# Improving the Facial Expression Recognition and Its Interpretability via Generating Expression Pattern-map

Jing Zhang, Huimin Yu

**摘要**

面部表情识别专注于提取面部表情相关特征。本文从以下两个方面提出了一种新的面部表情建模方法：更准确地搜索表情相关区域，并增强表情特征的可辨性。为此，我们设计了一个包含三个子模块的模型：表情特征提取器（EFE）、表情掩膜细化器（EMR）和表情模式映射生成器（EPMG）。EFE 模块是提取表情特征并生成粗略注意力掩膜的主干网络，粗略指示表情相关区域。EMR 模块通过建模表情相关区域之间的关系，将掩膜细化为更精确的掩模，并生成掩模特征。EPMG 模块利用掩膜的特征来进一步建模融合和提取过程，从而获得用于识别的紧凑且有区别的表情显著嵌入特征，并生成表情模式图，我们提出了表情模式图的概念，它提供了表情特征的统一可视化，并提高了面部表情识别的可解释性。我们的模型在四个公共数据集（CK+、Oulu CASIA、RAF-DB、AffectNet）上进行了评估，并取得了与最先进技术相比的竞争性能。

**关键词：**面部表情识别；面部表情可视化；表情模式图生成器；深度神经网络

## 1 引言

面部表情识别在下一代人机交互应用中具有巨大潜力。总体而言，研究路线可分为两个分支：基于 AU 的检测和基于特征的分类。第一个分支基于面部动作编码系统（FACS）[1]，该系统是响应于寻找面部表情区域的需求而兴起的。通过将每个与表情相关的肌肉运动编码为一个动作单元（AU），面部表情可以通过这些 AU 的不同组合来表达，这吸引了研究人员学习不同 AU 之间的关系，以在检测任务中获得更高的准确性。然而，该方法需要专家来标记 AU，这是昂贵且耗时。另一方面，基于特征的分支也吸引了许多研究者。大多数人更喜欢利用卷积神经网络（CNN）进行特征提取和分类。然而，提取的特征将与许多不相关的因素相结合，例如面部身份信息或照明变化，这将给目标带来负面影响。因此，抑制与表情无关的因素并提取最具区别性的特征是至关重要的。为了消除用于识别的冗余信息，注意力机制因其突出表情区域的性质而受到关注。Xie 等人[2]，Chen 等人[3]利用注意力机制产生用于寻找表情显著区域的注意力掩模，并增强用于识别的特征。此外，注意力机制也与 AU 相关。已有的研究证明[4]，CNN 生成的注意力掩膜倾向于学习 AU 的分布，尽管它们没有明确的一对一对应关系。然而，在以前的工作中存在一些缺陷。首先，CNN 产生的注意力面具缺乏监督，这导致难以证明它是否准确。其次，它可能反映某一表情模式的表情相关区域之间的关系尚未得到充分利用。第三，掩蔽的表情特征不能直接可视化，这使得难以解释其可解释性。该论文提出了表情模式图的概念，这有助于提高识别精度，并提供了对不同面部身份不变的特定表情类别的统一可视化。它可以被视为衡量特征鲁棒性的标准，并提高了面部表情识别的可解释性。本文的贡献总结如下：提出了一个面部表情识别模型，该模型包含三个模块：表情特征提取器（EFE）、表情掩模细化器（EMR）和表情模式图生成器（EPMG）。

• 设计了 EMR 模块，通过对表情相关区域之间的关系进行建模来细化注意力掩码。

• 设计 EPMG 模块，通过建模表情显著特征的融合过程，为表情识别生成更紧凑、更具辨别力的嵌入。

• 提出了表情模式图的概念，它提供了表情特征的统一可视化，并提高了面部表情识别的可解释性。

## 2 相关工作

### 2.1 注意力机制

表情识别有其特殊性，因为某些面部区域提供了最具辨别力的信息。因此，越来越多的研究者意识到注意机制的重要性。Xie 等人[2]提出了一个由 SERD 和 MPVS-Net 组成的模型，该模型负责自适应地估计不同图像区域对面部表情识别的重要性，并将表情信息从无关变化中分离出来。Yang 等人[5]利用去表情残差学习来捕获网络中的表情信息，并将其用作表情相关特征。

### 2.2 身份解耦

Chen 等人[3]通过使用中性脸和相应表情脸之间的平均差异作为指导，结合先验领域知识边缘，以生成用于提取仅表情特征的注意力掩膜。为了获得更多有助于提高识别准确性的有区别的表情特征，一些研究人员倾向于将表情特征与身份特征解耦。Liu 等人[6]将身份因素与其他影响面部表情的因素区分开来。最近，消除视频域中的主题间差异引起了 Liu[7]等研究人员的兴趣，他们从残余帧中推断表情，并使用预训练的人脸识别网络提取身份因素。然而，大多数相关的工作都未能找到一种适当的方法来监督注意力面具以获得更准确的搜索能力，也无法可视化表情特征或解释表情识别的可解释性。因此，我们提出了缓解这些问题的模型。

## 3 本文方法

### 3.1 本文方法概述

此部分对本文将要复现的工作进行概述，如图 1所示：

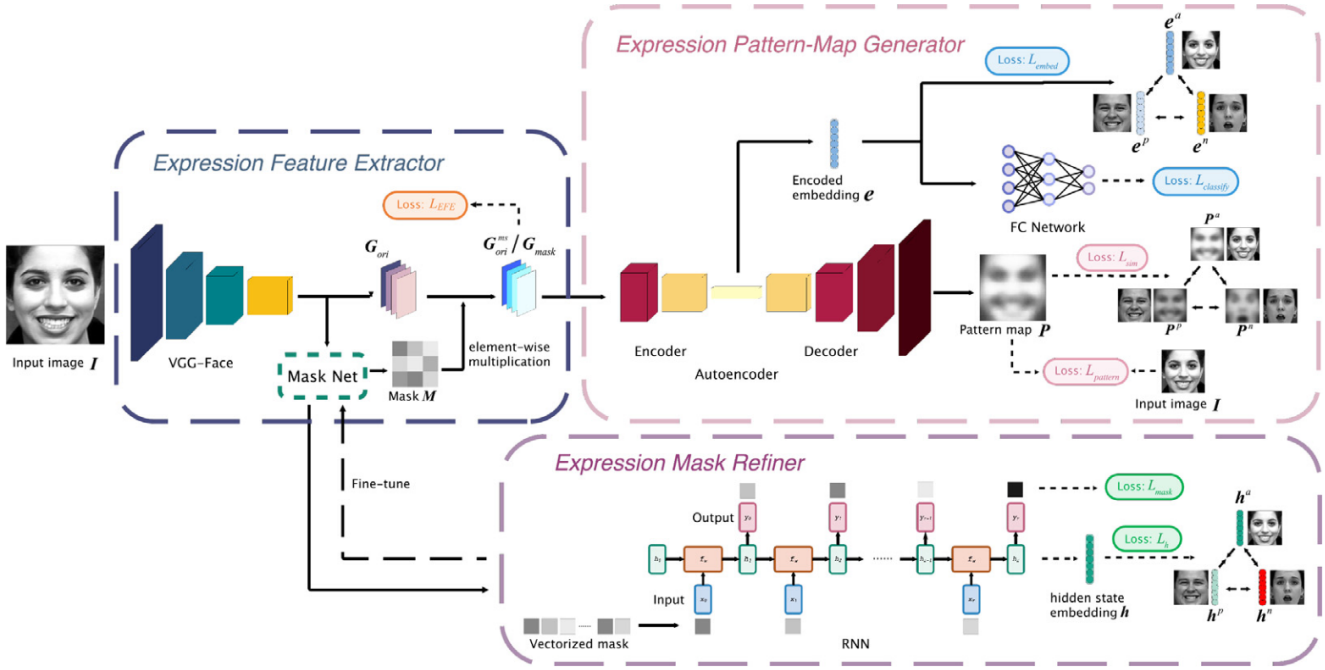


图 1: 方法示意图

### 3.2 特征提取模块

此部分采用 VGG-Face 预训练网络，通过 VGG-Face 获取特征图 $\mathbf{G}_{\text{ori}}$，然后将特征图输入到 Mask-net 获得掩膜 M, 通过如下公式得到 $\mathbf{G}_{\text{ori}}^{ms}$，最后利用交叉熵损失 $L_{EFE}$ 对 $\mathbf{G}_{\text{ori}}^{ms}$ 进行分类。

$$\mathbf{M} = f_a\left(\mathbf{W}_{\text{mask}} * \mathbf{G}_{\text{ori}} + \mathbf{b}_{\text{mask}}\right)$$
$$G_{\text{ori}}^{ms} = \mathbf{M} \odot \mathbf{G}_{\text{ori}}$$
$$L_{EFE} = -\frac{1}{n}\sum_{i=1}^{n} p_i \log q_i$$

$\mathbf{W}_{\text{mask}}$ 表示 mask-net 的卷积核权重，$\mathbf{b}_{\text{mask}}$ 表示 mask-net 的偏执。$p_i$ 是标签值，$q_i$ 是对应 $\mathbf{G}_{\text{ori}}^{ms}$ 的预测概率值。

### 3.3 表情掩膜精炼模块

通过设置损失函数，对掩膜 M 进行精炼，使其更能反应表情相关区域。设置 RNN 和相应的损失函数（L2 均方损失和三元组损失）进行精炼，公式如下。

$$\mathbf{h}_t = f_h\left(\mathbf{W}_h x_t + \mathbf{U}_h \mathbf{h}_{t-1} + \mathbf{b}_h\right)$$
$$y_t = f_y\left(\mathbf{W}_y \mathbf{h}_t + b_y\right)$$
$$L_{\text{mask}} = \frac{1}{n}\sum_{i=1}^{n} \|\mathbf{M} - \mathbf{M}'\|_2^2$$
$$L_h = \frac{1}{N}\sum_{i=1}^{N} \left[\|\mathbf{h}^a - \mathbf{h}^p\|_2^2 - \|\mathbf{h}^a - \mathbf{h}^n\|_2^2 + \alpha\right]_+$$
$$L_{EMR} = \lambda_1 L_{EFE} + \lambda_2 L_{\text{mask}} + \lambda_3 L_h$$

$h_t$ 是 RNN 隐藏层特征向量，$M'$ 是 RNN 输出后的序列预测值进行 reshap 后的预测掩膜 M。$L_{EMR}$ 损失是该子模块两个损失加上一模块的分类损失构成。

### 3.4 表情模式图生成模块

该模块通过编码-解码网络进行学习，设置四个损失函数进行正则化，编码器的损失函数有交叉熵损失和三元组损失，公式如下。

$$L_{\text{embed}} = \frac{1}{N}\sum_{i=1}^{N} \left[\|\mathbf{e}^a - \mathbf{e}^p\|_2^2 - \|\mathbf{e}^a - \mathbf{e}^n\|_2^2 + \gamma\right]_+$$
$$L_{\text{classify}} = -\frac{1}{n}\sum_{i=1}^{n} p_i' \log q_i'$$
$$L_{\text{encoder}} = \lambda_6 L_{\text{embed}} + \lambda_7 L_{\text{classify}}$$

$e$ 为编码后的特征向量，编码器的损失由 $L_{\text{embed}}$ 和 $L_{\text{sim}}$ 构成。

解码器的损失函数有三元组损失和均方损失，公式如下。

$$L_{\text{pattern}} = \frac{1}{n}\sum_{i=1}^{n} \|\mathbf{P} - \mathbf{I}\|_2^2$$
$$L_{\text{sim}} = \frac{1}{N}\sum_{i=1}^{N} \left[\|\mathbf{P}^a - \mathbf{P}^p\|_2^2 - \|\mathbf{P}^a - \mathbf{P}^n\|_2^2 + \beta\right]_+$$
$$L_{\text{decoder}} = \lambda_4 L_{\text{pattern}} + \lambda_5 L_{\text{sim}}$$

其中 P 为重构的表情模式图，I 为输入的图片。解码器的损失由 $L_{\text{pattern}}$ 和 $L_{\text{sim}}$ 构成。

最后，该模块的总损失如下所示。

$$L_{EPMG} = L_{\text{encoder}} + L_{\text{decoder}}$$

# 4 复现细节

## 4.1 无开源代码

该篇论文无开源代码可以参考，在复现过程中，按照论文所提及的方法流程进行复现。首先是论文提及的特殊三元组损失的实现，如下所示。

```
# Definition TripletMarginLoss : hard TripletMarginLoss
def vector_triplet_loss(embeding,labels,nums=7,alpha=0.2):
    # tensor is needed
    a = torch.FloatTensor([alpha]).cuda()
    loss = torch.FloatTensor([0.]).cuda()
    # get batchsize
    batch_size = labels.shape[0]
    for i in range(0, nums):
        anchor_list = []
        negative_list = []
        for L, E in zip(labels, embeding):
            if int(L.item()) == i:
                anchor_list.append(E)
            elif int(L.item()) != i:
                negative_list.append(E)
        if len(anchor_list) != 0 and len(negative_list) != 0:
            for anchor in anchor_list:
                max_list = []
                min_list = []
                for positive in anchor_list:
                    max_list.append((anchor - positive).square().sum())
                for negative in negative_list:
                    min_list.append((anchor - negative).square().sum())

                dap = max_list[0]
                for i in max_list:
                    if float(i.item()) > float(dap.item()):
                        dap = i
                dan = min_list[0]
                for j in min_list:
                    if float(j.item()) < float(dan.item()):
                        dan = j

                if (dap.item() - dan.item() + a.item()) > 0.:
                    loss += (dap - dan + a)

    # tensor can backpropagation
    return loss / batch_size
```

图 2: Vector triplet loss

```python
def matrix_triplet_loss(P,labels,nums=7, alpha=0.2):
    a = torch.FloatTensor([alpha]).cuda()
    loss = torch.FloatTensor([0.]).cuda()
    batch_size = labels.shape[0]
    for i in range(0, nums):
        anchor_list = []
        negative_list = []
        for label,Pic in zip(labels,P):
            if int(label) == i:
                anchor_list.append(Pic)
            else:
                negative_list.append(Pic)

        if len(anchor_list) > 0 and len(negative_list) > 0:
            for anchor in  anchor_list:
                P_max_list = []
                N_min_list = []
                for positive in anchor_list:
                    P_max_list.append((anchor - positive).square().sum())
                for negative in negative_list:
                    N_min_list.append((anchor - negative).square().sum())

                dap = P_max_list[0]
                for i in P_max_list:
                    if float(i.item()) > float(dap.item()):
                        dap = i
                dan = N_min_list[0]
                for j in N_min_list:
                    if float(j.item()) < float(dan.item()):
                        dan = j

                if (dap.item() - dan.item() + a.item()) > 0.:
                    loss += (dap - dan + a)

    return loss / batch_size
```

图 3: Matrix triplet loss

随机采样的代码如 4所示。

```python
class triplet_Dataset_train(torch.utils.data.Dataset):
    def __init__(self,root, file_list,transform=None, loader=img_loader):
        self.root = root
        self.transform = transform
        self.loader = loader
        '''...'''

        #Classified storage of data
        image_list = []
        for i in range(0,7):
            image_list.append([])

        with open(file_list) as f:
            img_label_list = f.read().splitlines()
        for info in img_label_list:
            image_path, label_name= info.split(' ')
            image_list[int(label_name)].append(image_path)


        self.image_list = image_list
        self.class_nums = 7

    def __getitem__(self, index):
        # Index by category
        label = index % 7
        img_path = self.image_list[label][random.randint(0,len(self.image_list[label])-1)]
        img = self.loader(os.path.join(self.root, img_path))
        if self.transform is not None:
            tensor_img = self.transform(img)
        return tensor_img, label

    def __len__(self):
        length = 0
        for i in range(0,7):
            length += len(self.image_list[i])
        return length
```

图 4: random sampling

模型训练的第一阶段的代码如 5所示。

```python
# first step
EFE_model.train()
# Unseal the last frozen model
for name, param in EFE_model.named_parameters():
    param.requires_grad = True
index = 0
for batch in train_loader:
    inputs, targets = batch
    if use_cuda:
        inputs, targets = inputs.cuda(), targets.cuda(non_blocking=True)

    optimizer_first.zero_grad()
    _, _, pre_EFE = EFE_model(inputs)

    L_EFE = criterion_ce(pre_EFE, targets).mean()
    index += 1
    loss_EFE += L_EFE.item()
    L_EFE.backward()
    # against gradient explosion
    torch.nn.utils.clip_grad_norm(EFE_model.parameters(), args.EFE_GT)
    optimizer_first.step()

# The average sample loss of one iteration over the entire training data set
loss_EFE = loss_EFE / index
```

图 5: step one

模型训练的第二阶段的代码如 6所示。

```
# Loading triples
triplet_set_iter = iter(triplet_set_loader)
# second step
EMR_model.train()
# Freeze the first 5 layers of the VGG
five_layers = 0
for name, param in EFE_model.named_parameters():
    if five_layers < 10:
        five_layers += 1
        param.requires_grad = False
index = 0
# random sampling
for batch_idx in range(args.train_iteration):
    try:
        inputs, targets = triplet_set_iter.next()
    except:
        triplet_set_iter = iter(triplet_set_loader)
        inputs, targets = triplet_set_iter.next()
    if use_cuda:
        inputs, targets = inputs.cuda(), targets.cuda(non_blocking=True)

    optimizer_second.zero_grad()
    mask, _ ,pre_EFE = EFE_model(inputs)
    Hn, output = EMR_model(mask)
    L_EFE = criterion_ce(pre_EFE,targets).mean()
    L_h = criterion_triplet_vector(Hn,targets)
    # change mask shape
    mask = mask.view(mask.shape[0],7,7)
    L_mask = mseloss(mask,output).sum(dim=(1,2)).mean()
    L_EMR = 2 * L_EFE + 100 * L_mask + 100 * L_h
    loss_EMR += L_EMR.item()
    index += 1
    L_EMR.backward()
    torch.nn.utils.clip_grad_norm(EFE_model.parameters(), args.EFE_GT)
    torch.nn.utils.clip_grad_norm(EMR_model.parameters(), args.EMR_GT)
    optimizer_second.step()
loss_EMR = loss_EMR / index
```

图 6: step two

模型训练的第三阶段的代码如 7所示。

```
#third step
EPMG_model.train()
# Freeze the EFE module
for name, param in EFE_model.named_parameters():
        param.requires_grad = False
index = 0
for batch_idx in range(args.train_iteration):
    try:
        inputs, targets = triplet_set_iter.next()
    except:
        triplet_set_iter = iter(triplet_set_loader)
        inputs, targets = triplet_set_iter.next()
    if use_cuda:
        inputs, targets = inputs.cuda(), targets.cuda(non_blocking=True)
    optimizer_third.zero_grad()
    _, Gmask,_ = EFE_model(inputs)
    e, c_e, Pic = EPMG_model(Gmask)
    L_embed = criterion_triplet_vector(e,targets)
    L_classify = criterion_ce(c_e, targets).mean()
    L_encoder = L_embed + L_classify
    Pic = Pic.view(Pic.shape[0], 224, 224)
    L_sim = criterion_triplet_matrix(Pic,targets)
    I = gray(inputs)
    I = I.view(I.shape[0],224,224)
    L_pattern = mseloss(Pic,I).sum(dim=(1,2))
    L_pattern = L_pattern.mean()
    L_decoder = 60 * L_pattern + 40 * L_sim
    L_EPMG = L_encoder + L_decoder
    loss_EPMG += L_EPMG.item()
    index += 1
    L_EPMG.backward()
    torch.nn.utils.clip_grad_norm(EPMG_model.parameters(), args.EPMG_GT)
    optimizer_third.step()
loss_EPMG = loss_EPMG / index
return  loss_EFE ,loss_EMR ,loss_EPMG
```

图 7: step three

### 4.2 实验环境搭建

Linux 系统，pytorch 深度学习框架，3090 显卡。模型的超参数设置如下：CK+ 和 AffectNet 的超参数为 λ1=2，λ2=λ3=1×10 2，λ6=λ7=1，α=β=γ=0.2，λ4=λ5=10 0，而 Oulu CASIA 和 RAF-DB 的超参数则为 λ4=60，λ5=40。形成三重 loss 训练批的策略是从每个表达类别中随机选取 t 个图像，其中，CK+ 和 Oulu CASIA 数据集的 t=8，RAF-DB 和 AffectNet 数据集的 t=30，因为这些数据集的规模不同。训练阶段可以分为三个步骤：首先，随机遍历数据库中的所有训练样本，以微调 VGG-Face，并训练掩码网络；其次，根据上述策略形成三重批次，冻结 VGG-Face 中 Conv-1 到 Conv-5 的权重，联合训练 EFE 和 EMR 以细化掩模；第三，将 EFE 和 EPMG 结合在一起，同时恢复模型的架构。使用三元组批次来训练 EPMG。并选择 SGD 作为优化器，每个步骤的学习率分别为 0.0 01、0.0 0 01 和 0.0 01。模型框架如 8所示。

| | | |
|---|---|---|
| VGG-Face | Same settings as Parkhi et al. [13] | |
| Mask Net | Conv($1 \times 1 \times 512 \times 1$), Sigmoid | |
| RNN | Hidden layer | FC($1 \times 16$), Tanh |
| | Output layer | FC($16 \times 1$), Sigmoid |
| Autoencoder | Encoder | Conv($4 \times 4 \times 512 \times 256$), ReLU |
| | | Conv($4 \times 4 \times 256 \times 128$), Sigmoid |
| | Decoder | De-Conv($4 \times 4 \times 128 \times 64$), ReLU |
| | | De-Conv($4 \times 4 \times 64 \times 32$), ReLU |
| | | De-Conv($4 \times 4 \times 32 \times 16$), ReLU |
| | | De-Conv($4 \times 4 \times 16 \times 8$), ReLU |
| | | De-Conv($4 \times 4 \times 8 \times 4$), ReLU |
| | | De-Conv($4 \times 4 \times 4 \times 2$), ReLU |
| | | De-Conv($4 \times 4 \times 2 \times 1$), Sigmoid |
| FC Network | FC($128 \times 64$), ReLU | |
| | FC($64 \times K$) | |
| | Softmax | |

图 8: Model architecture

### 4.3 实现效果展示

论文的效果展示如 9所示。


图 9: 掩膜展示

## 5 实验结果分析

本部分对实验所得结果进行分析，详细对实验内容进行说明，实验结果进行描述并分析。

| CK+ | | Oulu-CASIA | | RAF-DB | | AffectNet | |
|---|---|---|---|---|---|---|---|
| Methods | Accuracy | Methods | Accuracy | Methods | Accuracy | Methods | Accuracy |
| DCMA-CNNs [8] | 93.46 | DCD [10] | 84.80 | PG-CNN [22] | 83.27 | PG-CNN [22] | 55.33 |
| DAM [3] | 95.88 | DCPN [23] | 86.23 | ALT [24] | 84.50 | IPA2LT [25] | 57.31 |
| GA-CNN [26] | 96.4 | SNA [27] | 87.60 | GA-CNN [26] | 85.07 | GA-CNN [26] | 58.78 |
| DeRL [11] | 97.3 | DeRL [11] | 88.00 | SEP-Loss [28] | 86.38 | SEP-Loss [28] | 58.89 |
| LBVCNN [9] | 97.38 | IDFERM [2] | 88.25 | IPA2LT [25] | 86.77 | ESR-9 [29] | 59.30 |
| G2-VER [30] | 97.40 | DDL [31] | 88.26 | DDA-Loss [32] | 86.90 | LDER [33] | 60.53 |
| Ours | **98.06** | Ours | **89.05** | Ours | **87.10** | Ours | **62.10** |

图 10: 实验结果

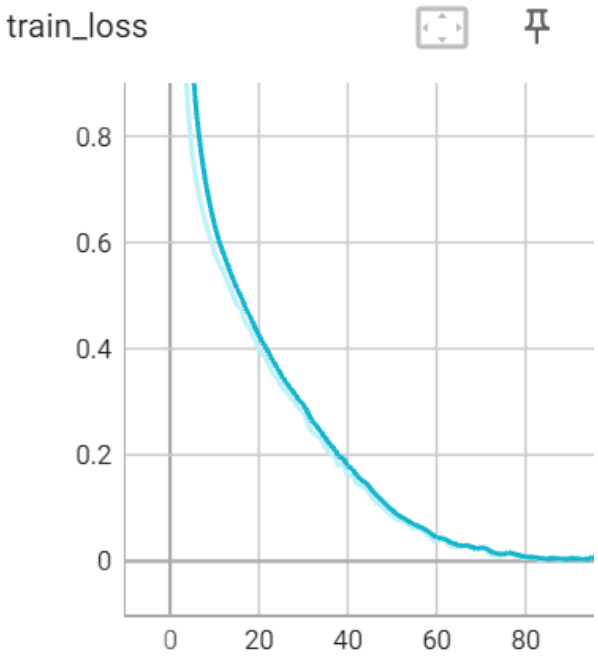


图 11: 训练的 loss 下降图

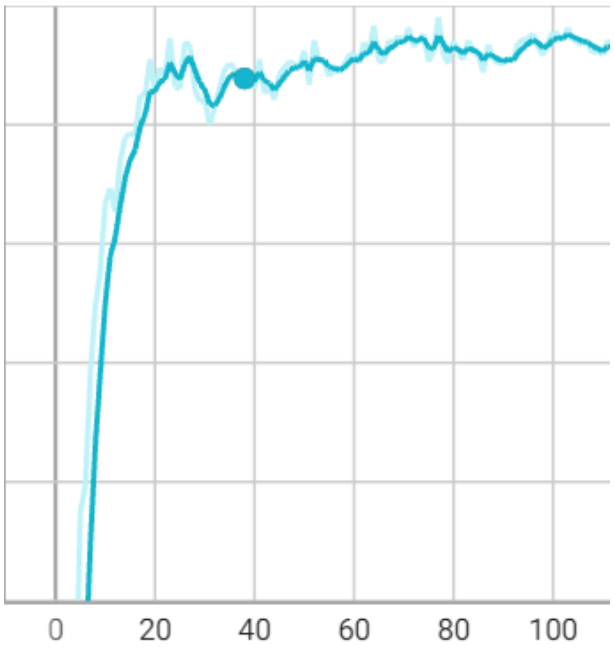


图 12: 测试精度图

# 6 总结与展望

因为本篇论文没有开源代码，故实现的效果相比于论文所给出的精度有几个点的差距，但是复现完的模型相比于基础模型是有提升的，所有该篇论文提出的模型对表情识别的可解释性是有帮助的，对于本次的复现，首先是提升了我的代码能力，也对我的研究方向更加清楚，对自己的研究目标更加明确，收获了很多。

# 参考文献


[1] EKMAN P, FRIESEN W V. Facial action coding system: a technique for the measurement of facial movement[C]//. 1978.

[2] XIE S, HU H, WU Y. Deep multi-path convolutional neural network joint with salient region attention for facial expression recognition[J]. Pattern Recognit., 2019, 92: 177-191.

[3] CHEN Y, WANG J, CHEN S, et al. Facial Motion Prior Networks for Facial Expression Recognition [J/OL]. CoRR, 2019, abs/1902.08788. arXiv: 1902.08788. http://arxiv.org/abs/1902.08788.

[4] WANG C, PENG M, BI T, et al. Micro-Attention for Micro-Expression recognition[J/OL]. CoRR, 2018, abs/1811.02360. arXiv: 1811.02360. http://arxiv.org/abs/1811.02360.

[5] YANG H, CIFTCI U, YIN L. Facial Expression Recognition by De-expression Residue Learning[C]// 2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition. 2018: 2168-2177. DOI: 10.1109/CVPR.2018.00231.

[6] LIU X, VIJAYA KUMAR B, JIA P, et al. Hard negative generation for identity-disentangled facial expression recognition[J/OL]. Pattern Recognition, 2019, 88: 1-12. https://www.sciencedirect.com/science/article/pii/S0031320318303819. DOI: https://doi.org/10.1016/j.patcog.2018.11.001.

[7] LIU X, JIN L, HAN X, et al. Identity-aware Facial Expression Recognition in Compressed Video[C]// 2020 25th International Conference on Pattern Recognition (ICPR). 2021: 7508-7514. DOI: 10.1109/ICPR48806.2021.9412820.