人工智能基础项目二报告

徐昕翃 2018010740 自94

一，项目简介：

针对本次项目，笔者选择**表情识别**，需要完成的任务如下：

1，设计深度学习算法，在给定数据集上实现表情识别，设计合适的模型评价方法评价模型的性能。

2，数据集的样本不均衡会如何影响模型的性能，提供可能的解决方案。

3，提供一个输入图像或视频的接口，识别其中人脸，利用 1 中的模型判断表情并完成标注（实现过程可以利用 $opencv$ 等资源，接口的实现方式需要注明）。

注：模型链接 click here，在模型解压后请将各个网络权重的 pkl 文件直接移植 $code\_etc$ 目录下运行，在 $readme.txt$ 文件中也有说明。

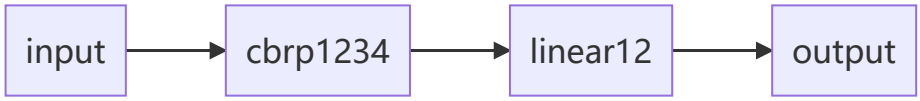
二，分类任务（必做）：

1，深度方法设计：

（1）网络结构：

使用 $pytorch$ 定义网络并进行训练。笔者首先定义了两种网络，分别为类 VGG 和 类 $resnet$，然后进行训练。$VGG10$ 模型和 $resnet17$ 模型结构如下所示，其中数字表示卷积和全连接层的层数和。其中 $cbrp$ 表示卷积，批归一化，$ReLU$ 激活函数，最大池化。这里的每一个 $cbrp$ 模块中 cbr 操作执行 2 次；res_block 表示自行设计的残差块，$resnet17$ 中每个残差块仅循环一次；$linear$ 表示线性层，$Dropout$ 方法，$ReLU$ 激活函数。最后一个 $linear$ 只有一个线性层。$resnet$ 中所有的激活函数均为参数为 0.1 的 $LeakyReLU$，详细代码请参见附件。

$VGG\ net$:



$resnet$:

（2）训练过程：

损失函数使用交叉熵，优化器采用 $NAdam$。这里没有采用验证集，因为本身训练集的数据就不是很多，这样对于模型最终的泛化能力可能有所损害。由于不清楚样本的具体情况，所以首先不使用任何 $trick$，数据处理仅仅做了 $centerCrop$ 和归一化，然后直接训练。

（3）实验结果：

VGG 网络在很短的时间内（10 个 $epoch$ 左右），训练集的正确率就能够到达 80% 以上，经过 30 个 $epoch$ 的训练，就能够达到 98% 以上的正确率。但奇怪的是，模型在测试集上的正确率只有 61.72%。这一结果引起了笔者很大的兴趣，因为对于一般的分类任务，如果在训练集上达到了 90% 以上的正确率，测试集上的正确率通常不会低至这般。

$resnet$ 网络训练的时间比 VGG 长不少，训练了 100 个 $epoch$。首先需要以较小的学习率进行 $warm\ up$，然后在提高学习率至训练集上的正确率饱和，再降低学习率迫近收敛。但是最终效果和 VGG 的也较为类似，在测试集上的效果很不好。

网络的正确率如下所示：

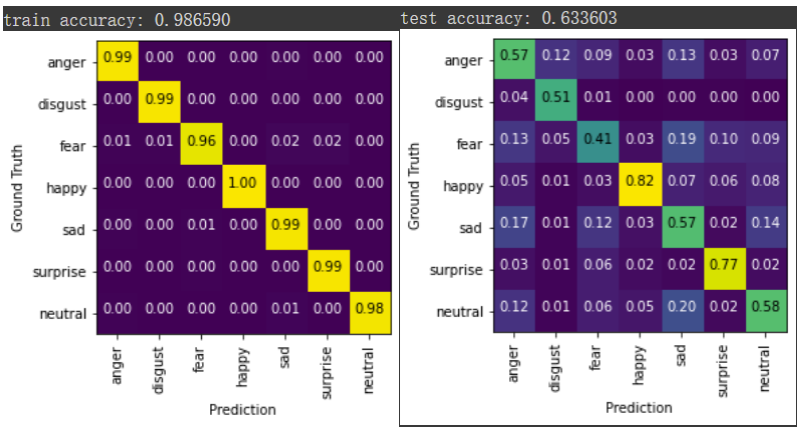
Network	best test accuracy
VGG	0.633603
resnet	0.535107

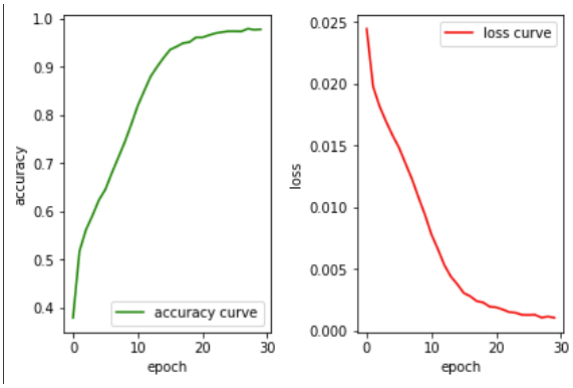
实验结果如下图所示。衡量多分类模型的一个很重要的指标就是以数量或者概率表示的混淆矩阵，通过混淆矩阵可以很轻松地观察到各种样本的分类情况。图一为测试正确率和测试集的混淆矩阵具体数字，图二为概率混淆矩阵，图三为测试集上的各种评价指标，其中的二分类指标（在 $kappa$ 前）是通过比较一类和其他所有类完成的；下面简要介绍采用的多分类指标。

$kappa$ 系数是用于统计学中评估一致性的一种方法，其取值范围为 $[-1, 1]$，越趋近 1 则说明模型越好，趋近 0 或者更低表明结果的预测更可能是一种偶然性。海明距离是衡量预测标签和真实标签之间的距离的，经过归一化后取值范围在 $[0, 1]$ 之间，结果越小，说明预测值和真实值越趋近，模型效果越好。

从实验结果可以看出，分类的结果并不好，测试正确率很低，重点在于 $anger, fear, sad, neutral$ 的分类效果很差。可见其中的召回率和准确率都很低，各类平均 AUC 也很小，多分类的指标也仅仅是居中。说明模型很不稳定，并且过拟合。

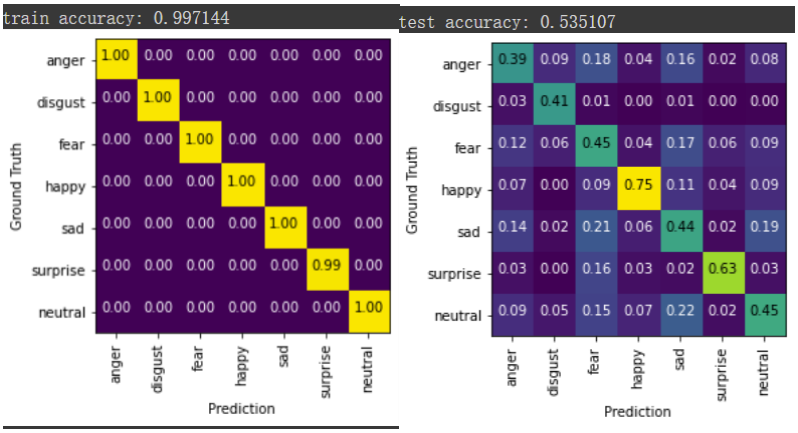
（1）VGG 网络指标：

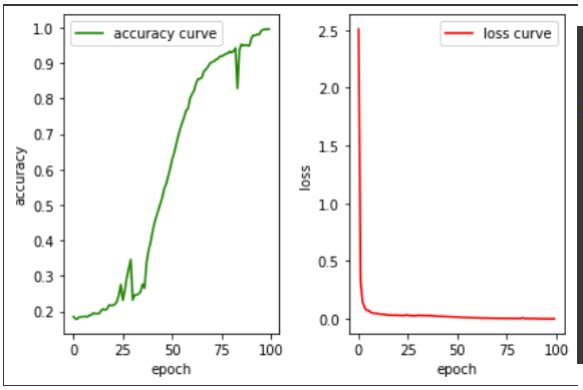





```
********************
evaluating indicators are as below:
BER: 0.228660
MCC: 0.556334
sensitivity: 0.605379
specificity: 0.937301
recall: 0.605379
precision: 0.635991
F1: 0.616478
auROC: 0.771340
auPRC: 0.450592
kappa: 0.556750
hamming: 0.366397
********************
```

（2）$resnet$ 网络指标：






```
**********
evaluating indicators are as below:
BER: 0.288588
MCC: 0.445463
sensitivity: 0.502607
specificity: 0.920217
recall: 0.502607
precision: 0.553516
F1: 0.521457
auROC: 0.711412
auPRC: 0.360367
kappa: 0.437014
hamming: 0.464893
**********
```

2，样本不均衡处理：

通过上述的混淆矩阵可以看出，样本的不均衡有体现在两处：一是样本的数量不均衡，一是样本的分辨难易程度不均衡。这两种不均衡在本次作业中体现得很明显：表情 $disgust$ 的数量相较于其他很小；与此同时通过混淆矩阵可以发现，分类器很容易区分开 $happy, surprise$，但是对于 $anger, fear, sad$ 等区分效果却很不好。所以这里需要采用很多方法修正目前遇到的问题。

（1）损失函数：

这里为了处理样本数量和分辨难易程度不均衡的问题，笔者设计引入了一个新的 $loss$，命名为 $adaptive\ loss$.其原理如下所示：

考虑 N 个样本，$\vec{y_i}, i \in [0, N)$ 表示样本的标签，i 表示对于 N 的索引。分类的种类数为 K，本次项目中为 7，k 作为标签中种类的索引：$k \in [0, K)$。$t = t(i),\ t \in [0, K)$ 作为每个标签 i 对应的真值位置，是 i 的函数，满足 $y_{it} = 1$。（$\vec{y_i}$ 表示样本 i 的标签向量，y_{ik} 表示样本 i 标签中的第 k 个元素）。

如此对于 $CrossEntropy$ 损失，有：

$$CE = -\frac{1}{N}\sum_{i=0}^{N-1} y_{it}\log p_{it}$$

这表明交叉熵损失仅关注正样本的概率，并希望其趋于最大，即 $CE \to 0 \leftrightarrow p_{it} \to 1,\ \forall i \in [0, N)$.

现在首先引入 $label\ smoothing$，将 $ground\ truth$ 转换为 $one\ hot$ 的形式，然后做 $smoothing$ 处理。如此是为了防止网络对于预测过于自信，在预测时保留一定的空间，希望以此减小难分辨样本分类错误的概率，提升模型的泛化性能。算法过程如下：对于样本 i 的标签 $\vec{y_i}$，有 $y_{it} = 1 - \epsilon,\ y_{ik} = \frac{\epsilon}{K}(1 - \frac{1}{K}), \forall k \neq t, k \in [0, K)$。这样得到的方程为：

$$CE\_smoothing = -\frac{1}{N}\sum_{i=0}^{N-1}\sum_{k=0}^{K-1} y_{ik}\log p_{ik}, \forall k \in [0, K)$$

然后引入 $focal\ loss$ 的思想：对于预测的结果 $\overrightarrow{CE\_smoothing} = [-\sum_{k=0}^{K-1} y_{ik}\log p_{ik}]$，这里写成了向量形式，即每一个样本对应一个单独的 ce_i 损失，可以对其做更强的平滑处理。考虑修改后的交叉熵损失：如果 ce_i 损失很大，则 $pt = \exp(-\mathrm{ce}_i) \in (0, 1)$ 会很小，故 $1 - pt$ 会变大，即与 ce_i 呈单调同向变化关系。对于容易学习的样本（例如特征明显或数量很多），在 ce_i 较小的情况下，可以通过乘以系数 $(1 - pt)^\gamma, \gamma > 0$，这样可以进一步抑制其权重的更新，防止其完全主导梯度，削弱其在梯度修正中起到的作用。而对于分类困难的样本（容易分错或数量很少），则 ce_i 较大，那么 $1 - pt$ 也很大，这样可以更加有效的更新困难样本和小样本的权重。实际上易分样本的数量一般比较多，故抑制损失和权重的情况下整体更新效率也未必低。但是笔者并不是想抑制易分样本的梯度变化，而是希望以此获得难易分类或者数量不均衡样本梯度更新时的平衡。

最后引入 $\overrightarrow{weight} = \overrightarrow{\alpha}$，其有 K 个值，便于控制每类样本的权重损失以及更新梯度的大小。α 值的选择依靠 $\vec{y}_i$ 的真值位置。即 $\alpha_{\vec{y}_i} = \alpha_t$。如果说 $focal\ loss$ 是自动调整梯度变化，那么这就是手动调整梯度了。这里通过对于难易程度的判断，选择将 $anger, disgust, fear, sad, neutral$ 的权值设置较大，将 $happy, surprise$ 的权值设置较小。

即最终样本 i 的 $final\_loss$ 为：

$$\mathrm{cef}_i = \alpha_t * (1 - \exp(\mathrm{ce}_i))^\gamma * \mathrm{ce}_i$$

$$\mathrm{ce}_i = -\sum_{k=0}^{K-1} y_{ik}\log p_{ik}$$

$$y_{ik} = \begin{cases} 1 - \epsilon(1 - \frac{1}{K}), & k = t \\ \frac{\epsilon}{K}, & k \neq t \end{cases}$$

如此实现的 $loss$ 代码如下所示，写成矩阵运算形式便于进行加速：

```
class adaptive_loss(nn.Module):
    """
    Perform label smoothing + focal loss + loss gradient weight
    Note that parameter 'alpha' has type: torch.FloatTensor and should be sent to
device before training if it is cuda
    """
    def __init__(self, alpha, gamma, smoothing_rate, num_class):
        super(adaptive_loss, self).__init__()
        self.alpha = alpha
        self.gamma = gamma
        self.smoothing_rate = smoothing_rate
        self.num_class = num_class

    def forward(self, pre, gt):
        """
        pre shape: batch_size, class number
        gt shape: batch_size,
        """
        log_softmax = F.log_softmax(pre, dim=1)
        expanded_gt = F.one_hot(gt, self.num_class)
        smoothing_gt = torch.clamp(
            expanded_gt.type(torch.float), self.smoothing_rate / self.num_class,
1 - self.smoothing_rate
        )
        weight = self.alpha.gather(dim=0, index=gt)  # weight[i] = alpha[gt[i]]
        smoothing_ce_loss = - torch.sum(log_softmax * smoothing_gt, dim=1)
```

```
        pt = torch.exp(- smoothing_ce_loss)
        adap_loss = weight * (1 - pt) ** self.gamma * smoothing_ce_loss
        return adap_loss.mean()
```

通过实验表明，这种 $loss$ 训练比交叉熵效果略好，而且能够在网络训练到后期通过权重的设置进行微调。

（2）数据增强：

“数据是上限，而算法本身则是不断趋近这个上限。”

本次数据集的数据并不是非常多，并且特定种类图片的数量还非常少，所以需要采用数据增强来缓解训练集本身对于算法的限制。这里采用的数据增强方式有很多，需要保证能够产生尽可能多的变化，选择使用 $torchvision.transforms$ 中尽可能多的变换，以此增加模型的鲁棒性，告诉网络即使图片发生了一定的变换，但是它的本质没有变。如此变相扩充数据集，能够提升最终的测试准确率。

对于训练集，笔者采用了随机拉伸裁剪，仿射变换，随机水平翻转，随机旋转一定角度等等尽可能多的变化，然后使用 $tenCrop$ 以及随机擦除等方法，以此保证模型见过足够多的变换。最终通过训练集数据计算出的均值和方差进行归一化。

测试集采用 $tenCrop$ 的方法进行测试，利用 $transforms$ 中的 $tenCrop$ 函数完成将测试图片按照一定规律切割成 10 份，然后预测时对概率进行加和选择最大值。$tenCrop$ 之后仍然通过均值和方差进行归一化。

（3）网络结构：

再次采用自己定义的 VGG 和 $resnet$ 进行实验，这些网络都是为了适应数据集而修改过的，和标准的网络不完全相同；同时这里采用了更深的 $resnet25$ 进行实验。详细网络的代码请参见附件。

（4）实验过程和结果：

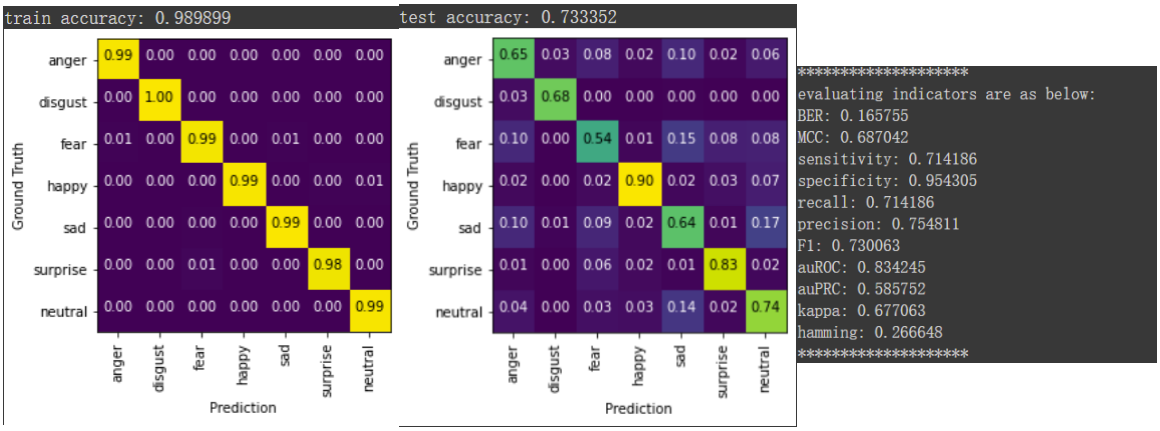
本次任务采用了大量的数据增强，训练使用 SGD 优化器，带动量和 $nesterov$ 方法，损失函数为 $adaptive\ loss$。设置初始的 $\gamma = 2, label\_smoothing = 0.05$，$\alpha$ 如下所示，在训练快要结束时，根据得到的测试混淆矩阵修改 α 的比例进行微调。

```
alpha = torch.FloatTensor([.6, 1., .8, .4, .8, .4, .6]).to(device)
```

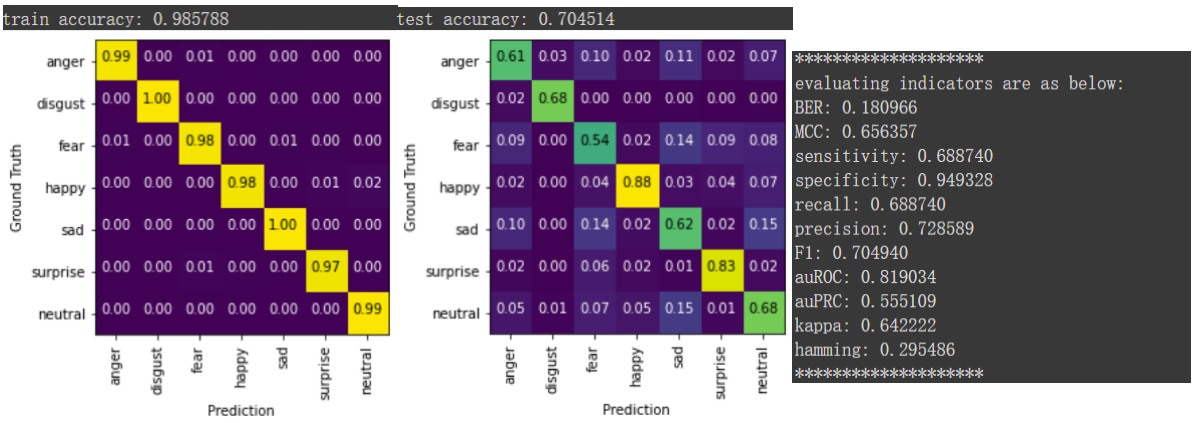
由于训练的次数大多在 300 个 $epoch$ 以上，手动改变参数（主要是学习率）的次数也很多，所以这里不提供损失和正确率的曲线。以下为各种不同网络的结果，网络之后的数字代表卷积和全连接的层数之和。实验中发现不但浅网络的训练速度比深网络快，而且其测试的效果也比深网络好。这里同时还展示了按照一定权重进行联合测试得到的结果，这里采用了三个网络作为联合测试的输入，第二，第三个网络对应下方的 $resnet25$ 和 $VGG10$，第一的网络对应另一个训练的 $VGG10$，其测试正确率为 72.5%。三个网络按照顺序命名为 a_net, b_net, c_net，通过 $grid\ search$ 得到最优权重比例为 0.1，0.3，0.6，得到的正确率为 74.0%，这也是本次实验中最高的正确率。 其他指标来看，相较于之前的实验也好很多，$kappa$ 的值相对较高，而海明距离的值相对较低，表明模型的精度提高了很多。

Net structure	Best test accuracy
$VGG10$	0.733352
$resnet25$	0.704514
$joint\ test$	0.740457

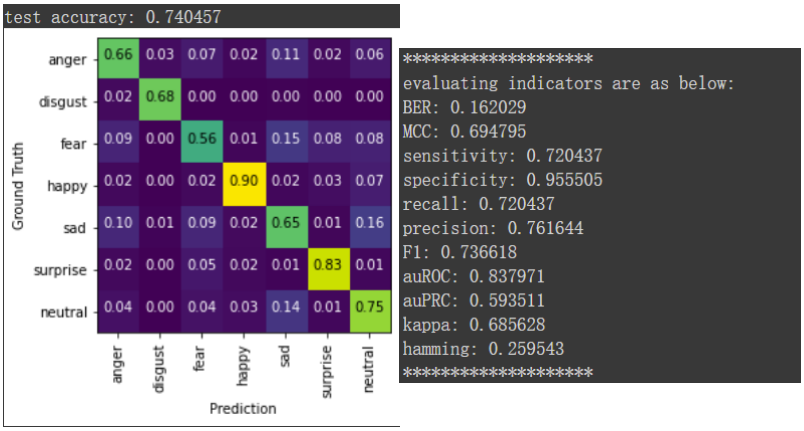
a，$VGG10$ 指标：



b，$resnet25$ 指标：



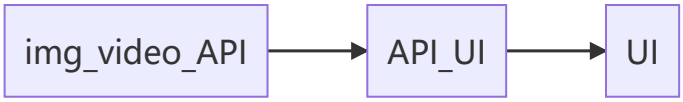
c，$joint\ test$ 指标：



三，接口设计（选做）：

（1）结构原理：

这里分成三个部分设计 GUI 接口，结构如下图所示：



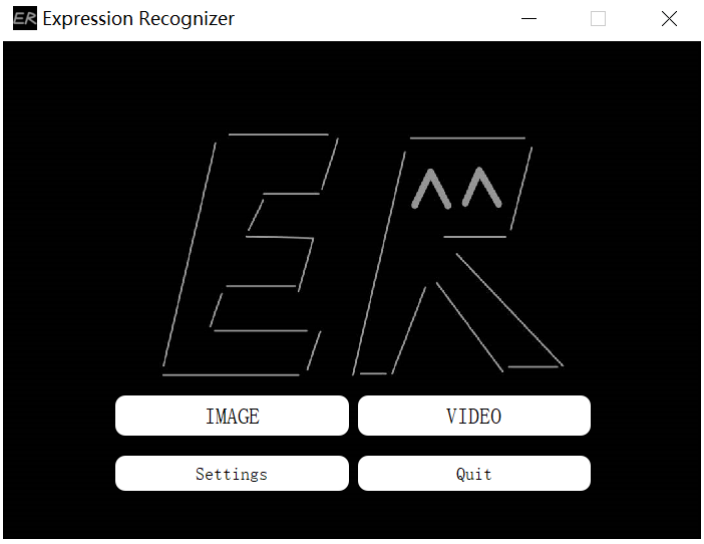
a，在 img_video_API 类中完成了 $interface$ 类，用于对图像和视频流中的帧进行人脸识别和表情识别等操作。这里的主要思路是通过 $cv2$ 中的 $CascadeClassifier$ 类完成对图像或视频流中帧的人脸识别。将识别出的人脸图取出，放缩到 $48*48$，转为灰度图，然后通过训练好的模型进行判别，输出表情。最终通过 $cv2$ 将人脸框和识别出的表情绘制出来，并放缩到指定的图片大小。这里的分类器使用单个测试结果最好的 $VGG10$ 网络作为分类的内核（即上文的 c_net）。

b，在 API_UI_class 中完成上层接口和底层的 UI 类的交互，主要包含各种 $callback$ 函数和 $thread$ 等操作，可以完成导入图片，调用 $interface$ 完成图像或者视频流的处理，将处理完成的结果进行显示等操作。

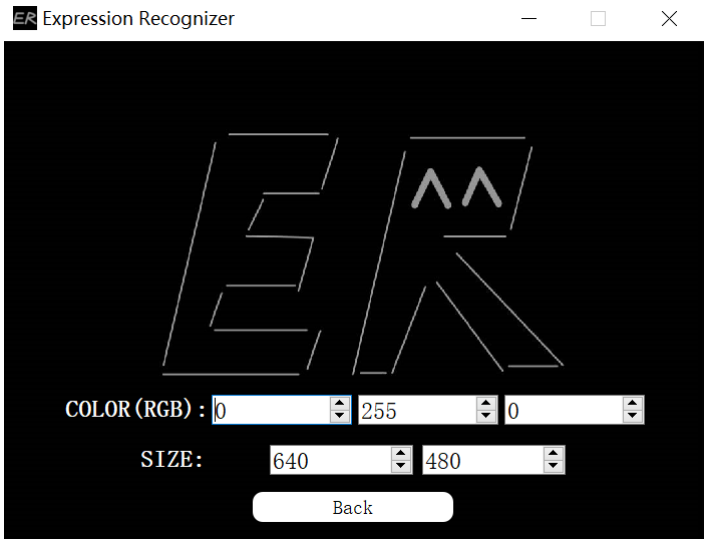
c，UI 包含了底层的界面，通过 $QtDesigner$ 完成设计。

（2）操作流程：

运行 $API\_UI\_class.py$ 文件（最下方有运行标志）。然后得到如下界面。其中有 4 个选项：$IMAGE$ 表示用于检测图片中的人脸并识别表情；$VIDEO$ 表示对视频流中的帧完成人脸检测及表情识别；$Settings$ 能够设置检测图片或视频标注框和字体的颜色以及显示图片或视频的播放尺寸；$Quit$ 则是退出。

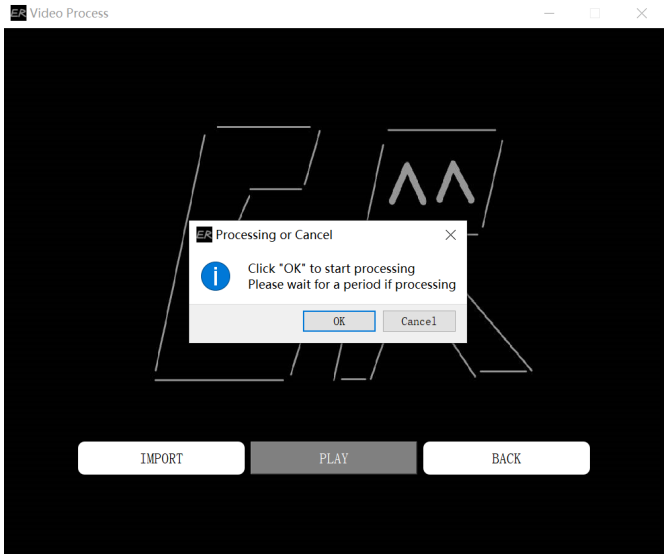


a，点击 $Settings$ 可以进入如下的界面。这里颜色选项排序依次为 RGB，范围是 0-255 的整数，默认是绿色。尺寸选项的宽度范围是 120-800 的整数值，高度范围是 80-600 的整数值。其中默认值为 640，480。颜色可以自行调整，但是不建议修改尺寸，因为尺寸是按照比例设定好的。



b，点击 $IMAGE$ 会进入图片导入界面，点击 $IMPORT$ 可以打开文件夹选择图片路径，这里图片只接受 jpg 格式的图片，而且不要过窄。导入图片后算法会很快完成测试，所以这里没有使用多线程。如果检测出了人脸，则会直接显示最终的人脸；否则会弹出提示框告知没有检测出人脸，并且将显示原图。可以连续导入图片，如果选择 $BACK$，则再次进入之前的图像会消失。

c，点击 $VIDEO$ 则会进入视频导入界面，可以通过点击 $IMPORT$ 导入视频，视频只接收 $mp4$ 文件。导入视频后会提示是否选择进行处理：如果选择 OK，则程序会以另一个线程处理视频，这需要花费一些时间。处理之后的视频会以 'output_video.mp4' 格式生成在当前目录下。处理完成之后，页面会自动弹出（如果在侧面则任务栏上的标识会闪烁）。在此过程中三个按键都是不能使用的，处理完成之后 $PLAY$ 按键就能够按了。如果点击 $BACK$ 退出，那么之前处理过的视频会丢失。



(3) 注意事项：

a，图像仅仅支持 jpg 文件（不包含 $jepg$ 后缀），视频仅支持 $mp4$ 文件。输入图像和视频的尺寸最好靠近 $(640, 480)$，请不要使用过于极端的尺寸。

b，请控制输入视频时间的长度，否则会处理很慢。

c，虽然可以修改显示的颜色和图像或视频大小（即 $color, size$），但是不建议修改 $size$，因为原先的尺寸是已经定好的，修改了反而不好。

d，$cv2$ 的人脸识别机制似乎并不是特别准确，有的时候也会出问题。

e，上交的 $test\_image\_video$ 文件夹中提供部分用于测试的图片和视频。

四，总结思考

1，在拿到作业之前本来以为分类任务会较为容易，但是实际却让人大跌眼镜。最开始一个很简单的网络可以轻易在训练集上到达 98% 以上的正确率，但是在测试集上的正确率却未必能够到及格线。后来笔者对测试集进行了可视化，发现了 2 个问题：一是有部分测试数据完全不正确，是脏数据，例如下图：

二是一部分数据标注确实人眼都无法判别，就例如 $fear$ 和 sad 等。这种情况下苛求机器判别正确似乎也不太合理。

2，针对本次项目，笔者进行了广泛的调研。虽然报告写得较为简洁，但是尝试了很多很多内容：例如不同的网络结构，不同的优化器，不同的损失函数等等。但是笔者认为最有趣的部分还是在于思考如何解决当前遇到的数据不平衡的问题。网络结构都是沿用前人的方法，优化器等也都可以直接通过接口进行调用，但损失函数是笔者通过调研设计的，尽管可能没有完美地获得设想的效果，但是确实在部分功能上是有作用的。通过这样的损失函数能够令 $happy, surprising$ 不会过分地占主导地位。这个 $loss$ 可以进行在训练快要完成时进行微调，如果有一些分类指标过低，那么可以重新修正比例进行调节，但是却不能保证原先分类高的数目不下降。所以笔者在想，如果能够固定一部分对正确分类某些种类起很大作用的权重，然后修改的权重能够在不影响之前已经分好的种类的情况下，提升难分样本的正确率，那这样就是很好的，但是似乎现在还完成不了。做出创新是很重要的。

3，神经网络不是越深越好。所谓"如无必要，勿增实体"，对于本次项目中面对的问题，使用简单的神经网络效果往往更好，使用复杂的网络反而很容易过拟合。这里采用了 VGG 和 $resnet$ 的结构，但是使用 $resnet$，哪怕每一个残差结构只有一个循环，训练时都很困难，会预热很长时间，而且最终得到的测试结果还打不过 VGG。进行大量实验也确实花费很多时间，所以这里笔者无暇做很多对比试验了，只能大致介绍一下整体的情况。本次实验中超参数也很多，除了基本的网络，优化器等的超参数，损失函数还有很多超参数，笔者实在没有很多时间控制变量进行训练。对于训练的学习率，是非常需要考量的，如果学习率过大，很容易出现 $loss$ 变为 $Not\ a\ number$ 的情况，这样网络直接就崩坏了，只能从头开始；如果学习率过小，容易无法跳出局部最小值，导致网络过早收敛。所以笔者一般先用比较小的学习率进行 $warm\ up$，然后等到训练正确率快速增加时，再使用较大的学习率。然后保持这样的学习率不变，等到其维持一个较高的正确率（$e.g.$ 85% 以上），再降低学习率，这样往往能够取得一个较好的效果。

4，在研究一个问题之前需要做广泛的预调研，不能直接盲目去行动，否则效率很低，会浪费很多时间。需要先快速熟悉项目相关的内容，做出 $baseline$，然后再考虑创新，这样会容易取得更好的效果。这里其实笔者还有一个想法：针对数据不平衡的模型，可以首先训练一个二分类器，专门区分是否是最少的那个类别，正样本为数量最少的那个类，负样本为从其他类中任选；然后再训练一个全分类器，进行加权得到最终的结果。但是最终训练的结果表明其实数据量少的影响并不是很大，而且这样训练很容易过拟合，所以笔者没有采取这样的处理方式。

5，提升模型的泛化能力很重要。数据增强在图像处理方面的应用确实太重要了，不但能够扩充训练集，提高模型的精度，而且能够减少过拟合，提高泛化能力。这也再次说明了数据集的重要性。联合测试也能够提升一定的模型测试正确率，这就是一种并行的集成学习，相当于引入了一定的随机性，希望最优分类器分不好的内容（即预测的概率权重较为接近时），其他的分类器能够补充。但是这种方法也是需要很多尝试，有点猜测的意味。如果想要深究的话，通过多模型的 CNN 进行平行判断，笔者认为正确率肯定还能继续提升。

6，项目中有很多细节，例如： $pyqtSignal$ 构建信号时首先要确定信号的类型，否则信号发送时系统会崩溃。之前随手写了，现在没写就不行了，被这个“小问题”占用了很多时间。对于线程、信号和槽的处理，还是需要小心；不要在交叉熵函数前使用 $ReLU$，这也是很不理智的行为；还有就是 $cv2$ 图像的索引是 (y, x)，这也是很容易犯错的问题，写错了的情况下会导致检测结果偏差很大。