

视频处理实践

程序清单

- detect_face_convert_casme2_to_yolo.py
- train_micro_expression_casme2_for_yolov8.py
- Yolov8_flask_client_repo_image_autodl.py
- Yolov8_flask_client_repo_text_autodl.py
- YOLOv8_Flask_server.py

第一步：使用 YOLOv8 和 CASME II 数据训练微表情识别模型

获取 CASME II 数据集

签 pdf 上传：官方邮件给出数据集：

License Agreement	
Chinese Academy of Sciences Micro-expression Database II (CASME2) I agree • to use the video or images for research purposes only. • not to provide the video or images to second parties. • if I reproduce images and video in electronic or print media, to use only in scientific journals and include notice of copyright (©Xiaolan Fu). Images and video from sub12 and sub22 should not be published.	You could find below the download link. FTP://71.111.111.111 ID: PW: The ftp transmission requires the specific software instead of browsers. Please be aware that the database should be only used for your personal use. Any commercial use is strictly prohibited. Best regards, Jingting Li

数据预处理

GPU 服务器目录结构:

源数据:

~/ultralytics/

```

├── casme2/
│   ├── CASME2_RAW/
│   │   └── sub01
│   ...
│   ...
│   ├── convert_casme_to_yolo.py
│   └── zyw_custom_yolov8.yaml

```

```
|   └── pinggu.py
|   └── CASME2-coding-20250308.xlsx
```

转换结果存储目录结构:

CASME2_YOLO/

```
|   └── images/
|       ├── train/
|       └── val/
└── labels/
    ├── train/
    └── val/
```

转换为 YOLOv8 可用的格式的程序:

```
import os
import cv2
import pandas as pd
import numpy as np
from tqdm import tqdm
import mediapipe as mp
from sklearn.model_selection import train_test_split

# 初始化MediaPipe面部检测
mp_face_detection = mp.solutions.face_detection
face_detection = mp_face_detection.FaceDetection(
    model_selection=1, # 1适用十距离面部
    min_detection_confidence=0.5
)

def detect_face_mediapipe(image):
    """使用MediaPipe检测面部返回YOLO格式的边界框"""
    results = face_detection.process(cv2.cvtColor(image, cv2.COLOR_BGR2RGB))

    if not results.detections:
        return None

    # 获取第一个检测到的面部
    detection = results.detections[0]
    bbox = detection.location_data.relative_bounding_box

    # 转换为YOLO格式 (center_x, center_y, width, height)
    x_center = bbox.xmin + bbox.width / 2
    y_center = bbox.ymin + bbox.height / 2

    return [x_center, y_center, bbox.width, bbox.height]

def process_casme2_video(video_path, onset, offset, output_dir, emotion_class, base_name):
    """处理单个视频文件"""
    cap = cv2.VideoCapture(video_path)
    if not cap.isOpened():
        print(f"无法打开视频: {video_path}")
        return []

    samples = []
    frame_count = 0

    while cap.isOpened():
        ret, frame = cap.read()
        if not ret:
            break

        frame_count += 1

        # 只处理指定帧区间内的帧
        if onset <= frame_count <= offset:
            # 检测面部
            face_bbox = detect_face_mediapipe(frame)

            if face_bbox is None:
                continue # 如果没有检测到面部则跳过

            # 生成唯一文件名
            frame_name = f"{base_name}_{frame_count:04d}"

            # 保存图片
            img_path = os.path.join(output_dir, "images", "train", f"{frame_name}.jpg")
            cv2.imwrite(img_path, frame)

            # 保存标注文件
            label_path = os.path.join(output_dir, "labels", "train", f"{frame_name}.txt")

            with open(label_path, 'w') as f:
                f.write(f"{emotion_class} {' '.join(map(str, face_bbox))}\n")

            samples.append((img_path, label_path, emotion_class))

    cap.release()
    return samples

def convert_casme2_to_yolo(casme_root, output_dir, test_size=0.2):
    """主转换函数"""
    # 创建输出目录
    for folder in ["images/train", "images/val", "labels/train", "labels/val"]:
        os.makedirs(os.path.join(output_dir, folder), exist_ok=True)

    # 读取标注文件
    annotation_path = os.path.join(casme_root, "CASME2-coding-20250308.xlsx")
    df = pd.read_excel(annotation_path)

    # 微表情类别映射
    emotion_map = {
        'happiness': 0,
        'surprise': 1,
        'disgust': 2,
        'repression': 3,
        'others': 4
    }

    all_samples = []

    # 处理每个视频
    for idx, row in tqdm(df.iterrows(), total=len(df), desc="处理视频"):
        subject = row['Subject']
        video_name = f"{row['Filename']}.avi"
        onset = row['OnsetFrame']
        offset = row['OffsetFrame']
        emotion = row['Estimated Emotion'].lower()

        # 获取类别ID
        emotion_class = emotion_map.get(emotion, 4) # 默认为"others"

        # 视频路径
        video_path = os.path.join(casme_root, "RAW", f"sub{subject:02}", video_name)
        base_name = f"sub{subject:02}_{video_name.split('.')[0]}"

        # 处理视频
        samples = process_casme2_video(
            video_path, onset, offset, output_dir,
            emotion_class, base_name
        )

    return samples

cap.release()
return samples
```

模型训练

准备 YOLOv8 配置文件

创建 custom_yolov8.yaml:

```
CASME2_YOLO > ! casme2.yaml
path: /root/ultralytics/CASME2_YOLO
train: images/train
val: images/val

# 类别
names:
  0: happiness
  1: surprise
  2: disgust
  3: repression
  4: others
```

服务器环境准备

容器实例 实例连续关机15天会释放实例，实例释放会导致数据清空且不可恢复，释放前实例在数据在。

订阅GPU通知 设置密钥登录 小程序管理实例

租用新实例 批量续费 筛选标签 搜索实例名称/ID

实例ID / 名称	状态	规格详情	本地磁盘	健康状态	付费方式	释放时间/停机时间	SSH登录	快捷工具	操作
西北B区 / 481机 5994418ddb-1cc19ec7 设置名称	运行中	RTX 4090D * 1卡 查看详情	系统盘 34.98% 数据盘 0.00%	正常	按量计费 余额不足24小时	关机15天后释放 设置定时关机	登录指令 ssh***** 密码 *****	JupyterLab AutoPanel 实例监控 自定义服务	关机 更多
西北B区 / 143机 86b443a353-c1f30b9a 设置名称	已关机	RTX 3080 Ti * 1卡 查看详情	系统盘 35.02% 数据盘 0.00%	正常	按量计费	14天12小时05分后释放 设置定时关机			开机 更多

登录到服务器：

```
root@casme2:~# ssh root@5994418ddb-1cc19ec7
Warning: Permanently added '5994418ddb-1cc19ec7' (ssh-rsa) to the list of known hosts.
root@casme2:~#
root@casme2:~# cat /etc/os-release
NAME="Ubuntu"
VERSION="22.04 LTS (Noble Numbat)"
ID=ubuntu
ID_LIKE=debian
PRETTY_NAME="Ubuntu 22.04 LTS"
VERSION_ID="22.04"
HOME_URL="https://www.ubuntu.com/"
SUPPORT_URL="https://help.ubuntu.com/"
BUG_REPORT_URL="https://bugs.launchpad.net/ubuntu/"
PRIVACY_POLICY_URL="https://www.ubuntu.com/privacy-policy"
DISTRIBUTION_CODENAME=noble
root@casme2:~#
root@casme2:~# cat /etc/passwd
root:x:0:0:root:/root:/bin/bash
daemon:x:1:1:daemon:/usr/sbin:/usr/sbin/nologin
bin:x:2:2:bin:/bin:/usr/sbin/nologin
sys:x:3:3:sys:/dev:/usr/sbin/nologin
games:x:4:65534:games:/usr/games:/usr/sbin/nologin
root@casme2:~#
```

安装 python 依赖库

```
pip install opencv-python pandas numpy tqdm
```

上传训练程序

```
1 from ultralytics import YOLO
2
3 # 加载模型
4 model = YOLO('yolov8n.pt') # 或选择其他预训练模型
5
6 # 训练配置
7 results = model.train(
8     data='CASME2_YOLO/casme2.yaml',
9     imgsz=640,
10    epochs=200,
11    batch=8, # 较小的batch size适合小数据集
12    patience=30,
13    device='0',
14    project='microexpression',
15    name='yolov8n_face_me',
16    augment=True, # 启用内置数据增强
17    hsv_h=0.1, # 色相增强
18    hsv_s=0.1, # 饱和度增强
19    hsv_v=0.1, # 明度增强
20    degrees=10, # 旋转角度
21    translate=0.1, # 平移
22    scale=0.1, # 缩放
23    shear=5, # 剪切
24 )
```

启动 GPU 主机训练微表情识别模型,

```
root@autodl-container-86b443a353-clf30b9a:~/ultralytics/casme2/RAW/sub07# nvidia-smi
Fri May 9 17:03:19 2025

+-----+
| NVIDIA-SMI 535.104.05                Driver Version: 535.104.05   CUDA Version: 12.2   |
+-----+-----+
| GPU Name                               Persistence-M | Bus-Id        Disp.A | Volatile Uncorr. ECC |
| Fan  Temp  Perf    Pwr:Usage/Cap       | Memory-Usage | GPU-Util  Compute M. |
|-----+-----+
| 0  NVIDIA GeForce RTX 3080 Ti         On           | 00000000:0E:00:00 Off |          0%      N/A |
| 59%   63C   P2             213W / 350W      | 1842MiB / 12288MiB |      46%    Default  |
+-----+-----+

+-----+
| Processes:                              |
| GPU   GI   CI        PID   Type   Process name                              | GPU Memory |
| ID    ID  ID           |              | Usage      |
+-----+-----+

root@autodl-container-86b443a353-clf30b9a:~/ultralytics/casme2/RAW/sub07#

connect.westb.seetacloud.com - PuTTY

9      -1  1  144400 ultralytics.nn.modules.block.BFPF [164, 164, 4]
10     -1  1  0 torch.nn.modules.conv.conv2d [None, 2, 'nearest']
11     -1  1  1 ultralytics.nn.modules.conv.Conv2d [1]
12     -1  1  144412 ultralytics.nn.modules.block.CIF [164, 128, 1]
13     -1  1  0 torch.nn.modules.conv.conv2d [None, 2, 'nearest']
14     -1  1  27244 ultralytics.nn.modules.block.CIF [164, 64, 1]
15     -1  1  28752 ultralytics.nn.modules.conv.conv2d [164, 64, 2, 2]
16     -1  1  144416 ultralytics.nn.modules.conv.Conv2d [1]
17     -1  1  144418 ultralytics.nn.modules.conv.Conv2d [128, 128, 1]
18     -1  1  147712 ultralytics.nn.modules.conv.Conv2d [128, 128, 2, 2]
19     -1  1  147714 ultralytics.nn.modules.conv.Conv2d [1]
20     -1  1  493016 ultralytics.nn.modules.conv.Conv2d [164, 164, 1]
21     -1  1  782287 ultralytics.nn.modules.head.Head [5, 164, 128, 284]
22     [15, 15, 25] 1 782287 ultralytics.nn.modules.head.Head

Model Summary: 225 layers, 3011262 parameters, 3011887 gradients, 4.2 GFLOPs

Transferred 318/355 items from pretrained weights
TensorBoard: Start with 'tensorboard --logdir microexpression/yolov8n_face_me', view at http://localhost:6006/
Freezing layer 'head.12.conv.weight'
INFO: running Automatic Mixed Precision (AMP) checks with YOLOv8n...
AMP checks passed
INFO: Scanning /root/.ultralytics/CASME2_YOLO/labels/train... 13619 images, 0 backgrounds, 0 corrupt: 100% [13619/13619 [00:15:00:00, 897.84it/s]
INFO: New cache created: /root/.ultralytics/CASME2_YOLO/labels/train.cache
INFO: Scanning /root/.ultralytics/CASME2_YOLO/labels/val... 4537 images, 0 backgrounds, 0 corrupt: 100% [4537/4537 [00:14:00:00, 520.10it/s]
INFO: New cache created: /root/.ultralytics/CASME2_YOLO/labels/val.cache
INFO: Finishing labels to microexpression/yolov8n_face_me/labels-17p...
optimizer: 'optimizer=adamw' found, ignoring 'lr=0.01' and 'momentum=0.937' and determining best 'optimizer', 'lr' and 'momentum' automatically...
optimizer: SGD(lr=0.01, momentum=0.9) with parameter groups 57 weight(decay=0.0), 44 weight(decay=0.0001), 43 bias(decay=0.0)
200 epochs...

Epoch: 1/200
GPU_mem box_loss cls_loss dfl_loss Instances Size
1.56 0.728 1.862 1.18 9 440/100% [1703/1703 [01:15:00:00, 15.28it/s]
Class Images Instances BoxAP mAP50 mAP50-95 100% 284/284 [00:17:00:00, 14.38it/s]
all 4537 4537 0.434 0.761 0.713 0.453

Epoch: 2/200
GPU_mem box_loss cls_loss dfl_loss Instances Size
1.562 0.5972 1.204 1.01 7 440/100% [1703/1703 [01:17:00:00, 17.46it/s]
Class Images Instances BoxAP mAP50 mAP50-95 100% 284/284 [00:14:00:00, 19.17it/s]
all 4537 4537 0.508 0.713 0.724 0.434

Epoch: 3/200
GPU_mem box_loss cls_loss dfl_loss Instances Size
1.556 0.4115 0.9661 1.02 9 440/100% [1703/1703 [01:18:00:00, 18.28it/s]
Class Images Instances BoxAP mAP50 mAP50-95 100% 284/284 [00:14:00:00, 19.55it/s]
all 4537 4537 0.515 0.824 0.921 0.553

Epoch: 4/200
GPU_mem box_loss cls_loss dfl_loss Instances Size
1.516 0.3036 0.7475 1.03 9 440/100% [1703/1703 [01:19:00:00, 19.36it/s]
Class Images Instances BoxAP mAP50 mAP50-95 100% 284/284 [00:14:00:00, 20.07it/s]
all 4537 4537 0.704 0.808 0.894

Epoch: 5/200
GPU_mem box_loss cls_loss dfl_loss Instances Size
1.430 0.1441 0.7437 1.04 13 440/100% [1445/1703 [01:19:00:13, 19.31it/s]
```

connect.westb.seetacloud.com - PuTTY

```
autodl-container-86b443a353-clf30b9a Fri May 9 17:07:34 2025 535.104.05
[0] NVIDIA GeForce RTX 3080 Ti | 64°C, 39% | 1842 / 12288 MB |
```

```
root@autodl-container-86b443a353-clf30b9a:~/ultralalytics/casme2/RAW/sub07# nvidia-smi
Fri May 9 17:03:19 2025

+-----+
| NVIDIA-SMI 535.104.05                Driver Version: 535.104.05   CUDA Version: 12.2   |
+-----+-----+
| GPU Name                               Persistence-M | Bus-Id        Disp.A | Volatile Uncorr. ECC |
| Fan  Temp  Perf              Pwr:Usage/Cap |      Memory-Usage | GPU-Util  Compute M. |
|                                           | MIG M.         |                       |
+-----+-----+
| 0  NVIDIA GeForce RTX 3080 Ti    On      | 00000000:5E:00:00 Off |          N/A         |
| 59%   63C   P2              213W / 350W | 1842MiB / 12288MiB |    46%    Default   |
+-----+-----+

Processes:
+-----+
| GPU  GI  CI       PID  Type  Process name                        GPU Memory |
| ID   ID  ID                                   Usage   |
+-----+
root@autodl-container-86b443a353-clf30b9a:~/ultralalytics/casme2/RAW/sub07#
```

```
Epoch GPU_mem box_loss cls_loss dfl_loss instances Size
71/200 1.51G 0.2612 0.2368 0.8903 10 640: 100% | 1703/1703 [01:35<00:00, 17.85it/s]
Class Images Instances Box(P R mAP50 mAP50-95): 100% | 284/284 [00:14<00:00, 20.14it/s]
all 4537 4537 1 1 0.995 0.992

Epoch GPU_mem box_loss cls_loss dfl_loss instances Size
72/200 1.51G 0.2641 0.2359 0.8913 6 640: 100% | 1703/1703 [01:35<00:00, 17.81it/s]
Class Images Instances Box(P R mAP50 mAP50-95): 100% | 284/284 [00:14<00:00, 20.00it/s]
all 4537 4537 1 1 0.995 0.992

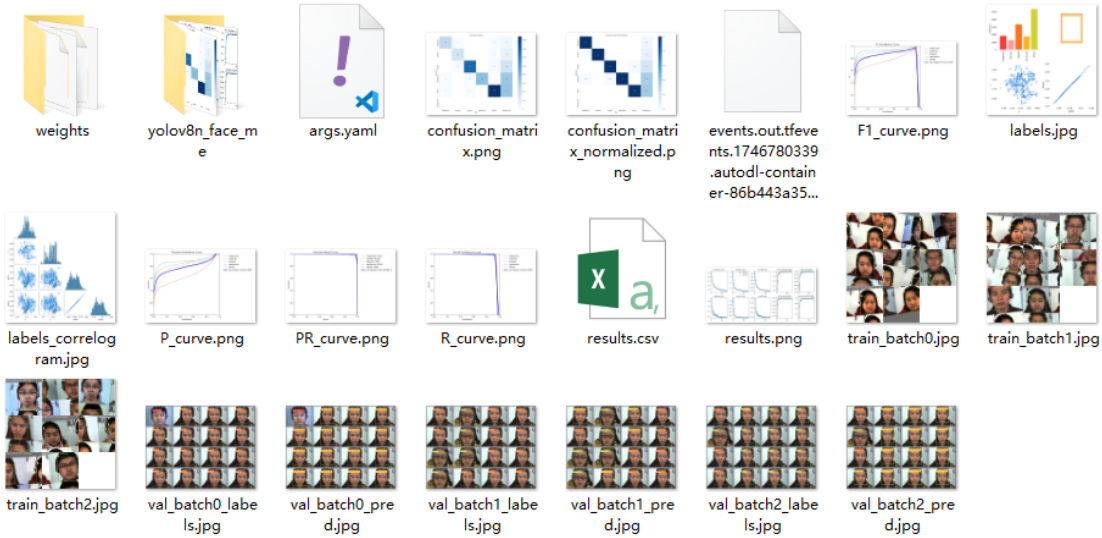
Stopping training early as no improvement observed in last 30 epochs. Best results observed at epoch 42, best model saved as best.pt.
To update EarlyStopping(patience=30) pass a new patience value, i.e. 'patience=300' or use 'patience=0' to disable EarlyStopping.

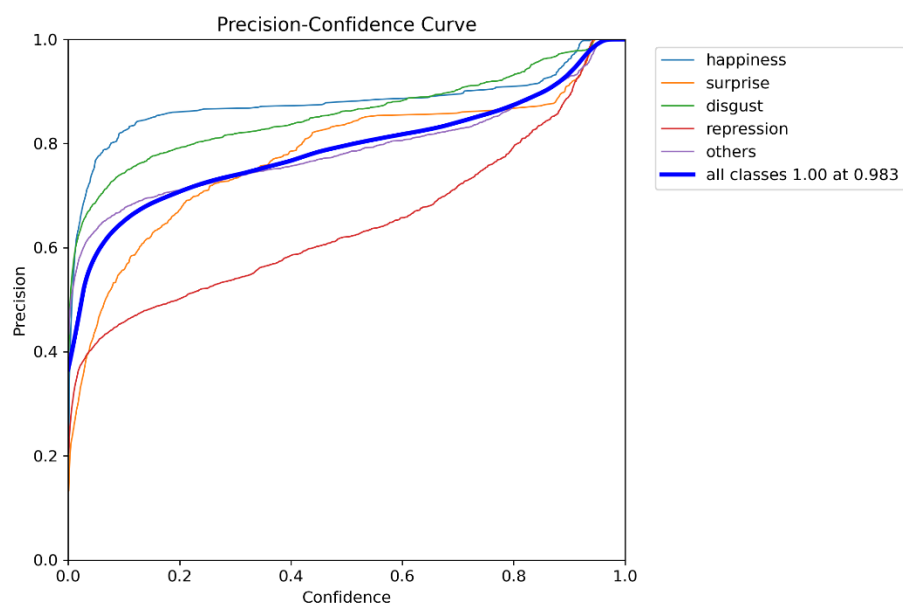
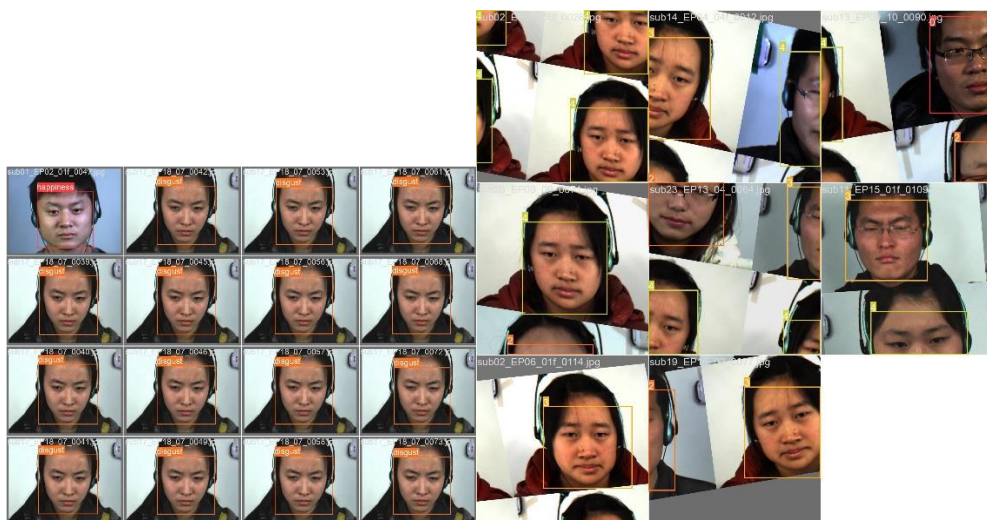
72 epochs completed in 2.188 hours.
Optimizer stripped from microexpression/yolov8n_face_me/weights/last.pt, 6.2MB
Optimizer stripped from microexpression/yolov8n_face_me/weights/best.pt, 6.2MB

Validating microexpression/yolov8n_face_me/weights/best.pt...
ultralalytics YOLOv8.1.2 Python-3.8.10 torch-2.0.0+cu118 CUDA:0 (NVIDIA GeForce RTX 3080 Ti, 12045MiB)
Model summary (fused): 168 layers, 3006623 parameters, 0 gradients, 8.1 GFLOPs
Class Images Instances Box(P R mAP50 mAP50-95): 100% | 284/284 [00:20<00:00, 13.91it/s]
all 4537 4537 0.989 0.992 0.992 0.954
happiness 4537 454 0.958 1 0.995 0.962
surprise 4537 321 0.911 1 0.995 0.963
disgust 4537 1339 0.975 0.979 0.993 0.984
repression 4537 715 0.887 0.994 0.994 0.927
others 4537 1708 0.926 0.972 0.985 0.936

Speed: 0.1ms preprocess, 1.8ms inference, 0.0ms loss, 0.5ms postprocess per image
Results saved to microexpression/yolov8n_face_me
root@autodl-container-86b443a353-clf30b9a:~/ultralalytics#
```

训练结果汇总





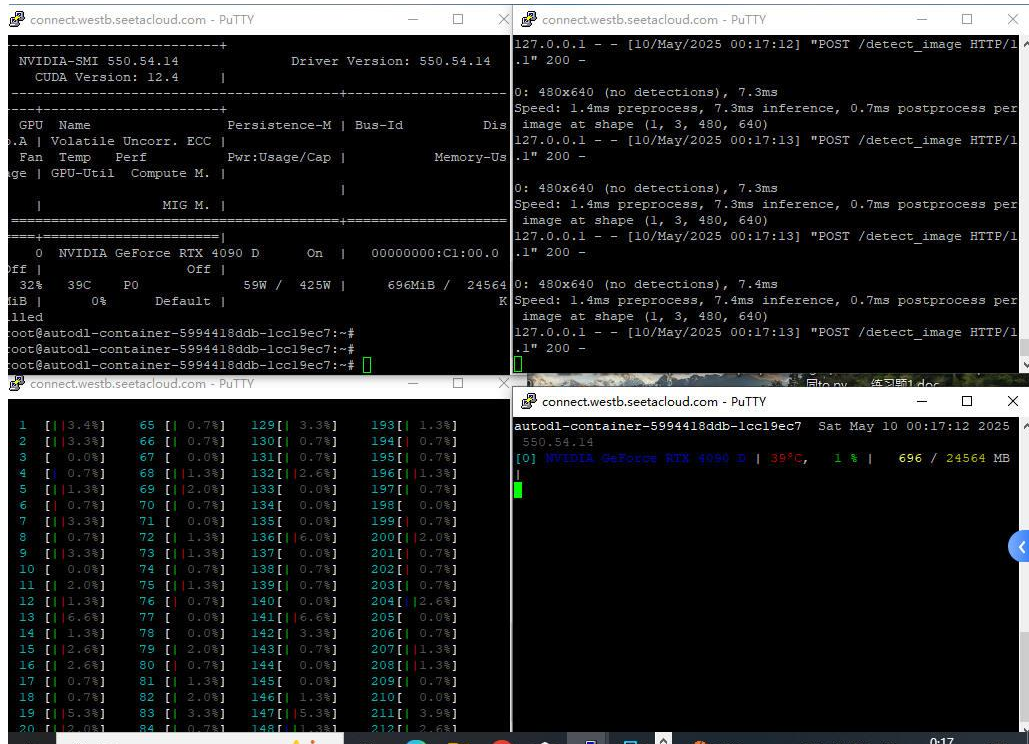
部署应用

服务器端调用训练好的模型 “yolov8_microexpressing.pt”, 接收客户端摄像头数据并检测:

程序:

```
1 # yolo_server.py
2 from flask import Flask, request, Response, jsonify
3 import cv2
4 import numpy as np
5 from ultralytics import YOLO
6 import time
7 import threading
8
9 app = Flask(__name__)
10
11 # 加载YOLOv8模型 (可以根据需要更换模型大小)
12 model = YOLO('yolov8n_microexpressing.pt') # 会自动下载模型如果不存在
13
14 # 用于统计性能
15 request_count = 0
16 start_time = time.time()
17 lock = threading.Lock()
18
19 @app.route('/detect', methods=['POST'])
20 def detect():
21     global request_count
22
23     # 接收图像数据
24     if 'image' not in request.files:
25         return jsonify({'error': 'No image provided'}), 400
26
27     try:
28         # 读取并解码图像
29         img_bytes = request.files['image'].read()
30         img = cv2.imdecode(np.frombuffer(img_bytes, np.uint8), cv2.IMREAD_COLOR)
31
32         if img is None:
33             return jsonify({'error': 'Failed to decode image'}), 400
34
35         # 运行YOLOv8检测
36         results = model(img)
37
38         # 获取检测结果
39         boxes = results[0].boxes.xyxy.tolist() # 边界框坐标
40         classes = results[0].boxes.cls.tolist() # 类别ID
41         confidences = results[0].boxes.conf.tolist() # 置信度
42
43         # 获取类别名称
44         class_names = results[0].names
45         detected_objects = []
46         for box, cls, conf in zip(boxes, classes, confidences):
47             detected_objects.append({
48                 'class': class_names[int(cls)],
49                 'confidence': float(conf),
50                 'bbox': [float(x) for x in box]
51             })
52
53         # 更新请求计数
54         with lock:
55             request_count += 1
56
57         # 返回JSON结果
58         return jsonify({
59             'objects': detected_objects,
60             'image_size': {'width': img.shape[1], 'height': img.shape[0]}
61         })
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91 @app.route('/detect_image', methods=['POST'])
92 def detect_image():
93     """返回带标注的图像而不是JSON"""
94     if 'image' not in request.files:
95         return jsonify({'error': 'No image provided'}), 400
96
97     try:
98         img_bytes = request.files['image'].read()
99         img = cv2.imdecode(np.frombuffer(img_bytes, np.uint8), cv2.IMREAD_COLOR)
100
101         if img is None:
102             return jsonify({'error': 'Failed to decode image'}), 400
103
104         # 运行检测并获取带标注的图像
105         results = model(img)
106         annotated_img = results[0].plot() # 自动绘制检测结果
107
108         # 编码图像为JPEG
109         _, img_encoded = cv2.imencode('.jpg', annotated_img)
110
111         return Response(img_encoded.tobytes(), mimetype='image/jpeg')
112
113     except Exception as e:
114         return jsonify({'error': str(e)}), 500
115
116
117 @app.route('/stats')
118 def stats():
119     """获取服务器统计信息"""
120     with lock:
121         elapsed = time.time() - start_time
122         return jsonify({
123             'request_count': request_count,
124             'uptime': elapsed,
125             'requests_per_second': request_count / elapsed if elapsed > 0 else 0
126         })
127
128
129 if __name__ == '__main__':
130     # 在AutoDL上运行时, 使用0.0.0.0监听所有网络接口
131     # 端口号可以根据需要修改, 但要确保AutoDL实例的安全组允许该端口
132     app.run(host='0.0.0.0', port=5000, threaded=True)
```

服务器部署好, 运行并做好接收数据分析的准备:



客户端调用摄像头上传视频流到服务器程序：

```
if not ret:
    break

# 编码图像为JPEG
_, img_encoded = cv2.imencode('.jpg', frame)

# 发送到服务器
start_time = time.time()
response = requests.post(SERVER_URL,
                        files={'image': img_encoded.tobytes()},
                        timeout=5)

# 处理响应
if response.status_code == 200:
    if SERVER_URL.endswith('/detect_image'):
        # 显示带标注的图像
        img = cv2.imdecode(np.frombuffer(response.content, np.uint8), cv2.IMREAD_COLOR)
        cv2.imshow('YOLOv8 Detection', img)
    else:
        # 处理JSON结果
        result = response.json()
        print("检测到对象:", result['objects'])
else:
```

完整客户端代码：

```
# client.py
import cv2
import requests
import time
import numpy as np

SERVER_URL = "http://127.0.0.1:5000/detect_image" # 或/detect获取JSON结果

cap = cv2.VideoCapture(0) # 本地摄像头

while True:
    ret, frame = cap.read()
    if not ret:
        break

    # 编码图像为JPEG
    _, img_encoded = cv2.imencode('.jpg', frame)

    # 发送到服务器
    start_time = time.time()
    response = requests.post(SERVER_URL,
                            files={'image': img_encoded.tobytes()},
                            timeout=5)

    # 处理响应
    if response.status_code == 200:
        if SERVER_URL.endswith('/detect_image'):
            # 显示带标注的图像
            img = cv2.imdecode(np.frombuffer(response.content, np.uint8), cv2.IMREAD_COLOR)
            cv2.imshow('YOLOv8 Detection', img)
        else:
            # 处理JSON结果
            result = response.json()
            print("检测到对象:", result['objects'])
    else:
        print("请求失败:", response.text)

    # 显示FPS
    fps = 1 / (time.time() - start_time)
    print(f"FPS: {fps:.2f}")

    if cv2.waitKey(1) & 0xFF == ord('q'):
        break

cap.release()
cv2.destroyAllWindows()
```

Ssh 加密隧道映射端口：

AutoDL

AutoDL SSH隧道工具

ssh指令:

ssh -p 27794 root@connect.westb.seetacloud.com

ssh密码:

代理到本地端口:

5000

将远程端口代理到本地，多个端口使用逗号分隔

代理到远程端口:

10801

将本地端口代理到远程，多个端口使用逗号分隔

停止代理

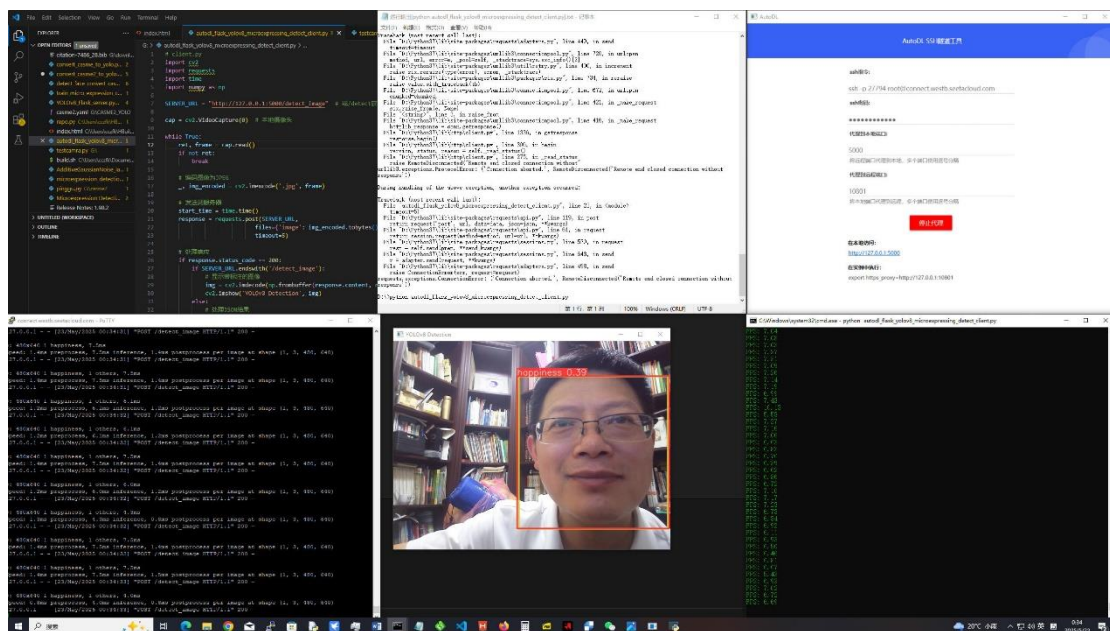
在本地访问:

<https://127.0.0.1:5000>

在实例中执行:

export https_proxy=http://127.0.0.1:10801

返回带分类标注图像



The screenshot displays a Windows desktop with a terminal window open. The terminal shows the execution of a Python script named 'face_detector.py'. The script is located at 'C:\Users\user\Desktop\face_detector.py'. It uses OpenCV to capture video from a camera (video0) and processes each frame to detect and classify faces. The output shows detection and classification results for each frame, including bounding boxes, class names (e.g., 'surprise'), and confidence scores. The terminal also shows the script's execution path and the location of the pre-trained model files.

```

C:\Users\user\Desktop> python face_detector.py
FPS: 10.32
检测到对象: [{"bbox": [208, 73321533203]25, 76.391357421875, 508.666748046875,
352.70629828125], "class": "surprise", "confidence": 0.49345060605010986}]
FPS: 10.31
检测到对象: [{"bbox": [202.4830322265625, 68.82373046875, 509.831787109375, 3
57.5390625], "class": "surprise", "confidence": 0.3848098516424334}]
FPS: 10.27
检测到对象: [{"bbox": [202.18194580078125, 66.74252319335938, 511.1171264684
84, 360.6621301269531], "class": "surprise", "confidence": 0.2791531383399124
15}]
FPS: 10.26
检测到对象: [204.069091796875, 51.6673583984375, 514.3492431640625,
372.07147216796875], "class": "surprise", "confidence": 0.37197190523147383}]
FPS: 10.76
检测到对象: [{"bbox": [190.8660888671875, 3.82568353375, 513.25610315625, 36
8.8083041972656], "class": "surprise", "confidence": 0.5156636503677368}]
FPS: 10.53
检测到对象: [{"bbox": [200.04315185546875, 41.43377685546875, 512.03961181640
62, 368.938996484375], "class": "surprise", "confidence": 0.480563679337088}]
FPS: 10.64
检测到对象: [{"bbox": [201.69586181640625, 39.6463623046875, 510.4765625, 365
13716513671875], "class": "surprise", "confidence": 0.5426963657733765}]
FPS: 10.42

```