疲劳驾驶分类-Drowsiness Driven Classification

说明

文件：

- gain_mean_std.py：获取均值和标准差
- pre_processing.txt：预处理文件（内置均值和标准差）
- this.py：训练、验证模型的文件
- predict.py：输入图片进行预测得到对应类别
- my_utils.py：自定义的工具包文件(绘图、模型微调等)
- requirements.txt：依赖的环境包

文件夹：

- data：使用的数据集
- model：自定义的本地模型(VGG、ResNet、GooLeNet)
- weights：训练好的模型权重
- results：模型训练的结果可视化
- pic_predict：用来预测的图片
- vit_model_drowsy_driven：使用 vit 预训练模型进行训练

数据集：

- **训练集：**共2836张（80%用于训练，20%用于验证）
- **测试集：**共101张

标签：

- 'closed'（闭眼）: 0
- 'no_yawn'（没有打哈欠）: 1
- 'open'（睁眼）: 2
- 'yawn'（打哈欠）: 3

模型

- 整合包：**https://pan.quark.cn/s/70536c418824**
- 数据集：**https://pan.quark.cn/s/5a4b1140d306**
- VGG16：**https://pan.quark.cn/s/46a1d3d1f29b**
- VGG19：**https://pan.quark.cn/s/c13604666d41**
- ResNet65：**https://pan.quark.cn/s/24ec94d0676a**
- VGG16_Pre：**https://pan.quark.cn/s/11a66e0e5d4d**
- ResNet50_Pre：**https://pan.quark.cn/s/33c05f29d7a1**
- GoogLeNet_Pre：**https://pan.quark.cn/s/e1c1ab0fd170**
- VIT16 预训练模型（放到 pretrain 文件夹内）：**https://pan.quark.cn/s/cf0516956931**
- VIT16 训练好的模型（放到 checkpoints 文件夹内）：**https://pan.quark.cn/s/703da7c6d738**

运行

数据集：将数据集下载好解压后放到 `data` 文件夹内

VGG / ResNet65 / GoogLeNet 运行方式：

将训练好的模型文件下载好放到 `weights` 文件夹内

1、下载所需的环境包

```
pip install -r requirements.txt
```

2、进入项目所在路径，运行训练脚本

```
python this.py
```

3、运行预测脚本

```
python predict.py
```

VIT16 运行方式：

将预训练模型文件下载好放到 `pretrain` 文件夹内

将训练好的模型文件下载好放到 `checkpoints` 文件夹内

配置好环境后，进入项目所在路径，依次执行：

```
python train.py
python test.py
python predict.py
```

一. 从头训练

通用超参数：

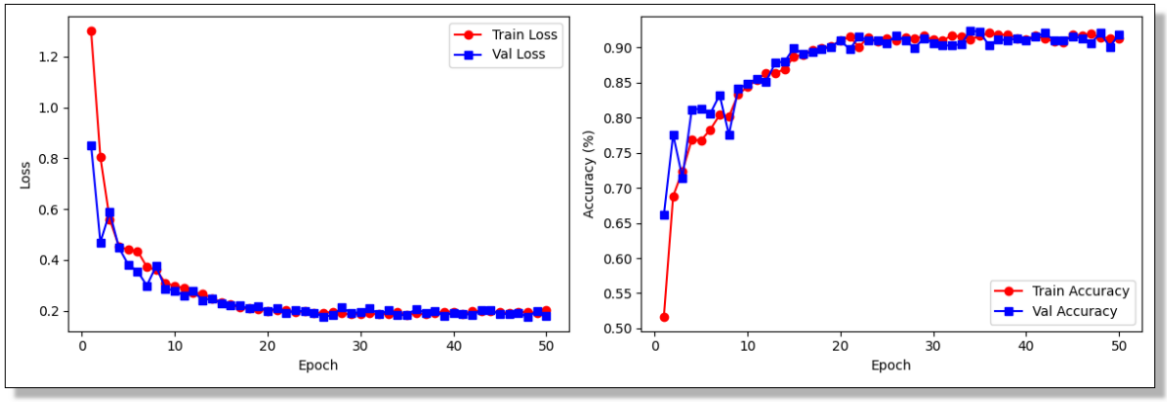
- LR = 1e-2 / 3e-2
- EPOCHS = 50
- BATCH_SIZE = 32
- optimizer：Adam / SGD
- scheduler：StepLR / LambdaLR

1.1 VGG

VGG16

```
data_transforms = {
      'train': transforms.Compose([
            transforms.Resize(256),
            transforms.CenterCrop(224),
            transforms.RandomHorizontalFlip(), # 水平翻转，因为不改变闭眼、打哈欠等状态
```

```
            transforms.ColorJitter(brightness=0.05, contrast=0.05, saturation=0.05, hue=0.02), # 色彩抖动应当谨慎使用，以保持面部特征的真实性
            transforms.ToTensor(),
            transforms.Normalize(mean=[0.151, 0.136, 0.129], std=[0.058, 0.051, 0.048]) # 需自己计算
        ]),
        'test': transforms.Compose([
            transforms.Resize(256),
            transforms.CenterCrop(224),
            transforms.ToTensor(),
            transforms.Normalize([0.151, 0.136, 0.129], [0.058, 0.051, 0.048])
        ]),
    }
    optimizer = optim.SGD(model.parameters(),lr=LR,momentum=0.9,weight_decay=5e-4)
    # 法1：学习率每8个epoch衰减成原来的1/10
    # scheduler = optim.lr_scheduler.StepLR(optimizer, step_size=8, gamma=0.1)
```

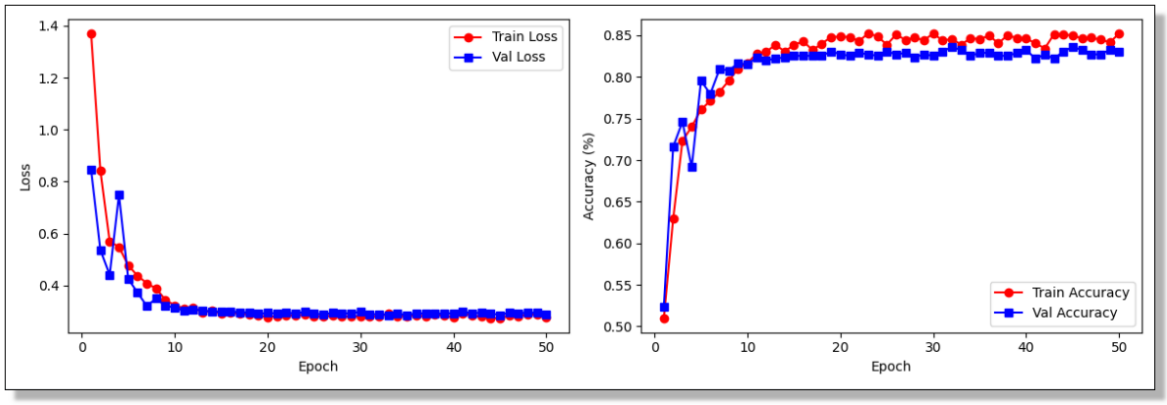



```
---------- Train Start 第 34 轮 ----------
Epoch [34]: 100%|                                        | 71/71 [00:45<00:00,  1.56it/s, acc=0.911, loss=0.323]
[Epoch]: [34] | train loss: 0.1940 | train accuracy: 0.9114
---------- Train Finished 第 34 轮 ----------
------ Val Start -----
下一轮 Optimizer learning rate : 0.0000010
[Epoch]: [34] | val loss: 0.1835 | val accuracy: 0.9242
------ Val Finished -----
训练和验证耗时:  31.00min 7.1844s
```

在第 34 个EPOCH时，验证集准确率达到最佳：0.9242，测试集为 0.9189

```
---------- Train Start 第 50 轮 ----------
Epoch [50]: 100%|                                        | 71/71 [00:46<00:00,  1.54it/s, acc=0.913, loss=0.23]
[Epoch]: [50] | train loss: 0.2029 | train accuracy: 0.9132
---------- Train Finished 第 50 轮 ----------
------ Val Start -----
下一轮 Optimizer learning rate : 0.0000000
[Epoch]: [50] | val loss: 0.1788 | val accuracy: 0.9189
------ Val Finished -----
训练和验证耗时:  45.00min 18.5486s
------ Test Start -----
[Epoch]: [50] | test loss: 0.2124 | test accuracy: 0.9010
```

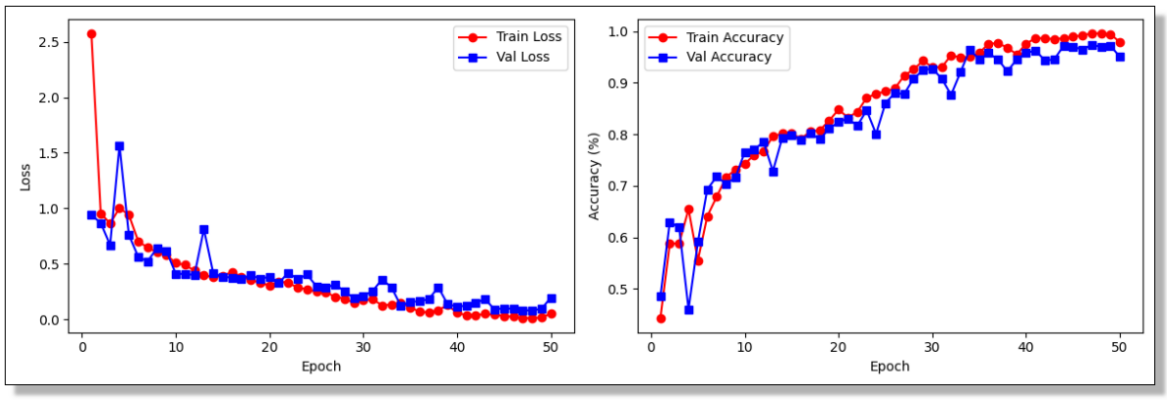
VGG19




```
---------- Train Start 第 50 轮 ----------
Epoch [50]: 100%|                                        | 71/71 [00:48<00:00,  1.46it/s, acc=0.852, loss=0.28]
[Epoch]: [50] | train loss: 0.2782 | train accuracy: 0.8519
---------- Train Finished 第 50 轮 ----------
------ Val Start -----
下一轮 Optimizer learning rate : 0.0000000
[Epoch]: [50] | val loss: 0.2898 | val accuracy: 0.8307
------ Val Finished -----
训练和验证耗时:  48.00min 34.9521s
------ Test Start -----
[Epoch]: [50] | test loss: 0.2377 | test accuracy: 0.8416
```

1.2 ResNet65

```
optimizer = optim.SGD(model.parameters(),lr=LR,momentum=0.9,weight_decay=5e-4)
# 学习率衰减通常在训练集使用
# 法2：预热和余弦退火策略
warmup_epochs = 5   # 预热阶段的epoch数
scheduler = optim.lr_scheduler.LambdaLR(optimizer, lr_lambda=lambda
epoch:warmup_cosine_schedule(epoch,warmup_epochs, EPOCHS))
```




```
---------- Train Start 第 47 轮 ----------
Epoch [47]: 100%|                                        | 71/71 [00:40<00:00,  1.73it/s, acc=0.996, loss=0.00518]
下一轮 Optimizer learning rate : 0.0298540
[Epoch]: [47] | train loss: 0.0156 | train accuracy: 0.9960
---------- Train Finished 第 47 轮 ----------
------ Val Start -----
[Epoch]: [47] | val loss: 0.0828 | val accuracy: 0.9735
```

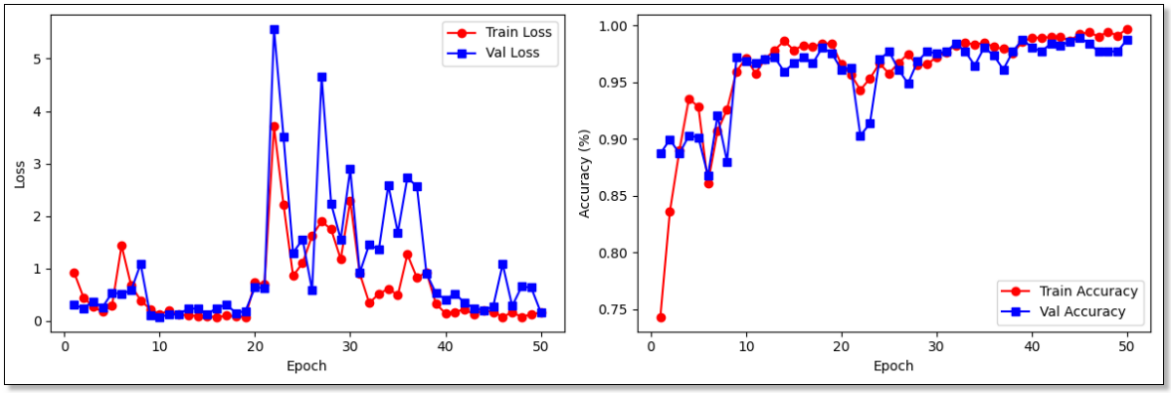
在第 47 个EPOCH时，验证集准确率达到最佳：0.9960，测试集为 0.9307

```
---------- Train Start 第 50 轮 ----------
Epoch [50]: 100%|| 71/71 [00:39<00:00,  1.79it/s, acc=0.979, loss=0.0381]
下一轮 Optimizer learning rate : 0.0035093
[Epoch]: [50] | train loss: 0.0541 | train accuracy: 0.9788
---------- Train Finished 第 50 轮 ----------
------ Val Start -----
[Epoch]: [50] | val loss: 0.1965 | val accuracy: 0.9506
------ Val Finished -----
训练和验证耗时:  40.00min 7.1530s
------ Test Start -----
[Epoch]: [50] | test loss: 0.3757 | test accuracy: 0.9307
```

二. 迁移学习

拿大数据集训练好的模型作为预训练，微调全连接层在小数据集上能获得比较好的结果

2.1 VGG16_Pre

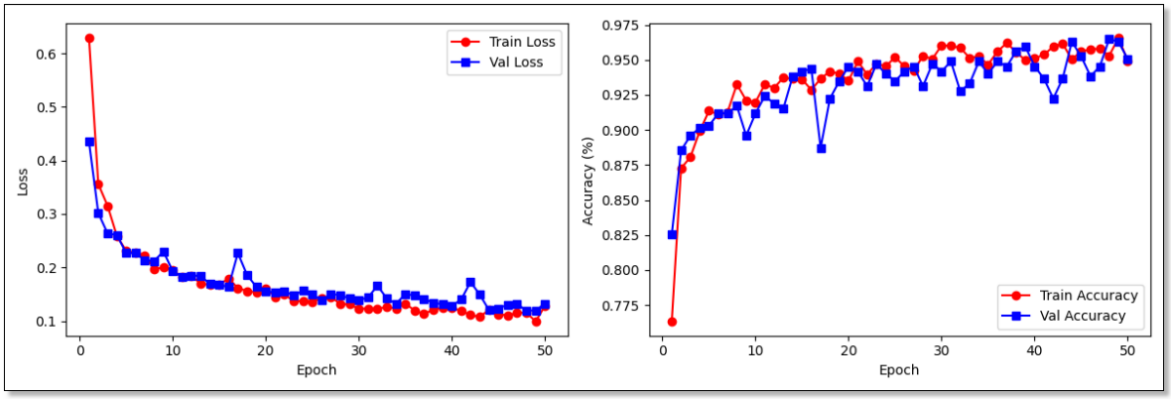



```
---------- Train Start 第 44 轮 ----------
Epoch [44]: 100%|| 71/71 [00:33<00:00,  2.13it/s, acc=0.986, loss=1.62e-6]
下一轮 Optimizer learning rate : 0.0002204
[Epoch]: [44] | train loss: 0.2085 | train accuracy: 0.9863
---------- Train Finished 第 44 轮 ----------
------ Val Start -----
[Epoch]: [44] | val loss: 0.2098 | val accuracy: 0.9859
```

在第 44 个EPOCH时，验证集准确率达到最佳：0.9859，测试集为 0.9505

```
---------- Train Start 第 50 轮 ----------
Epoch [50]: 100%|| 71/71 [00:34<00:00,  2.08it/s, acc=0.997, loss=0]
下一轮 Optimizer learning rate : 0.0001170
[Epoch]: [50] | train loss: 0.1587 | train accuracy: 0.9969
---------- Train Finished 第 50 轮 ----------
------ Val Start -----
[Epoch]: [50] | val loss: 0.1674 | val accuracy: 0.9877
------ Val Finished -----
训练和验证耗时:  35.00min 27.8621s
------ Test Start -----
[Epoch]: [50] | test loss: 2.5513 | test accuracy: 0.9505
```

2.2 ResNet50_Pre

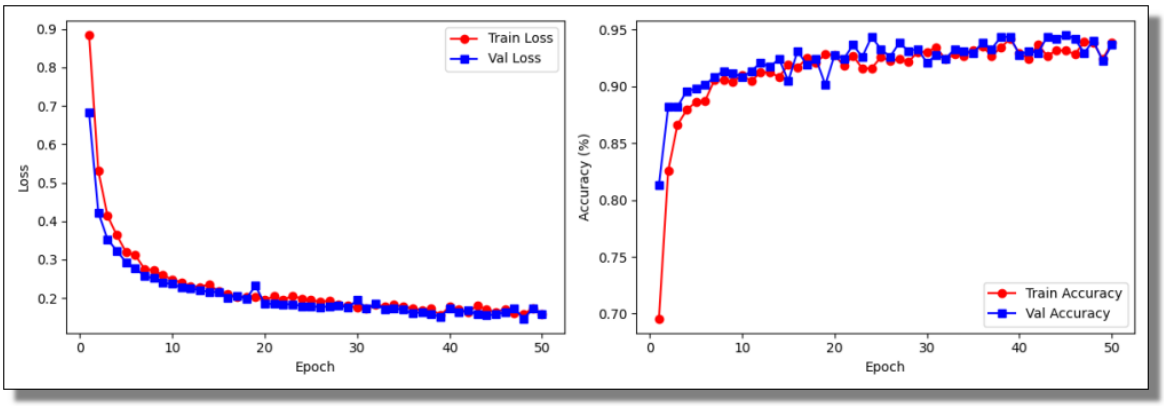



```
---------- Train Start 第 48 轮 ----------
Epoch [48]: 100%|                                  | 71/71 [00:29<00:00,  2.40it/s, acc=0.953, loss=0.0976]
下一轮 Optimizer learning rate : 0.0006873
[Epoch]: [48] | train loss: 0.1159 | train accuracy: 0.9528
---------- Train Finished 第 48 轮 ----------
------ Val Start -----
[Epoch]: [48] | val loss: 0.1195 | val accuracy: 0.9647
```

在第 48 个EPOCH时，验证集准确率达到最佳：0.9647，测试集为 0.9604

```
---------- Train Start 第 50 轮 ----------
Epoch [50]: 100%|                                  | 71/71 [00:29<00:00,  2.38it/s, acc=0.949, loss=0.0779]
下一轮 Optimizer learning rate : 0.0001170
[Epoch]: [50] | train loss: 0.1283 | train accuracy: 0.9493
---------- Train Finished 第 50 轮 ----------
------ Val Start -----
[Epoch]: [50] | val loss: 0.1324 | val accuracy: 0.9506
------ Val Finished -----
训练和验证耗时:  31.00min 56.7815s
------ Test Start -----
[Epoch]: [50] | test loss: 0.1559 | test accuracy: 0.9604
```

2.3 GoogLeNet_Pre




```
---------- Train Start 第 45 轮 ----------
Epoch [45]: 100%|                                  | 71/71 [00:26<00:00,  2.68it/s, acc=0.932, loss=0.106]
下一轮 Optimizer learning rate : 0.0000302
[Epoch]: [45] | train loss: 0.1638 | train accuracy: 0.9317
---------- Train Finished 第 45 轮 ----------
------ Val Start -----
[Epoch]: [45] | val loss: 0.1572 | val accuracy: 0.9453
------ Val Finished -----
训练和验证耗时:  24.00min 33.4082s
```

```
---------- Train Start 第 50 轮 ----------
Epoch [50]: 100%|                    | 71/71 [00:27<00:00,  2.62it/s, acc=0.938, loss=0.0711]
下一轮 Optimizer learning rate : 0.0001170
[Epoch]: [50] | train loss: 0.1589 | train accuracy: 0.9383
---------- Train Finished 第 50 轮 ----------
------ Val Start -----
[Epoch]: [50] | val loss: 0.1575 | val accuracy: 0.9365
------ Val Finished -----
训练和验证耗时：  27.00min 21.3209s
------ Test Start -----
[Epoch]: [50] | test loss: 0.1283 | test accuracy: 0.9505
```

2.4 VIT16_Pre

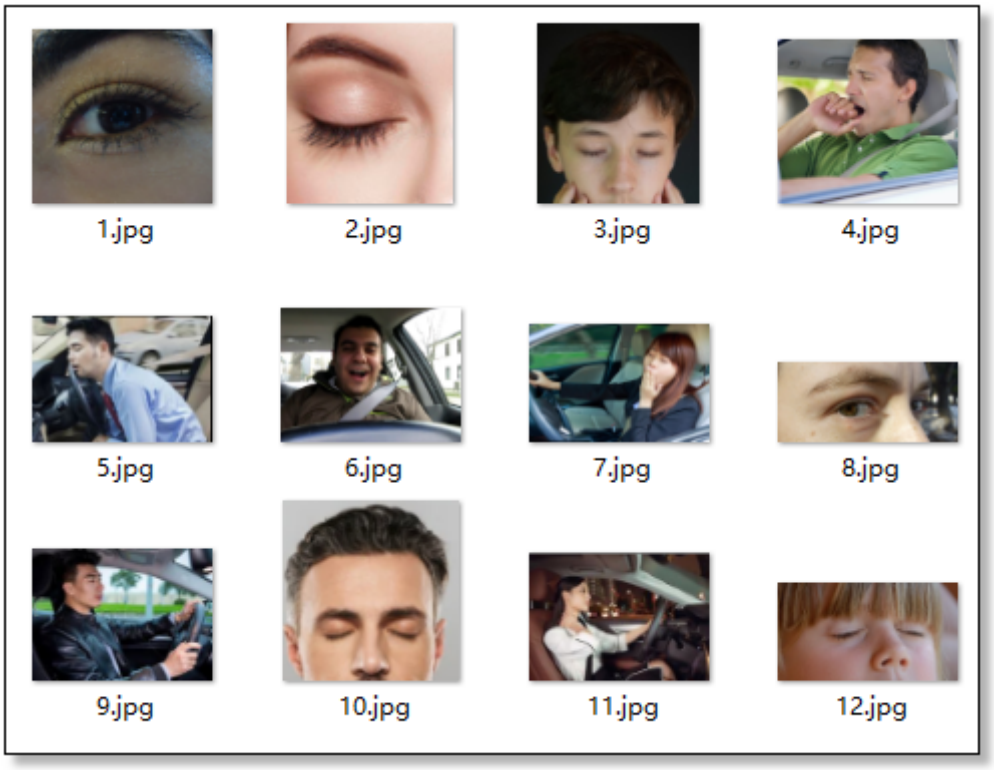
```
Training (3195 / 3550 Steps) (loss=0.10381): 100%|| 71/71 [00:53<00:00,  1.34it/s]
Validating: 100%|| 36/36 [00:04<00:00,  7.98it/s]|| 70/71 [00:48<00:00,  1.49it/s]
accuracy: 0.9735449735449735:04<00:00,  8.75it/s]
Training (3266 / 3550 Steps) (loss=0.13622): 100%|| 71/71 [00:53<00:00,  1.32it/s]
Validating: 100%|| 36/36 [00:04<00:00,  7.81it/s]|| 70/71 [00:47<00:00,  1.48it/s]
accuracy: 0.9664902998236331:04<00:00,  8.53it/s]
Training (3337 / 3550 Steps) (loss=0.02583): 100%|| 71/71 [00:52<00:00,  1.36it/s]
Validating: 100%|| 36/36 [00:04<00:00,  7.83it/s]|| 70/71 [00:48<00:00,  1.47it/s]
accuracy: 0.9506172839506173:04<00:00,  8.21it/s]
Training (3408 / 3550 Steps) (loss=0.07206): 100%|| 71/71 [00:53<00:00,  1.34it/s]
Validating: 100%|| 36/36 [00:04<00:00,  7.57it/s]|| 70/71 [00:48<00:00,  1.45it/s]
accuracy: 0.9700176366843033:04<00:00,  7.44it/s]
Training (3479 / 3550 Steps) (loss=0.02318): 100%|| 71/71 [00:52<00:00,  1.34it/s]
Validating: 100%|| 36/36 [00:04<00:00,  7.73it/s]|| 70/71 [00:48<00:00,  1.48it/s]
accuracy: 0.9594356261022927:04<00:00,  8.76it/s]
Training (3550 / 3550 Steps) (loss=0.07616):  99%|| 70/71 [00:53<00:00,  1.31it/s]
```

```
(pytorch) gongms@lthpc-X10DAi:~/deeplearning/classfication/vit/vision_transformer_pytorch-main$ python test.py
./data/Drowsiness_Driven_Dataset/train/
{'closed': 0, 'no_yawn': 1, 'open': 2, 'yawn': 3}
{'closed': 0, 'no_yawn': 1, 'open': 2, 'yawn': 3}
testing: 100%|| 777 [00:01<00:00,  5.43it/s]
accuracy: 0.9108910891089109
```

预测

> 拿训练好的模型进行推理，经过测试后**推荐使用 ResNet65**

输入的图片（网上随机选取）：



ResNet65

ResNet65 推理结果：第 12 张图片错误(应该是闭眼)，其余均正确

```
(pytorch) gongms@lthpc-X10DAi:~/deeplearning/classfication/demo/Drowsiness_Driven$ python predict.py
File: ./predict/1.jpg
softmax输出： tensor([[ 2.0481, -6.9757,  9.3531, -4.4577]], device='cuda:0')
预测结果(取最大概率的下标)： 2
预测结果： open
------------------------------------------------
File: ./predict/2.jpg
softmax输出： tensor([[ 6.6812, -5.0692,  4.8991, -6.4213]], device='cuda:0')
预测结果(取最大概率的下标)： 0
预测结果： closed
------------------------------------------------
File: ./predict/3.jpg
softmax输出： tensor([[ 4.5098, -5.4403, -0.2354,  1.2022]], device='cuda:0')
预测结果(取最大概率的下标)： 0
预测结果： closed
------------------------------------------------
File: ./predict/4.jpg
softmax输出： tensor([[-4.6923,  4.6188, -8.9444,  8.9530]], device='cuda:0')
预测结果(取最大概率的下标)： 3
预测结果： yawn
------------------------------------------------
File: ./predict/5.jpg
softmax输出： tensor([[-4.5146, -0.8348, -4.8882, 10.1284]], device='cuda:0')
预测结果(取最大概率的下标)： 3
预测结果： yawn
------------------------------------------------
File: ./predict/6.jpg
softmax输出： tensor([[-6.6924,  1.2075, -6.1557, 11.5300]], device='cuda:0')
```

```
预测结果(取最大概率的下标):  3
预测结果:  yawn
--------------------------------------------------
File: ./predict/7.jpg
softmax输出:  tensor([[-9.6726,  7.1853, -9.2824, 11.6336]], device='cuda:0')
预测结果(取最大概率的下标):  3
预测结果:  yawn
--------------------------------------------------
File: ./predict/8.jpg
softmax输出:  tensor([[-0.4393, -4.2403,  2.6494,  1.9546]], device='cuda:0')
预测结果(取最大概率的下标):  2
预测结果:  open
--------------------------------------------------
File: ./predict/9.jpg
softmax输出:  tensor([[-9.4713,  8.4201, -7.4771,  8.4061]], device='cuda:0')
预测结果(取最大概率的下标):  1
预测结果:  no_yawn
--------------------------------------------------
File: ./predict/10.jpg
softmax输出:  tensor([[ 3.5375,  0.0377, -2.3862, -1.1385]], device='cuda:0')
预测结果(取最大概率的下标):  0
预测结果:  closed
--------------------------------------------------
File: ./predict/11.jpg
softmax输出:  tensor([[-7.3199,  5.1182, -0.3611,  2.4797]], device='cuda:0')
预测结果(取最大概率的下标):  1
预测结果:  no_yawn
--------------------------------------------------
File: ./predict/12.jpg
softmax输出:  tensor([[-2.5971,  4.6659, -2.8244,  0.7054]], device='cuda:0')
预测结果(取最大概率的下标):  1
预测结果:  no_yawn
```

ResNet50_Pre

ResNet50_Pre 推理结果：第 3、9、11 错误

```
File: ./predict/1.jpg
softmax输出:  tensor([[ 1.9476, -7.7202,  3.3956, -5.0835]], device='cuda:0')
预测结果(取最大概率的下标):  2
预测结果:  open
--------------------------------------------------
File: ./predict/2.jpg
softmax输出:  tensor([[ 4.7925, -6.5107, -1.3402, -4.0957]], device='cuda:0')
预测结果(取最大概率的下标):  0
预测结果:  closed
--------------------------------------------------
File: ./predict/3.jpg
softmax输出:  tensor([[-2.5213,  0.6090, -7.3282, -3.7169]], device='cuda:0')
预测结果(取最大概率的下标):  1
预测结果:  no_yawn
--------------------------------------------------
File: ./predict/4.jpg
softmax输出:  tensor([[-7.5170, -3.4700, -4.4934,  2.3296]], device='cuda:0')
预测结果(取最大概率的下标):  3
```

```
预测结果：  yawn
------------------------------------------------
File: ./predict/5.jpg
softmax输出：  tensor([[-6.7431, -0.3106, -7.0701,  0.7344]], device='cuda:0')
预测结果(取最大概率的下标)：  3
预测结果：  yawn
------------------------------------------------
File: ./predict/6.jpg
softmax输出：  tensor([[-10.2935,  -3.1230,  -9.5969,  5.0351]], device='cuda:0')
预测结果(取最大概率的下标)：  3
预测结果：  yawn
------------------------------------------------
File: ./predict/7.jpg
softmax输出：  tensor([[-8.9757, -0.7012, -9.1853,  1.6414]], device='cuda:0')
预测结果(取最大概率的下标)：  3
预测结果：  yawn
------------------------------------------------
File: ./predict/8.jpg
softmax输出：  tensor([[-4.2363, -5.8626,  1.1278, -1.3190]], device='cuda:0')
预测结果(取最大概率的下标)：  2
预测结果：  open
------------------------------------------------
File: ./predict/9.jpg
softmax输出：  tensor([[-6.4169, -0.0626, -8.3639,  0.7276]], device='cuda:0')
预测结果(取最大概率的下标)：  3
预测结果：  yawn
------------------------------------------------
File: ./predict/10.jpg
softmax输出：  tensor([[-1.9250, -2.3541, -2.6557, -2.4747]], device='cuda:0')
预测结果(取最大概率的下标)：  0
预测结果：  closed
------------------------------------------------
File: ./predict/11.jpg
softmax输出：  tensor([[-7.7524, -1.3616, -5.3749,  1.1164]], device='cuda:0')
预测结果(取最大概率的下标)：  3
预测结果：  yawn
------------------------------------------------
File: ./predict/12.jpg
softmax输出：  tensor([[-0.9589, -3.0045, -3.5116, -1.6397]], device='cuda:0')
预测结果(取最大概率的下标)：  0
预测结果：  closed
```

VGG16_Pre

VGG16_Pre 推理结果：第 3、7、12 张图片错误

```
File: ./predict/1.jpg
softmax输出：  tensor([[-420.9501,  180.3911, 1125.6870, -215.3293]], device='cuda:0')
预测结果(取最大概率的下标)：  2
预测结果：  open
------------------------------------------------
File: ./predict/2.jpg
softmax输出：  tensor([[ 296.8548, -354.6957, -246.7148, -249.6912]], device='cuda:0')
预测结果(取最大概率的下标)：  0
预测结果：  closed
```

```
----------------------------------------------------
File: ./predict/3.jpg
softmax输出： tensor([[-23.3848,  22.9013, -23.4173,  28.4764]], device='cuda:0')
预测结果(取最大概率的下标)： 3
预测结果： yawn
----------------------------------------------------
File: ./predict/4.jpg
softmax输出： tensor([[ -84.9323,  88.2497, -145.4513,  126.0761]], device='cuda:0')
预测结果(取最大概率的下标)： 3
预测结果： yawn
----------------------------------------------------
File: ./predict/5.jpg
softmax输出： tensor([[-133.5100,  142.5634, -245.4011,  201.1856]], device='cuda:0')
预测结果(取最大概率的下标)： 3
预测结果： yawn
----------------------------------------------------
File: ./predict/6.jpg
softmax输出： tensor([[ -628.7358,  717.5584, -1081.9597,  940.1487]], device='cuda:0')
预测结果(取最大概率的下标)： 3
预测结果： yawn
----------------------------------------------------
File: ./predict/7.jpg
softmax输出： tensor([[-66.5986, 109.3648, -92.4563,  89.2949]], device='cuda:0')
预测结果(取最大概率的下标)： 1
预测结果： no_yawn
----------------------------------------------------
File: ./predict/8.jpg
softmax输出： tensor([[-223.7428,  142.1971,  153.2383,  94.2483]], device='cuda:0')
预测结果(取最大概率的下标)： 2
预测结果： open
----------------------------------------------------
File: ./predict/9.jpg
softmax输出： tensor([[-37.8921,  53.2545, -77.9772,  52.5458]], device='cuda:0')
预测结果(取最大概率的下标)： 1
预测结果： no_yawn
----------------------------------------------------
File: ./predict/10.jpg
softmax输出： tensor([[ 19.2300, -23.8901, -24.9098, -19.0613]], device='cuda:0')
预测结果(取最大概率的下标)： 0
预测结果： closed
----------------------------------------------------
File: ./predict/11.jpg
softmax输出： tensor([[-44.7589,  72.3245, -62.9549,  57.2859]], device='cuda:0')
预测结果(取最大概率的下标)： 1
预测结果： no_yawn
----------------------------------------------------
File: ./predict/12.jpg
softmax输出： tensor([[-43.3609,  6.5199, 120.3114, -19.4932]], device='cuda:0')
预测结果(取最大概率的下标)： 2
预测结果： open
```

即使 VGG16_Pre 和 ResNet50_Pre 测试集的准确率高于 ResNet65，但是泛化能力并没有 ResNet65 强

GoogLeNet_Pre

GoogLeNet_Pre 推理结果：第 3、10、11 错误

```
File: ./pic_predict/1.jpg
softmax输出： tensor([[-1.4140, -6.0615,  3.1387, -3.2899]], device='cuda:0')
预测结果(取最大概率的下标)： 2
预测结果： open
-------------------------------------------------
File: ./pic_predict/2.jpg
softmax输出： tensor([[ 3.5432, -3.5095, -3.3725, -4.9142]], device='cuda:0')
预测结果(取最大概率的下标)： 0
预测结果： closed
-------------------------------------------------
File: ./pic_predict/3.jpg
softmax输出： tensor([[-7.2783,  1.1537, -4.6258, -0.7067]], device='cuda:0')
预测结果(取最大概率的下标)： 1
预测结果： no_yawn
-------------------------------------------------
File: ./pic_predict/4.jpg
softmax输出： tensor([[-6.0416, -2.0454, -4.7209, -1.0360]], device='cuda:0')
预测结果(取最大概率的下标)： 3
预测结果： yawn
-------------------------------------------------
File: ./pic_predict/5.jpg
softmax输出： tensor([[-7.8202, -0.3775, -6.5736,  1.0719]], device='cuda:0')
预测结果(取最大概率的下标)： 3
预测结果： yawn
-------------------------------------------------
File: ./pic_predict/6.jpg
softmax输出： tensor([[-7.5805, -1.2463, -7.5935,  0.9389]], device='cuda:0')
预测结果(取最大概率的下标)： 3
预测结果： yawn
-------------------------------------------------
File: ./pic_predict/7.jpg
softmax输出： tensor([[-5.6567, -0.0981, -6.6784,  0.5969]], device='cuda:0')
预测结果(取最大概率的下标)： 3
预测结果： yawn
-------------------------------------------------
File: ./pic_predict/8.jpg
softmax输出： tensor([[-2.2141, -3.7654, -0.3469, -2.4287]], device='cuda:0')
预测结果(取最大概率的下标)： 2
预测结果： open
-------------------------------------------------
File: ./pic_predict/9.jpg
softmax输出： tensor([[-6.4012,  0.2477, -7.7052, -0.9869]], device='cuda:0')
预测结果(取最大概率的下标)： 1
预测结果： no_yawn
-------------------------------------------------
File: ./pic_predict/10.jpg
softmax输出： tensor([[-5.1743,  1.0588, -3.9650, -2.6168]], device='cuda:0')
预测结果(取最大概率的下标)： 1
预测结果： no_yawn
-------------------------------------------------
File: ./pic_predict/11.jpg
softmax输出： tensor([[-5.0613, -1.3764, -6.2466,  0.3814]], device='cuda:0')
预测结果(取最大概率的下标)： 3
```

```
预测结果：  yawn
-----------------------------------------------
File: ./pic_predict/12.jpg
softmax输出：  tensor([[ 0.5895, -1.6516, -5.9351, -2.8247]], device='cuda:0')
预测结果(取最大概率的下标)：  0
预测结果：  closed
```

VIT16_Pre

VIT16_Pre 推理结果：第 3、4、7、9、12 错误

```
File: ./predict/1.jpg
softmax输出：  tensor([[-0.1342, -4.2932,  9.9871, -7.4602]], device='cuda:1')
预测结果(取最大概率的下标)：  2
预测结果：  open
-----------------------------------------------
File: ./predict/2.jpg
softmax输出：  tensor([[13.3638, -7.0934,  0.7089, -5.7441]], device='cuda:1')
预测结果(取最大概率的下标)：  0
预测结果：  closed
-----------------------------------------------
File: ./predict/3.jpg
softmax输出：  tensor([[-1.7041,  9.4396, -2.6631, -5.8740]], device='cuda:1')
预测结果(取最大概率的下标)：  1
预测结果：  no_yawn
-----------------------------------------------
File: ./predict/4.jpg
softmax输出：  tensor([[-5.3395,  7.0319, -4.4615,  2.1551]], device='cuda:1')
预测结果(取最大概率的下标)：  1
预测结果：  no_yawn
-----------------------------------------------
File: ./predict/5.jpg
softmax输出：  tensor([[-8.3445,  7.6477, -7.7515,  8.1106]], device='cuda:1')
预测结果(取最大概率的下标)：  3
预测结果：  yawn
-----------------------------------------------
File: ./predict/6.jpg
softmax输出：  tensor([[-6.8393, -1.6793, -5.3142, 13.4725]], device='cuda:1')
预测结果(取最大概率的下标)：  3
预测结果：  yawn
-----------------------------------------------
File: ./predict/7.jpg
softmax输出：  tensor([[-7.2221,  5.9493, -5.3427,  4.6794]], device='cuda:1')
预测结果(取最大概率的下标)：  1
预测结果：  no_yawn
-----------------------------------------------
File: ./predict/8.jpg
softmax输出：  tensor([[-4.4516,  2.0728,  2.1185,  0.4809]], device='cuda:1')
预测结果(取最大概率的下标)：  2
预测结果：  open
-----------------------------------------------
File: ./predict/9.jpg
softmax输出：  tensor([[-5.8411,  4.1920, -5.3130,  6.7928]], device='cuda:1')
预测结果(取最大概率的下标)：  3
```

```
预测结果：  yawn
-------------------------------------------------
File: ./predict/10.jpg
softmax输出： tensor([[ 2.7023,  0.3983, -2.2087, -1.3971]], device='cuda:1')
预测结果(取最大概率的下标):  0
预测结果：  closed
-------------------------------------------------
File: ./predict/11.jpg
softmax输出： tensor([[-3.0988, -0.1286, -4.0446,  7.0364]], device='cuda:1')
预测结果(取最大概率的下标):  3
预测结果：  yawn
-------------------------------------------------
File: ./predict/12.jpg
softmax输出： tensor([[-2.8541,  4.6394, -5.3871,  3.4944]], device='cuda:1')
预测结果(取最大概率的下标):  1
预测结果：  no_yawn
```