



EmotionStyle&SmokingDetct

- 基于 Gram 矩阵解析优化的风格迁移
- 基于Yolov8的表情检测以及抽烟检测

- ◆ 姓名：李贤达（50%）、史健申（50%）
- ◆ 学号：10235101495、10235101561
- ◆ 组号：34组
- ◆ 课程：数字图像处理
- ◆ 任课教师：曹桂涛
- ◆ 日期：2025年6月14号

目录

1 引言以及应用背景	1
2 基础知识	3
2.1 数字图像处理基本原理	3
2.2 风格迁移原理解析	5
2.2 YOLOv8底层原理	
3 NST模型优化	8
3.1 L-BFGS优化	8
3.2 Adam进行优化	8
4 YOLOv8模型训练	12
4.1 数据集	12
4.2 训练方法	13
4.2.1 数据集预处理	13
4.2.2 增强策略	13
4.3 训练过程	14
4.3.1 表情检测	13
4.3.2 视频流的实时表情识别	13
4.3.3 抽烟检测	13
5 图像处理系统UI页面搭建	20
6 完成效果	20
7 总结	20

摘要

学习了数字图像处理这门课程后，对数字图像处理的基本方法以及其背后的知识原理有了较为深刻的认识。随着人工智能的迅速发展，将传统的数字图像处理技术与深度学习的方法相结合成为了一个全新的研究方向。

从预训练的卷积神经网络中不同层的激活中可以解耦出内容与风格，对其进行更加深度的训练就可以得到支持任意风格迁移的模型；根据神经网络可以自动学习图像特征的原理，实现了图像分类、目标检测以及实例分割的统一框架YOLO。对于这两个经典模型我们对他进行训练和处理将其应用到现实生活中的例子上去。本项目中，我们将数字图像基础功能与风格迁移和目标检测整合成一个系统。

关键词：卷积神经网络，YOLO框架，风格迁移，表情检测，吸烟检测

1 引言

随着人工智能技术的快速发展，计算机视觉在人机交互、心理健康评估等多个领域中发挥着重要的作用。其中，表情检测作为情感计算的关键环节，能够使机器识别并理解人类的情绪状态，是实现“情绪智能”的基础。

表情是人类表达情绪的重要方式，具有自然直观的特点。通过对图像或视频中的人脸表情进行识别，可以推断出个体在某一时刻的情绪状态和内心想法。然而，表情识别仍存在着精度不够、识别不准的问题，例如表情的微小差异、头部姿态偏转、遮挡、个体差异等。近年来，深度学习特别是卷积神经网络和迁移学习方法的应用，极大提升了模型对复杂表情特征的提取与识别能力。同时，随着大规模表情数据集的开源，表情识别算法逐渐贴近生活。

在此背景下，本项目旨在构建一个基于深度学习原理的**表情检测系统**，支持图像或视频输入，能够准确、高效地识别包括高兴、悲伤、愤怒、惊讶这四种常见的情绪。项目将结合YOLOv8进行人脸检测与定位，并集成专用的表情识别模型，最终在电脑网页中实时表情分析与可视化展示。

我国抽烟问题日渐加剧，世界卫生组织指出，烟草使用每年造成**超过800万人死亡**，其中约130万人是被动吸烟者。吸烟和被动吸烟正在成为死亡的首要风险，所以在公共场合我们要着力于杜绝吸烟行为，实现空间全覆盖，但这也导致需要极大的人力成本。我们的项目将通过YOLO模型结合自定义增强策略实现对香烟这种小目标的精准检测，从而降低人工巡检成本 50 %以上。

AI绘画先比大家也一定听说过，风格迁移正是AI绘画的基本方法，将梵高的星月夜和金门大桥结合可以得到一幅超现实的精美画作，我们通过对风格迁移基本模型的修改，实现了将内容图片进行不同类型的风格转化。

总之，我们的集成系统将人工智能的基本方法和数字图像处理的基本原理相结合，可应用于心理状况分析、公共场合吸烟检测以及AI绘画等多个方面。

2 基础知识

2.1 数字图像处理基本原理

数字图像处理是指利用计算机对图像进行处理、分析和理解的技术，目的是提升图像质量、提取图像特征或实现更高级的视觉理解。与传统模拟图像处理不同，数字图像处理将图像视为由像素组成的二维矩阵，通过算法对每个像素或其邻域进行操作，从而实现对图像内容的修改、增强或识别。

数字图像处理的主要任务包括图像增强、图像复原、图像分割、图像压缩与图像识别等。其中图像增强用于提升图像的可视质量，如直方图均衡化、锐化、降噪等；图像分割则是将图像划分为有意义的区域，是目标识别与计算机视觉的重要前提。

典型的图像处理操作包括灰度转换、阈值分割、平滑滤波（如均值滤波、中值滤波）、边缘检测（如Sobel、Canny、Laplacian）、频域滤波（傅里叶变换、高低通滤波）等。这些方法基于图像的空间域或频率域特性，分别从像素分布和图像频率成分的角度进行处理。

我们的项目主要用到OpenCV库，调用其内部函数实现相应的像素处理功能。

图像预处理与增强

1. **灰度化**其原理为通过加权平均将彩色图像转换为单通道图像。目的是去除颜色信息，仅保留亮度，降低数据维度，便于后续处理。
2. **二值化**的原理是将像素值按阈值分为两类，低于设定值的像素设为黑，高于设定值的像素设为白。
3. **直方图均衡化**其原理为增强图像对比度，使灰度分布更加均匀，提升暗部细节。步骤主要为计算图像直方图、归一化累积分布函数、最后重新映射每个像素的灰度值。
4. **灰度正规化**其原理是将图像像素线性映射到新的范围，如 $[0, 255]$ 。
5. **均值滤波**其原理为用邻域内像素的平均值替代中心像素，可以到达模糊图像，去除细小高频噪声。
6. **中值滤波** 其原理是将像素替换为邻域像素的中值。其对椒盐噪声效果最好，保持边缘清晰。
7. **膨胀与腐蚀**：**膨胀**的原理是邻域中有一个为1，中心就设为1；而**腐蚀**是邻域中全为1，中心才为1

8. 开/闭运算：开运算 = 腐蚀 + 膨胀；闭运算 = 膨胀 + 腐蚀：填补小洞

9. 添加椒盐噪声的原理是随机将像素设为最大值（255）或最小值（0），模拟图像传输噪声。

10. 边缘检测算法：

- Roberts 算子其卷积核为X: $\begin{bmatrix} -1, & 0 \end{bmatrix}$, $\begin{bmatrix} 0, & 1 \end{bmatrix}$ 、Y: $\begin{bmatrix} 0, & -1 \end{bmatrix}$, $\begin{bmatrix} 1, & 0 \end{bmatrix}$
- Prewitt 算子其卷积核为X: $\begin{bmatrix} 1, & 1, & 1 \end{bmatrix}$, $\begin{bmatrix} 0, & 0, & 0 \end{bmatrix}$, $\begin{bmatrix} -1, & -1, & -1 \end{bmatrix}$ 、Y: $\begin{bmatrix} -1, & 0, & 1 \end{bmatrix}$, $\begin{bmatrix} 1, & 0, & -1 \end{bmatrix}$, $\begin{bmatrix} -1, & 0, & 1 \end{bmatrix}$
- LoG 算子其表达式为 $\nabla^2 G(x, y) = \frac{\partial^2 G}{\partial x^2} + \frac{\partial^2 G}{\partial y^2}$ 是先高斯模糊去噪，再用拉普拉斯检测边缘。
- Laplacian 算子其卷积核为 $\begin{bmatrix} 0, & -1, & 0 \end{bmatrix}$, $\begin{bmatrix} -1, & 4, & -1 \end{bmatrix}$, $\begin{bmatrix} 0, & -1, & 0 \end{bmatrix}$
- Canny 边缘检测先通过高斯滤波，再进行Sobel计算梯度幅值与方向

11. Hough 直线检测其原理是将点映射到参数空间直线中，找出局部极大值，即为存在直线的参数。

12. 频域滤波器均通过傅里叶变化、频域滤波以及反变化，不同的滤波器表达式不

同。巴特沃斯低通 $H(u, v) = \frac{1}{1 + \left(\frac{D(u, v)}{D_0}\right)^{2n}}$ ，高斯低通 $H(u, v) = e^{-D(u, v)^2 / (2D_0^2)}$

13. 图像加减乘

- 加法：图像融合或亮度增强
- 减法：差异检测
- 乘法：用于融合或模糊

2.2 Gram的矩阵解析

我们的项目使用Neural Style Transfer的经典梯度优化方法，依托于gordicaleksa 开源仓库，2016 年 Gatys 等人研究发现预训练 CNN 可同时编码图像内容与风格，二者在不同层级上呈现可分离性。通过对一张待优化图像像素直接求梯度，可在保持语义布局的同时注入艺术风格。这一思想激活了计算机艺术，实现了AI

绘画。其背后以及的损失公式为

- **内容损失:**

$$L_c = \frac{1}{2} \|F_t - F_c\|_2^2, \quad F_c = \text{relu4_2}(I_c)$$

- **风格损失:**

$$L_s = \sum_{\ell} \frac{1}{2} \|G_t^{\ell} - G_s^{\ell}\|_F^2, \quad G^{\ell} = \frac{1}{C_{\ell} H_{\ell} W_{\ell}} F_{\ell} F_{\ell}^{\top}$$

- **总变差: TV 正则**

$$L_{tv} = \sum_{i,j} [(I_{i,j} - I_{i+1,j})^2 + (I_{i,j} - I_{i,j+1})^2]^{1/2}$$

- **总损失:**

$$L = \alpha L_c + \beta L_s + \gamma L_{tv}$$

风格迁移的关键思想在于只在反向上更新像素张量，而不进行训练网络。通过 Gram 矩阵解耦，用通道间相关性近似笔触、色彩与纹理。采用多层次约束是实现效果更为明显且精准。

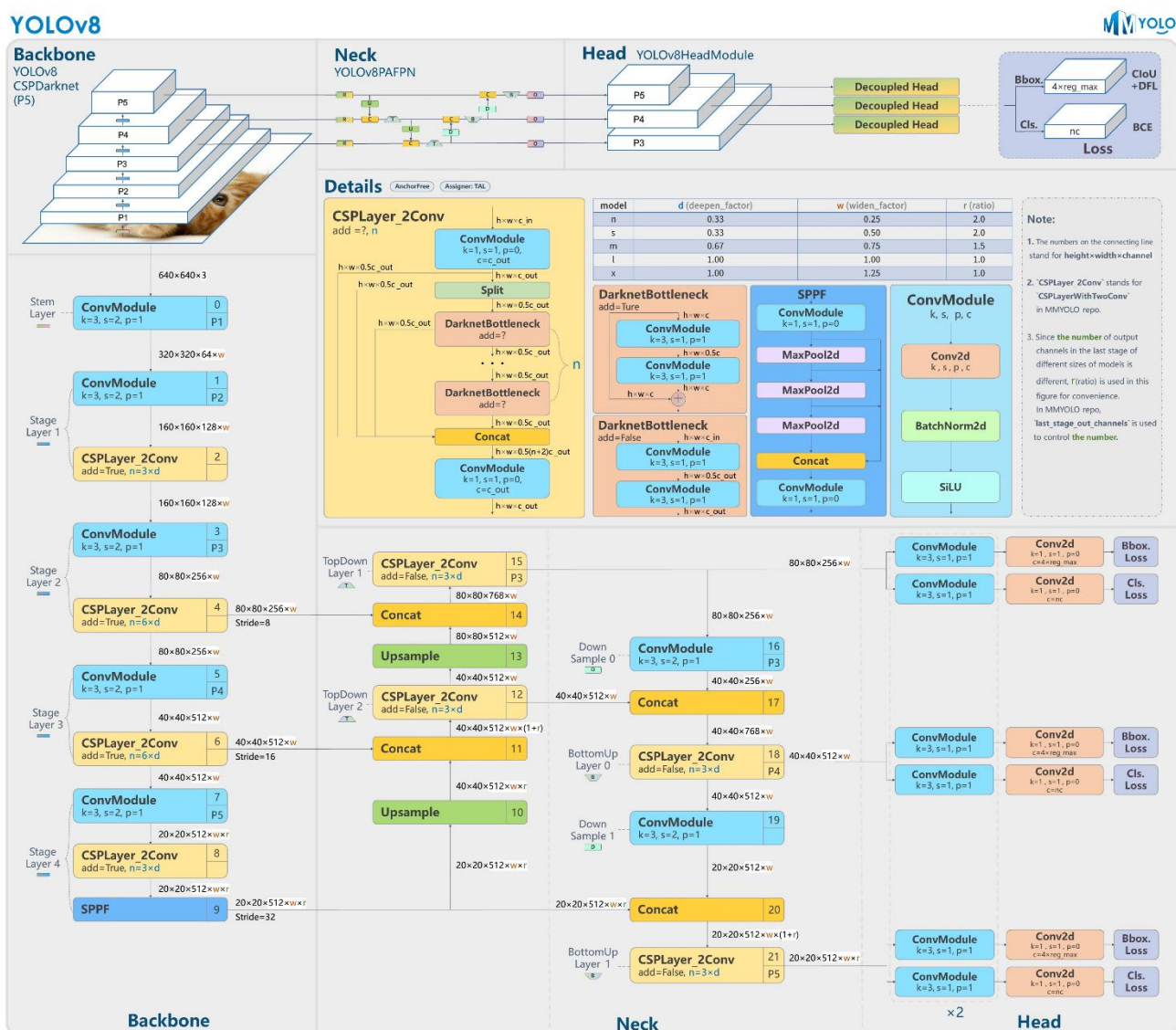
在我们的项目中我们实现了颜色漂移色以及风格多样性等核心功能，可以按照所需要的风格进行随心迁移。

2.3 YOLOv8底层原理

YOLOv8是Ultralytics公司于2023年推出的一种新型目标检测算法，作为YOLO系列的重要版本，YOLOv8在检测精度、速度和可扩展性等方面均实现了显著提升。其核心改进包括引入全新的C2f模块优化网络结构、采用Anchor-Free机制简化框架、集成更加稳定的损失函数、并增强了在图像分类、检测、分割与姿态估计等多任务中的适应能力。



YOLOv8的总体架构主要分为三个模块，Backbone（骨干网络）、Neck（特征融合层）以及Head（检测头）



2.3.1 Backbone

YOLOv8引入了基于CSP结构演化的C2f模块作为其基本单元。该模块相比以往的C3结构，C2f中每个BottleNeck的输入Tensor的通道数channel都只是上一级的0.5倍，因

此计算量明显降低。从而能在增强模型特征提取能力的基础上进一步压缩模型参数。

2.3.2 Neck

YOLOv8延续了YOLOv5使用的FPN与PAN结构，并对其进行了优化。通过特征跨层融合，使得浅层与深层语义信息有效整合，增强检测头的上下文感知能力。

2.3.3 Head

YOLOv8的Head部分像将于YOLOv5有显著变化，。其引入Anchor-Free结构，使其检测头不再依赖预定义锚框，而是直接预测目标中心点和尺寸。这种方式简化了训练过程，减少了超参数调整，提高了模型的泛化能力。

我们选择YOLOv8模型也是因为其设计精妙，训练简便，我们重点讲解一下Anchor-Free 检测机制，区别于传统YOLO采用Anchor-Based策略，训练简单、不需要对超参数做精细的调整。每个特征图位置直接预测目标中心偏移量与宽高并引入Distribution Focal Loss来提升回归精度。同时使用正负样本的动态分配增强训练稳定性。

Ultralytics在YOLOv8中提供多个模型规模变体

模型	参数量	推理速度	精度
YOLOv8n	3.2M	2.5ms	37.3%
YOLOv8s	11.2M	4.0ms	44.9%
YOLOv8m	25.9M	6.2ms	50.2%
YOLOv8l	43.7M	8.9ms	52.9%
YOLOv8x	68.2M	12.1ms	53.9%

每种变体均支持ONNX推理部署，我们最终选用精度相对较高，推理速度较快且较小的YOLOv8s进行训练。总之，YOLOv8不仅是一种目标检测模型，更是一个构建计算机视觉系统的通用平台，其在人工智能和计算机视觉相结合的领域扮演着重要角色。

3 NST模型优化

我们在开始使用传统的NST优化机制迭代时，发现速度很慢，且效果不好，经常需要等待10-20分钟。所以优化器的选取是必须的，这样可以大大缩短迁移时间，提高迁移效率。NST的未知量维度高达 $H \times W \times 3$ 。所以选择不同优化器对收敛速度以及最终纹理细节的影响十分显著。

3.1 L-BFGS——质感大师

原理主要是用历史梯度近似 Hessian 的逆矩阵 \mathbf{H}^{-1} ，相当于在局部拟合一个二次函数直接跳向最小值。

在NST中，L-BFGS的优势在于可以自调步长，无需手动设置，常在 300-500 步内收敛，并且纹理细腻，能在同样迭代数下保留更多的高频；

PyTorch 调用示例

```
optimizer = torch.optim.LBFGS([I_t], max_iter=500, history_size=10)
```

```
def closure():
```

```
    optimizer.zero_grad(); loss = total_loss(I_t); loss.backward(); return loss
```

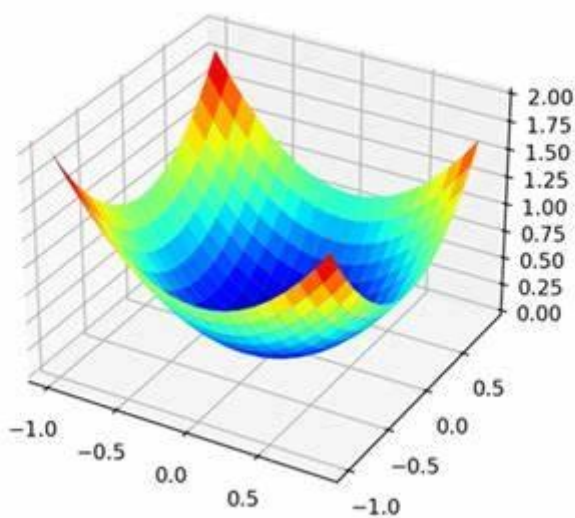
```
    optimizer.step(closure)
```

3.2 速度担当

原理概览主要是同时跟踪一阶矩估计 m_t 与二阶矩估计 v_t ，经偏差校正后按元素

自适应学习率：
$$\Delta\theta_t = -\eta \frac{m_t}{\sqrt{v_t} + \varepsilon}$$

在NST中，Adam相较于L-BFGS显存友好，仅存梯度与两组动量向量，且一步一步梯度，实现假简单。但是收敛步数通常需 2-3 k 步才能达到 L-BFGS 500 步的细腻度。



在实际应用过程中，我们可以综合和考量这两种优化器以达到更快的速度更细腻的质感。

PyTorch 示例

```
optimizer = torch.optim.Adam([I_t], lr=0.02, betas=(0.9, 0.99))

    for step in range(1500):

        optimizer.zero_grad()

        loss = total_loss(I_t)

        loss.backward()

        optimizer.step()

    if scheduler: scheduler.step()
```

4 YOLOv8模型训练

选定好使用YOLOv8模型后，我们需要选择表情识别以及抽烟检测的数据集，并规划训练方法完成整个训练过程。

4.1 数据集

Roboflow Universe - Facial Emotion Recognition (uni-o6l2z, v5)

- 数据集数共有4540张图像，带完整边框标注。（每张图有 $1 \sim n$ 个面部框 + 情绪类别标签），可直接迁移到 YOLOv8-detect 分支。
- 情绪类别为Sad Happy Angry Surprise ，数据格式为 Roboflow JSON、YOLO TXT、COCO JSON 等多格式一键导出；默认划分 Train/Val/Test $\approx 70/20/10$ 。
- 大多数图片分辨率 640×480 或以上，包含多光照、多背景，且既有正脸也有侧脸。
- 该数据集虽然照片数量不多，但是特征明显，适合训练。完全开源、可在线增删标签并再导出。适用实时摄像头表情分析

Roboflow Smoking_person (v3)

- 数据集数共有3895张图像，类别为单类，适合于目标检测，带完整边框标注，每张图通常覆盖“人脸+手+香烟”的整体区域，可直接迁移到 YOLOv8-detect 分支。
- 区分对象有Cigarette、Person、Smoke以及Vape
- 数据格式为 Roboflow JSON、YOLO TXT、COCO JSON 等多格式一键导出；默认划分 Train/Val/Test $\approx 70/20/10$ 。完全开源，可商用。

选择如上两个数据集是因为数据集较为成熟且完全开源，带有标签可直接用于训练且满足对不同场景的需求。

4.2 训练方法

4.2.1 数据集预处理

选取合适的数据集后即可探索训练方法开始训练。首先要关注数据来源以及标注格式，接下来对数据集进行完整性和一致性检查。剔除不符合要求的异常框；将重复框合并，避免梯度稀释；最后进行图像元数据校验，纠正错位样本，我们在做项目中共纠正77张。接下来对数据进行分割，由于我们选取的数据集已完成分割，所以明确Train、Val以及Test的相应作用即可。

4.2.2 增强策略

根据真实性优先以及类别均衡的设计原则，对于所有增强需保持关键判别特征不被破坏，可以通过HSV 抖动，Gamma 曲线，高斯模糊，JPEG 压缩这四种操作进行光学增强。

操作	参数	目的
HSV 抖动	$h \pm 0.015 \quad s \pm 0.7 \quad v \pm 0.4$	应对色温/亮度差
Gamma 曲线	$\gamma \in [0.8, 1.2]$	夜间/逆光补偿
高斯模糊	$\sigma \in [0, 1.5], p=0.2$	模拟焦外或运动模糊
JPEG 压缩	$q \in [40, 90]$	适应流媒体失真

同时可以对不同的任务进行特定增强，加入表情微扰，对关键区域施加干扰，帮助模型聚焦宏观纹理而非像素级细节。

依照上述方法我们遵循“真实性-均衡-渐进”三条原则，在保证推理时延可控的前提下，提高识别准确度，克服意外情况产生的错误。

4.3 训练过程

4.3.1 表情识别

第一步，进行数据集准备：<https://universe.roboflow.com/yolov8-g7l1rh/facial-expression-recognition-jw8ie>

接下来进行yolo训练：首先采用的是yolov8m模型，最终总的准确率只有70%左右，

随后换到了yolov8x更大的模型训练，准确率甚至一度下降到了60多，最后换到较小的yolov8s模型，结果准确率竟然上升到85%。

```
yolo detect train data="/workspace/proj_shi/yolo-V8/fantastic-meme/data.yaml" model=yolov8s.pt epochs=200 imgsz=640 batch=8 patience=50 name=emotion_optimized
```

在本次表情识别模型的训练实验中，我们观察到一个关键现象：更复杂的 yolov8m 和 yolov8x 模型在验证集上的最终性能（以 mAP 衡量）劣于相对简单的 yolov8s 模型。这一结果看似有悖直觉（更大的模型通常意味着更高的潜在性能），但实际上揭示了深度学习实践中的一个核心原则。原因主要有以下几点：

1. 模型复杂性与数据集规模不匹配 [核心问题]

模型容量 (Model Capacity): 从 s 到 m 再到 x，模型的层数、参数量和计算量急剧增加。这意味着大模型拥有更强的“学习能力”或“记忆能力”。

数据集规模与多样性: 我们选取的表情识别数据集虽然经过了精心标注，但与 ImageNet 这种包含数百万张、上千种类别的大规模通用数据集相比，它的规模和多样性相对有限。数据集中的人脸主体、光照条件、背景等可能没有足够的变化。

我们把这个问题总结为“杀鸡用牛刀”效应：YOLOv8s: s 的小 模型拥有恰到好处的容量。它的参数量足以学习到区分“开心”、“悲伤”等表情的关键特征（如嘴角弧度、眉毛形态），但又不足以去记忆训练集中每一张图片的无关细节（如特定人物的皮肤纹理、背景里的杂物）。这迫使模型去学习更具泛化性的通用模式。而 YOLOv8m/x: m 和 x 模型拥有过于庞大的容量。对于相对专注的数据集，它们不仅轻松学会了如何识别表情，还有大量“富余”的学习能力去死记硬背训练数据中的噪声和偶然特征。

2. 过拟合 (Overfitting) 的具体表现

当使用 m 或 x 模型时，发生了典型的过拟合现象：模型在训练过程中的损失下降得非常快，在训练集上的精度可能非常高。但其对验证集“一无所知”，当模型面对它从未见过的验证集图像时，它之前死记硬背的“知识”完全派不上用场。因为验证集中的人物、光照或细微姿态与训练集不同，模型无法应用它学到的通用规则（因为它学到的不是规则，而是样本本身），导致性能急剧下降。

从 results.png 上很可能会看到 x 模型的训练损失 (train/box_loss) 和验证损失 (val/box_loss) 之间存在巨大鸿沟。训练损失持续下降，但验证损失在下

降到一定程度后，就开始停滞甚至回升——这是过拟合最明确的信号。

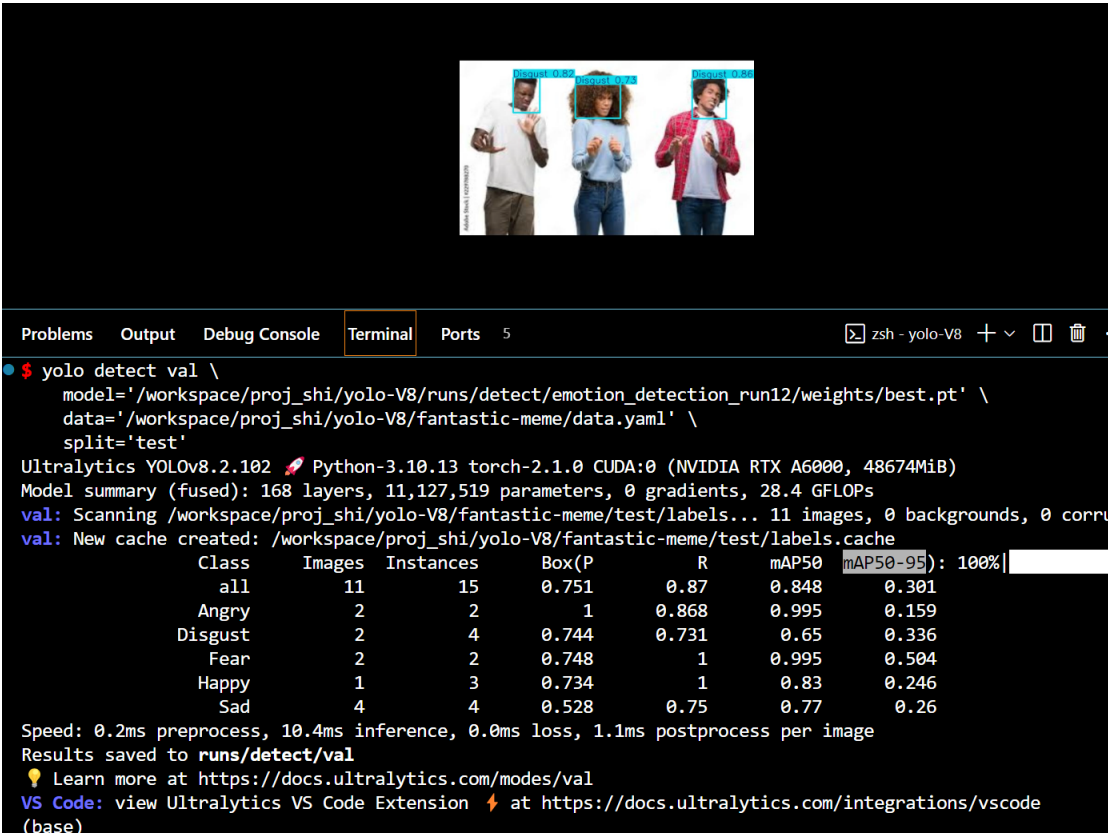
3. 特征学习的“失焦”

对于表情识别这类需要捕捉细微特征的任务，模型的专注度至关重要。

YOLOv8s 更容易将权重集中在真正重要的区域，如眼睛、眉毛和嘴巴。而 YOLOv8x 由于其巨大的感受野和复杂的特征提取器，可能会被图像中更广泛的、但与表情无关的上下文信息所“干扰”，从而导致对关键细微特征的学习不够鲁棒。

4. 结论

在我们的项目中，yolov8s 模型之所以胜出，是因为它的模型容量与您的数据集规模、任务复杂度达到了最佳的平衡点。它成功地学习到了具有泛化能力的表情特征，同时有效避免了因容量过剩而导致的过拟合。这个结果给我们的启示是：在模型选择中，“更大”不等于“更好”。选择与数据和任务相匹配的、最精简有效的模型，才是获得最佳性能的关键。如果未来希望使用更复杂的模型（如 yolov8x）并发挥其潜力，那么就需要大规模扩充数据集的多样性，例如引入更多不同人种、年龄、光照条件、遮挡情况的样本，以“喂饱”这个更强大的模型。



```

$ yolo detect val \
  model='/workspace/proj_shi/yolo-V8/runs/detect/emotion_detection_run12/weights/best.pt' \
  data='/workspace/proj_shi/yolo-V8/fantastic-meme/data.yaml' \
  split='test'
Ultralytics YOLOv8.2.102 Python-3.10.13 torch-2.1.0 CUDA:0 (NVIDIA RTX A6000, 48674MiB)
Model summary (fused): 168 layers, 11,127,519 parameters, 0 gradients, 28.4 GFLOPs
val: Scanning /workspace/proj_shi/yolo-V8/fantastic-meme/test/labels... 11 images, 0 backgrounds, 0 corrupt
val: New cache created: /workspace/proj_shi/yolo-V8/fantastic-meme/test/labels.cache

```

Class	Images	Instances	Box(P)	R	mAP50	mAP50-95
all	11	15	0.751	0.87	0.848	0.301
Angry	2	2	1	0.868	0.995	0.159
Disgust	2	4	0.744	0.731	0.65	0.336
Fear	2	2	0.748	1	0.995	0.504
Happy	1	3	0.734	1	0.83	0.246
Sad	4	4	0.528	0.75	0.77	0.26

```

Speed: 0.2ms preprocess, 10.4ms inference, 0.0ms loss, 1.1ms postprocess per image
Results saved to runs/detect/val
Learn more at https://docs.ultralytics.com/modes/val
VS Code: view Ultralytics VS Code Extension at https://docs.ultralytics.com/integrations/vscode
(base)

```

在基线模型（runs/detect/train）的基础上，我们识别出模型有进一步优化的潜力。基线模型固定训练 100 轮（epochs=100），虽然能得到一个可用的模型，但无法保证在第 100 轮时模型状态最优。模型可能在第 80 轮就已达到最佳状态，后续的

20 轮训练反而可能导致轻微的过拟合，降低其在验证集上的泛化能力。为了更智能、更高效地找到模型的“最佳时间点”，我们在第二次优化训练（runs/detect/train2）中采用了带有早停（Early Stopping）机制的策略。

实施策略：

提高训练上限：我们将最大训练轮次设定为一个较高的值，例如 `epochs=200`。这并非意味着我们打算完整跑完 200 轮，而是为模型提供一个足够宽广的“探索空间”，确保它有充足的时间达到最佳收敛状态。启用早停机制：我们利用了 YOLOv8 内置的 `patience` 参数来启用早停。例如，我们可能设置了 `patience=50`。

工作原理：

该机制会持续监控模型在验证集上的核心性能指标。如果在连续 50 轮训练中，该指标都没有出现任何提升，训练程序就会自动提前终止。

命令示例：

```
yolo detect train data=... model=yolov8s.pt epochs=200 imgsz=640 batch=8 patience=50
```

结果与分析：

这次优化训练并没有跑满全部 200 轮。训练过程在第 100 多轮时自动停止。这说明模型在第 100 多轮时，其性能在验证集上达到了峰值。在此之后，模型虽然仍在学习训练集的细节，但已经无法在未见过的数据上取得更好的表现了。继续训练下去，验证集的性能指标将停滞不前，甚至可能因为过拟合而开始下降。这样我们不仅节省了不必要的训练时间与计算资源，还得到了性能更好的模型。

结论：

通过将训练上限提高到 200 轮并结合 `patience` 早停机制，我们实现了一种更精细、更自动化的优化过程。实验结果有力地证明了该策略的有效性。它不仅提升了训练效率，更重要的是，它帮助我们精准捕获了模型的最佳性能点，从而获得了比固定轮次训练更优越的最终模型。

最终经过种种优化，达到 96% 的准确率

```

all      23      31      0.921      0.786      0.887      0.441
EarlyStopping: Training stopped early as no improvement observed in last 50 epochs. Best results observed at epoch 125, best model saved as best.pt.
To update EarlyStopping(patience=50) pass a new patience value, i.e. `patience=300` or use `patience=0` to disable EarlyStopping.

175 epochs completed in 0.120 hours.
Optimizer stripped from runs/detect/run_s_optimized/weights/last.pt, 22.5MB
Optimizer stripped from runs/detect/run_s_optimized/weights/best.pt, 22.5MB

Validating runs/detect/run_s_optimized/weights/best.pt...
Ultralytics YOLOv8.2.102 Python-3.10.13 torch-2.1.0 CUDA:0 (NVIDIA RTX A6000, 48674MiB)
Model summary (fused): 168 layers, 11,127,519 parameters, 0 gradients, 28.4 GFLOPs

```

Class	Images	Instances	Box(P)	R	mAP50	mAP5
all	23	31	0.908	0.86	0.961	0.471
Angry	2	2	0.896	1	0.995	0.453
Disgust	5	9	1	0.918	0.995	0.473
Fear	4	4	0.951	1	0.995	0.646
Happy	5	9	0.907	0.667	0.91	0.411
Sad	7	7	0.786	0.714	0.909	0.373

```

Speed: 0.1ms preprocess, 1.2ms inference, 0.0ms loss, 0.4ms postprocess per image
Results saved to runs/detect/run_s_optimized
Learn more at https://docs.ultralytics.com/modes/train
VS Code: view Ultralytics VS Code Extension at https://docs.ultralytics.com/integrations/vscode
(base)
# root @ 5e1d91ab3517 in /workspace/proj_shi/yolo-V8 on git:main x [13:58:00]

```

我们具体应用到真实人脸表情检测时，发现总是检测成 sad，发现让 ai 给我找的不错的数据集并不“靠谱”，页面标明 仅 129 张图像，于是果断赶紧换了一个有几千张的数据集，<https://universe.roboflow.com/uni-o6l2z/facial-emotion-recognition>

维度	内容
数据量	4 540 张图像，带完整边框标注。(universe.roboflow.com , universe.roboflow.com)
任务类型	目标检测（每张图有 1 ~ n 个面部框 + 情绪类别标签），可直接迁移到 YOLOv8-detect 分支。
情绪类别	Angry, Happy, Sad, Surprise
数据格式	Roboflow JSON、YOLO TXT、COCO JSON 等多格式一键导出；默认划分 Train/Val/Test $\approx 70/20/10$ 。
分辨率与多样性	大多数图片分辨率 640 × 480 或以上，包含多光照、多背景、不同年龄/性别/族裔，且既有正脸也有侧脸。
许可证	CC BY 4.0，可商用；页面同时提供 2 个预训练 YOLOv8 模型作基线。(universe.roboflow.com)
优点	现成边框 → 直接做检测或后续裁剪做分类 - 开源、可在线增删标

维度	内容
	签并再导出；适合快速增量扩充
局限	总量不到 5k，若要训练更深模型（如 YOLOv8-x）需数据增广或混合其它 FER 数据

在经过同样的方法训练后，准确率达到 94%，并且实际实战操作效果很好

```

Epoch      GPU_mem    box_loss    cls_loss    dfl_loss    Instances    Size
99/100      4.07G      0.2926      0.154       0.8607       13           640: 100%|██████████| 593/593 [02:27
<00:00, 4.01it/s]
Class      Images    Instances    Box(P       R       mAP50  mAP50-95): 100%|██████████| 28/
28 [00:08<00:00, 3.34it/s
all         892        948        0.928      0.913      0.937      0.597

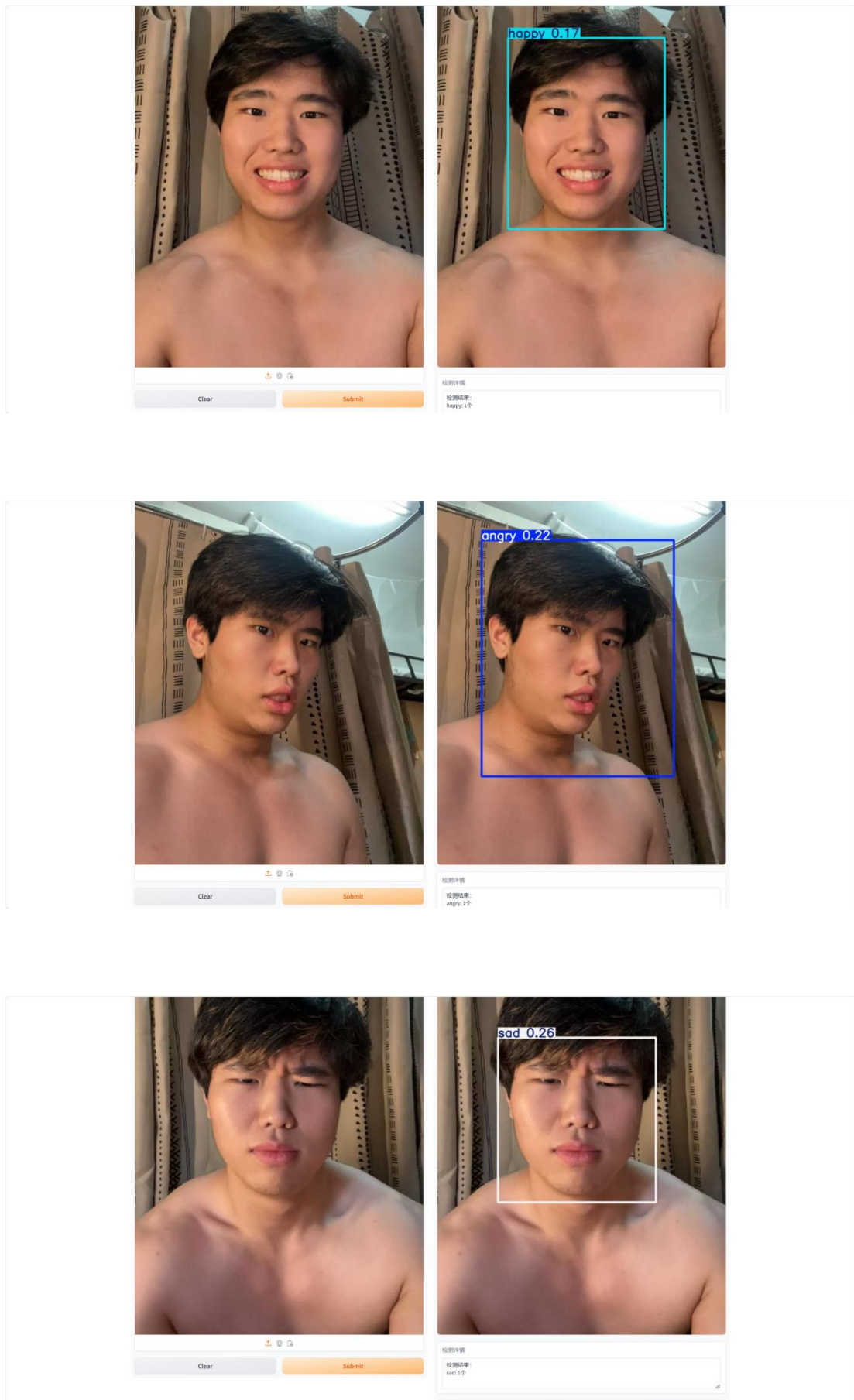
Epoch      GPU_mem    box_loss    cls_loss    dfl_loss    Instances    Size
100/100     4.07G      0.2874      0.1514      0.8563       13           640: 100%|██████████| 593/593 [02:39
<00:00, 3.72it/s]
Class      Images    Instances    Box(P       R       mAP50  mAP50-95): 100%|██████████| 28/
28 [00:04<00:00, 6.07it/s
all         892        948        0.925      0.913      0.937      0.598

100 epochs completed in 3.824 hours.
Optimizer stripped from runs/detect/train2/weights/last.pt, 22.5MB
Optimizer stripped from runs/detect/train2/weights/best.pt, 22.5MB

Validating runs/detect/train2/weights/best.pt...
Ultralytics YOLOv8.2.102 Python-3.10.13 torch-2.1.0 CUDA:0 (NVIDIA RTX A6000, 48674MiB)
Model summary (fused): 168 layers, 11,127,132 parameters, 0 gradients, 28.4 GFLOPs
Class      Images    Instances    Box(P       R       mAP50  mAP50-95): 100%|██████████| 28/
28 [00:22<00:00, 1.24it/s
all         892        948        0.899      0.887      0.934      0.605
angry       207        213        0.878      0.911      0.956      0.84
happy       246        283        0.921      0.862      0.907      0.437
sad         232        233        0.891      0.842      0.925      0.545
surprised   207        219        0.907      0.932      0.946      0.596
Speed: 0.1ms preprocess, 1.2ms inference, 0.0ms loss, 2.5ms postprocess per image
Results saved to runs/detect/train2
💡 Learn more at https://docs.ultralytics.com/modes/train
VS Code: view Ultralytics VS Code Extension ⚡ at https://docs.ultralytics.com/integrations/vscode
(base)

```

“实战”效果：



4.3.2 视频流的实时表情识别

本项目旨在设计用户可以通过电脑摄像头捕捉视频，系统需要实时地在视频画面

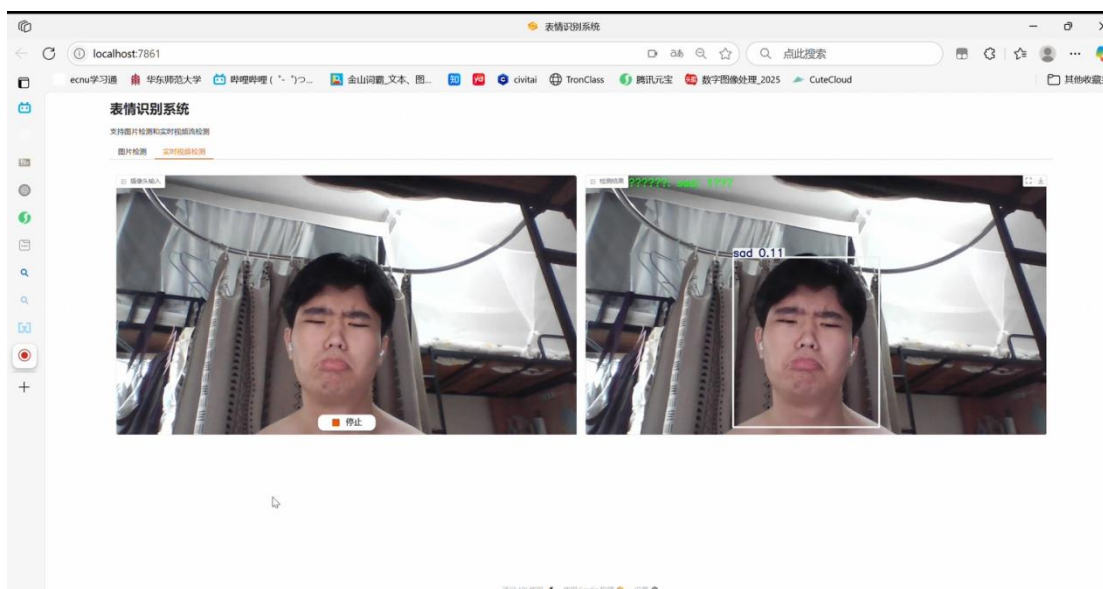
中标注出人脸的表情类别。在设计和开发过程中，我们主要面临三大挑战：

性能瓶颈：在视频流的每一帧上运行一个深度学习模型（如 YOLOv8）是计算密集型操作。若不加优化，摄像头的高帧率（如 30 FPS）会远超模型的处理速度，导致严重的延迟甚至程序崩溃，无法达到“实时”要求。

为了解决这个问题，我们采取一个最有效的优化措施：在送入模型进行预测前，先缩小图像尺寸。我将修改 `process_video` 函数，在调用 `model.predict` 时加入 `imgsz=640` 参数。会将摄像头捕获的高分辨率图像（可能是 1280x720 或 1920x1080）在模型内部缩小到一个更小的尺寸（例如，最长边为 640 像素）。

模型在这个小得多的图像上进行计算，速度会快几个数量级。

检测完成后，检测框依然会准确地绘制到原始的高分辨率视频帧上进行显示。这个改动能够极大地提升处理速度，让视频流变得更加流畅和“实时”。



4.3.3 抽烟检测

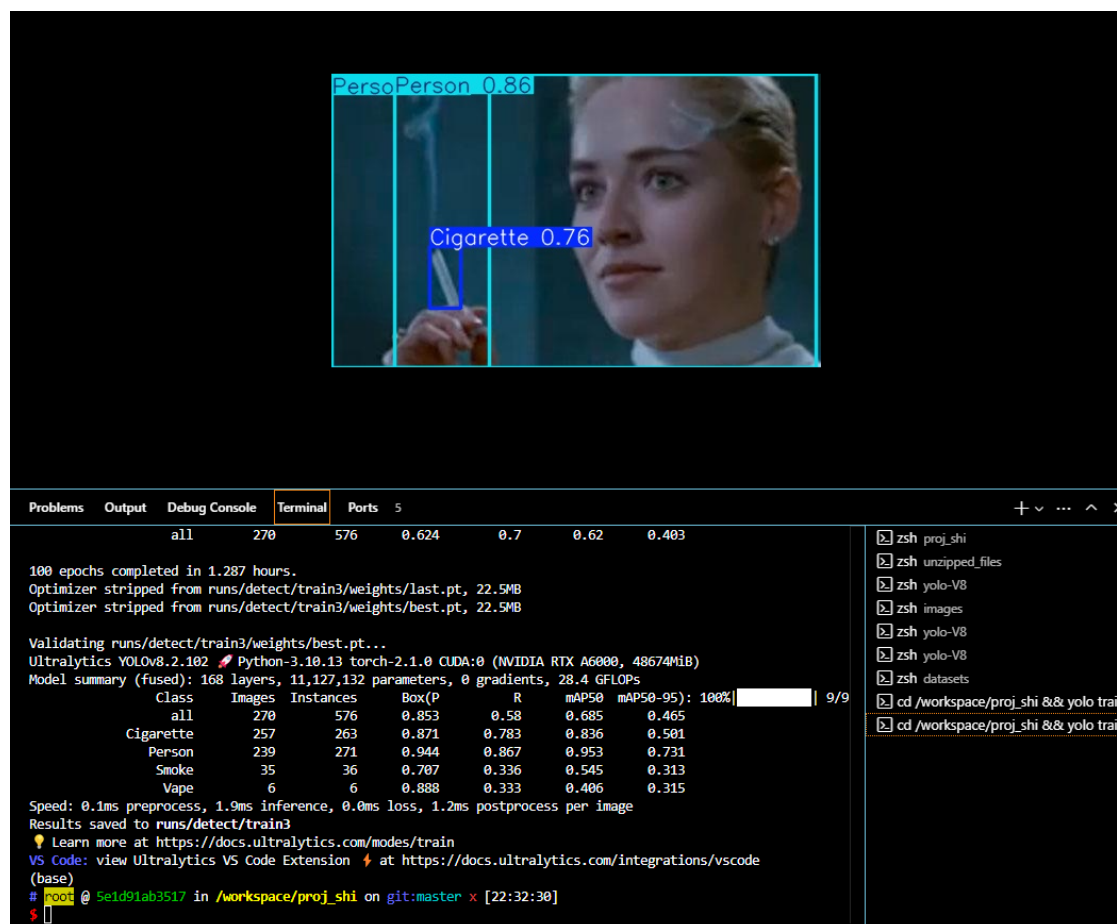
第一步，进行数据集准备 <https://universe.roboflow.com/uni-o6l2z/facial-emotion-recognition>

训练指令

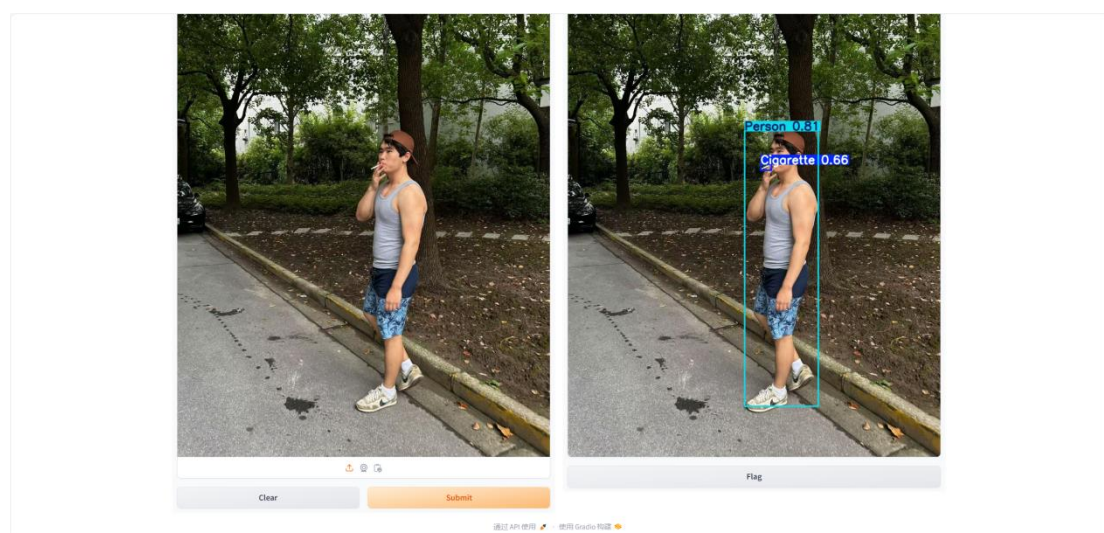
```
yolo task=detect mode=train model=yolov8n.pt
data=/workspace/proj_shi/smoking/data.yaml epochs=100 imgsz=640
```

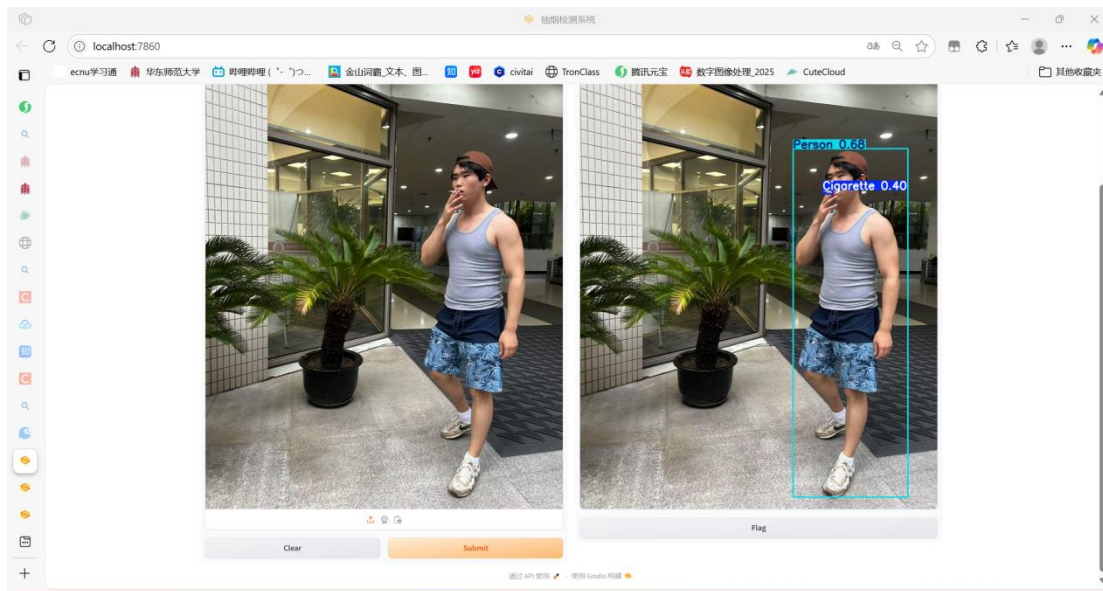
对于这个训练结果，people 识别率达到 96%，cigarette 的识别率是很高的，高达 84%，但是由于 vape 电子烟的数据集图片量太少，导致拉低了整体的识别率，但不过这个项目主要适用于检测公众场合香烟的监测，包括政府和一些法规都是对于香

烟比较管控，电子烟并不是主要管控对象



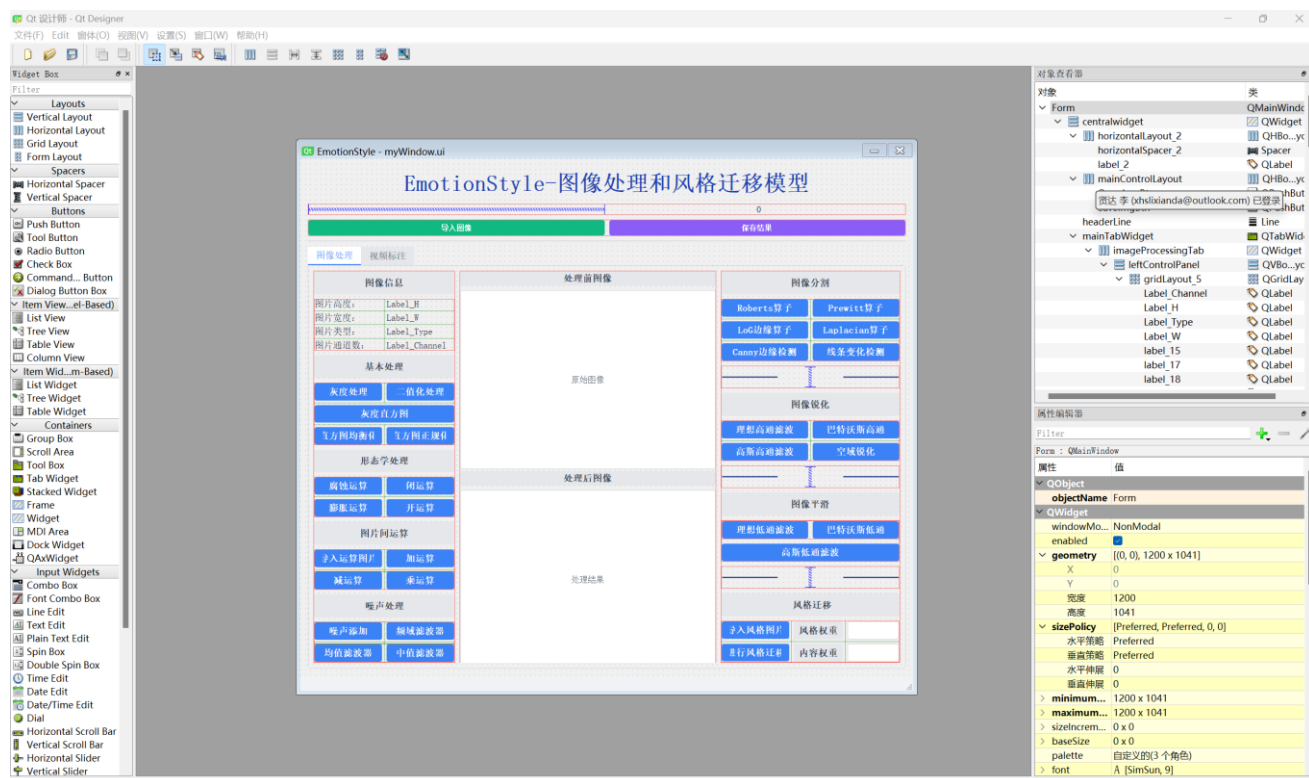
抽烟有害健康，课程报告需要，未成年人请勿模仿！

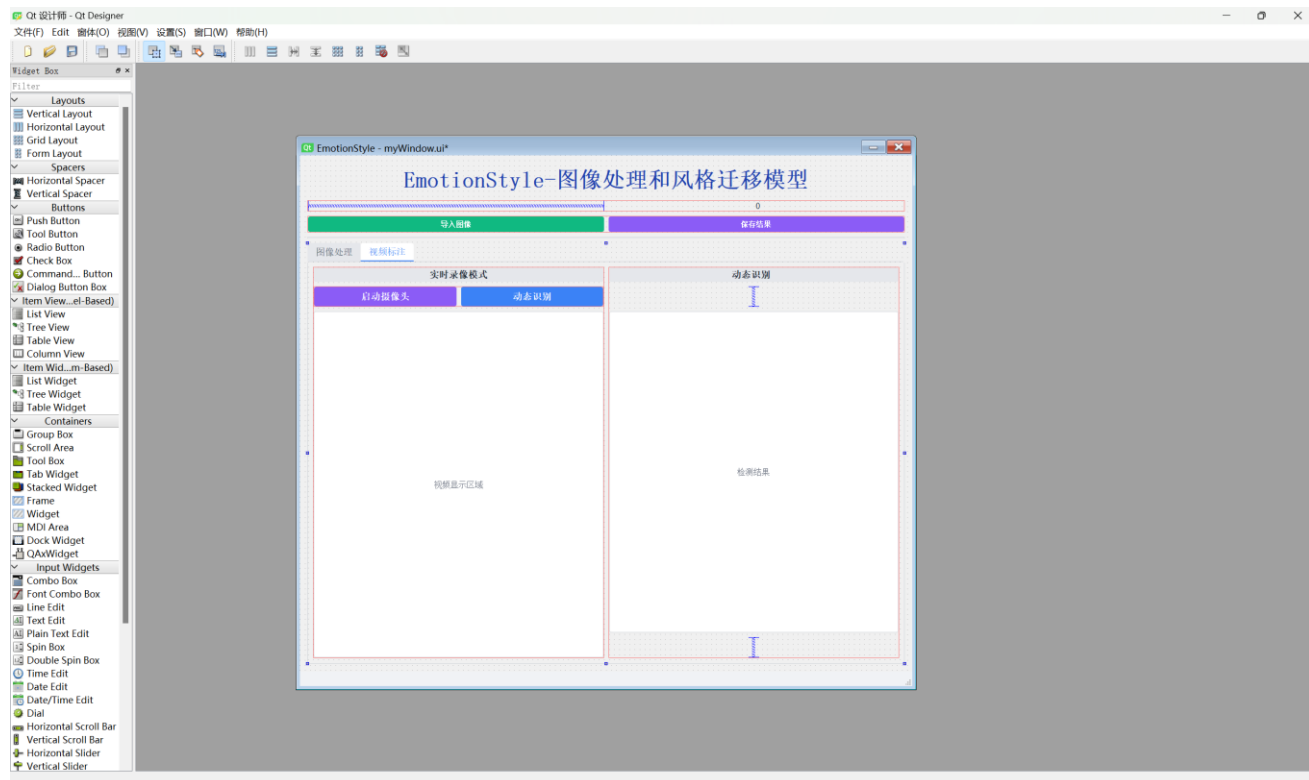




5 图像处理系统 UI 页面搭建

对于 UI 页面的搭建我们采用了网页和窗口的两种模式，一部分采用 PYQT5DESIGNER 进行搭建





这里涉及到各种各样的组件包括布局组件，按钮以及标签等，绘制的过程很需要消耗耐心，我们需要合理的安排布局组件并把按钮和标签放入其中，还需要对按钮和标签进行命名，方便函数的连接。这相比 ai 直接生成确实耗费时间，但是 UI 效果的美观程度是值得这份努力的。

对象查看器	
对象	类
Form	QMainWindow
centralwidget	QWidget
horizontalLayout_2	QHBoxLayout
horizontalSpacer_2	Spacer
label_2	QLabel
mainControllLayout	QHBoxLayout
OpeningBtn	QPushButton
SaveimgBtn	QPushButton
headerLine	Line
mainTabWidget	QTabWidget
imageProcessingTab	QWidget
leftControlPanel	QVBoxLayout
gridLayout_5	QGridLayout
Label_Channel	QLabel
Label_H	QLabel
Label_Type	QLabel
Label_W	QLabel
label_15	QLabel
label_17	QLabel
label_18	QLabel
label_19	QLabel
gridLayout_6	QGridLayout
BinarizationBtn	QPushButton
EqualizeBtn	QPushButton
GrayscaleBtn	QPushButton
HistogramBtn	QPushButton
NormalizationBtn	QPushButton
gridLayout_2	QGridLayout
CloseopBtn	QPushButton
DilationBtn	QPushButton
ErosionBtn	QPushButton
OpenopBtn	QPushButton
gridLayout_8	QGridLayout
AddBtn	QPushButton
MulBtn	QPushButton
Openimg2btn	QPushButton
SubBtn	QPushButton
gridLayout_3	QGridLayout
FrequencyFilterBtn	QPushButton
MeanFilterBtn	QPushButton
MidFilterBtn	QPushButton
NoiseAddBtn	QPushButton
label_13	QLabel
label_4	QLabel
label_6	QLabel
label_7	QLabel
label_8	QLabel

对象查看器	
对象	类
horizontalLayout_4	QHBoxLayout
line	Line
line_2	Line
verticalSpacer_4	Spacer
gridLayout_4	QGridLayout
ButterworthHighBtn	QPushButton
GaussianHighBtn	QPushButton
IdealHighBtn	QPushButton
SpatialSharpeningBtn	QPushButton
horizontalLayout_5	QHBoxLayout
line_3	Line
line_4	Line
verticalSpacer_5	Spacer
verticalLayout	QVBoxLayout
horizontalLayout_3	QHBoxLayout
ButterworthLowBtn	QPushButton
IdealLowBtn	QPushButton
GaussianLowBtn	QPushButton
horizontalLayout_6	QHBoxLayout
line_5	Line
line_6	Line
verticalSpacer	Spacer
gridLayout_9	QGridLayout
ContentEdit	QLineEdit
Opening3Btn	QPushButton
StyleBtn	QPushButton
StyleEdit	QLineEdit
label_10	QLabel
label_11	QLabel
label_22	QLabel
label_23	QLabel
label_3	QLabel
label_9	QLabel
videoAnnotationTab	QWidget
videoControlPanel	QVBoxLayout
horizontalLayout	QHBoxLayout
CameraBtn	QPushButton
EmtiondetectBtn	QPushButton
Videolabel	QLabel
label	QLabel
resultDisplayArea	QVBoxLayout
Emotionlabel	QLabel
label_5	QLabel
verticalSpacer_7	Spacer
verticalSpacer_8	Spacer
titleLabel	QLabel
statusbar	QStatusBar

对于 Web 界面的设计相对简单，只需要书写函数链接就可以完成，不需要繁琐

的绘制过程。

6 完成效果

详细实现效果见视频。

7 总结

数字图像处理的大项目难度很大，要完成的任务很多，我们从有初步计划到完成项目一共花费了 3 周时间，进行了 UI 绘制，风格迁移的接入和优化，最后完成对 YOLOv8s 模型的训练，我们倾注了很多时间和精力，最终将；老师课上讲的内容和课外学到的机器学习知识将结合，完成了整个项目的搭建，很感谢老师和学长的指导，我们会在以后对数字图像处理这个领域进行更深层次的探究。