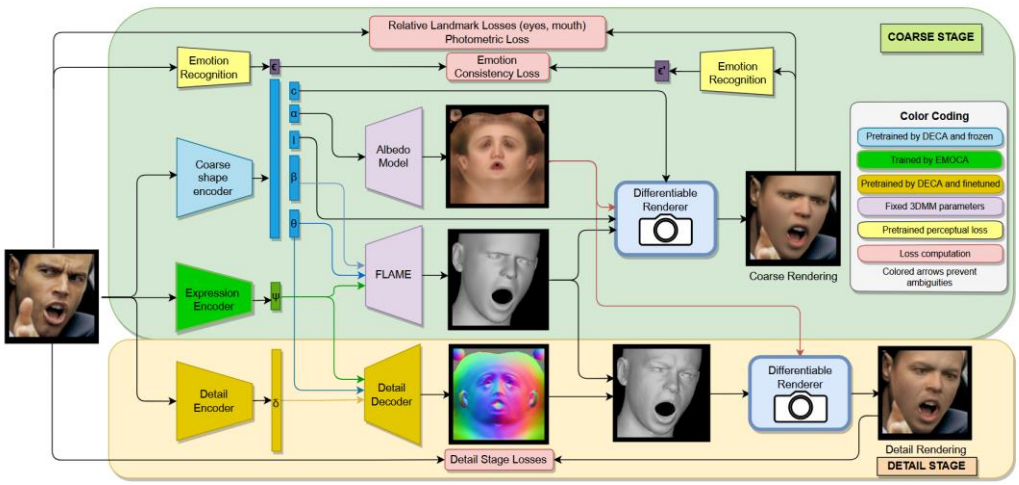
```
(base) (.venv) PS D:\feiyifan\DECA>
(base) (.venv) PS D:\feiyifan\DECA> python main_train.py --cfg configs/release_version/deca_detail.yml
Namespace(cfg='configs/release_version/deca_detail.yml', mode='train')

D:\feiyifan\DECA\.venv\lib\site-packages\torchvision\models\_utils.py:208: UserWarning: The parameter 'pretrained' is deprec
  warnings.warn(
D:\feiyifan\DECA\.venv\lib\site-packages\torchvision\models\_utils.py:223: UserWarning: Arguments other than a weight enum o
behavior is equivalent to passing `weights=None`.
  warnings.warn(msg)
creating the FLAME Decoder
D:\feiyifan\DECA\data\generic_model.pkl
trained model found. load ./train_data/deca_model.tar
D:\feiyifan\DECA\decalib\deca.py:91: FutureWarning: You are using `torch.load` with `weights_only=False` (the current defaul
s pickle data which will execute arbitrary code during unpickling (See https://github.com/pytorch/pytorch/blob/main/SECURITY
y` will be flipped to `True`. This limits the functions that could be executed during unpickling. Arbitrary objects will no
 via `torch.serialization.add_safe_globals`. We recommend you start setting `weights_only=True` for any use case where you d
d to this experimental feature.
  checkpoint = torch.load(model_path)
D:\feiyifan\DECA\.venv\lib\site-packages\pytorch3d\io\obj_io.py:550: UserWarning: Mtl file does not exist: D:\feiyifan\DECA\
  warnings.warn(f"Mtl file does not exist: {f}")
```



4. 讨论与改进思路

后续发表于 CVPR2022 的 EMOCA 方法在 DECA 的基础上进行了改进，着重提升了表情重建的能力，EMOCA 鼓励输入的图像和输出的渲染图像之间的情感相似性，使其能够捕捉并重建更为极端，细腻的面部表情。如下图所示，最顶部为重建目标图片，中间一栏为 DECA 重建，最下面一栏为 EMOCA 重建。可以看到 EMOCA 的结果有更多和表情相关的细节。

EMOCA 的原理如下图所示，和 DECA 相同，训练时也分为粗略重建编码器 Coarse Encoder 和细节重建编码器 Detail Coarse，但是与 DECA 不同的是，EMOCA 引入了一个额外的 Emotion Recognition 模块计算情感损失，如图中黄色模块。



Emotion Recognition 模块使用 ResNet-50 网络加一个全连层输出表情分类，该网络使用 AffectNet 数据集训练，是一个大规模的有情感标注的图片数据集。该模型训练完毕后，网络的输出为情感特征向量ε，该向量用来计算输入图片和重建图片的情感一致性损失，

该损失定义如下：

$$L_{\text{emo}} = d(e_I, e_{Re}) \quad d(e_1, e_2) = |e_1 - e_2|_2.$$

该损失项是 EMOCA 效果强于 DECA 的关键所在。最后模型评估对比如下，可以看出 EMOCA 在不同参数上的效果都优于 DECA。

Model	V-PCC ↑	V-CCC ↑	V-RMSE ↓	V-SAGR ↑	A-PCC ↑	A-CCC ↑	A-RMSE ↓	A-SAGR ↑	E-ACC ↑
EmoNet [86]	0.75	0.73	*0.32*	*0.80*	0.68	0.65	**0.29**	*0.78*	*0.68*
Deep3DFace [19]	0.75	0.73	0.33	*0.80*	0.66	0.65	0.31	0.78	0.65
ExpNet [15]	0.45	0.42	0.43	0.73	0.39	0.36	0.38	0.64	0.46
MGCNet [76]	0.71	0.69	0.35	*0.80*	0.59	0.58	0.34	0.77	0.60
3DDFA_V2 [34]	0.63	0.62	0.39	0.75	0.53	0.50	0.34	0.73	0.52
DECA [27]	0.70	0.69	0.36	0.76	0.59	0.58	0.33	0.74	0.59
DECA w/ details [27]	0.70	0.69	0.37	0.77	0.59	0.57	0.33	0.77	0.58
EMOCA (Ours)	**0.78**	**0.77**	**0.31**	**0.81**	*0.69*	*0.68*	*0.30*	*0.81*	*0.68*
EMOCA w/ details (Ours)	*0.77*	*0.76*	**0.31**	**0.81**	**0.70**	**0.69**	**0.29**	**0.83**	**0.69**

参考文献：

[1] Blanz V, Vetter T. A morphable model for the synthesis of 3D faces[M]//Seminal Graphics Papers: Pushing the Boundaries, Volume 2. 2023: 157-164.

[2] He, K., Zhang, X., Ren, S., & Sun, J. (2016). Deep residual learning for image recognition. Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 770-778.

[3] Nirkin Y, Masi I, Tuan A T, et al. On face segmentation, face swapping, and face perception[C]//2018 13th IEEE International Conference on Automatic Face & Gesture Recognition (FG 2018). IEEE, 2018: 98-105.

[4] Daněček R, Black M J, Bolkart T. Emoca: Emotion driven monocular face capture and animation[C]//Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition. 2022: 20311-20322.

[5] Wang Y, Tao X, Qi X, et al. Image inpainting via generative multi-column convolutional neural networks[J]. Advances in neural information processing systems, 2018, 31.

[6] Sanyal S, Bolkart T, Feng H, et al. Learning to regress 3D face shape and expression from an image without 3D supervision[C]//Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition. 2019: 7763-7772.

[7] Deng Y, Yang J, Xu S, et al. Accurate 3d face reconstruction with weakly-supervised learning: From single image to image set[C]//Proceedings of the IEEE/CVF conference on computer vision and pattern recognition workshops. 2019: 0-0.

[8] Gecer B, Ploumpis S, Kotsia I, et al. Ganfit: Generative adversarial network fitting for high fidelity 3d face reconstruction[C]//Proceedings of the IEEE/CVF conference on computer vision and pattern recognition. 2019: 1155-1164.