Python 机器学习环境部署报告

一. 任务完成摘要

1. 目标

搭建基于 Python 3.9 的机器学习环境，并完成图像处理及其在 Flask 框架中的应用。主要内容包括安装 Python 3.9 并验证环境、安装 `numpy` 和 `matplotlib` 软件包并绘制正弦曲线、借助 OpenCV 进行人脸检测、并利用 Flask 实现实时视频流展示。同时在 Gitee 平台安装相关软件包并测试其功能。

2. 内容

1. **安装 Python 3.9** 并验证版本，输出 Hello World，编写斐波那契数列程序。
2. **使用 pip 安装 numpy 和 matplotlib**，绘制正弦曲线。
3. **安装 OpenCV 和 Flask**，实现摄像头人脸检测并实时显示。
4. **使用 GaussianBlur** 进行人脸区域高斯模糊处理，支持 MP4 格式导入。
5. **安装 Gitee 上的软件包**，并使用 `ImageTest` 进行图像处理测试。

3. 完成情况

任务基本完成，额外实现了路由扩展和算法优化，解决了 CPU 负载问题。

4. 值得分享的思考或发现

1. `yield` 语法在视频帧处理中的应用，使代码更加高效。
2. Flask 框架的应用实现了 Python 对网页控制能力。
3. CPU 负载的优化：通过图像处理函数的替换提升了流畅度。

二. 任务目标

本次任务的目标是搭建一个 Python 3.9 的开发环境，使用 OpenCV 进行图像处理，并通过 Flask 实现实时视频流展示。

三. 主要内容

1. **Python 3.9 安装**：验证 Python 3.9 环境，编写简单的输出及斐波那契数列程序。

```
    print('Hello World!')

    a = 1
    b = 1

    def printFibo(n):
        global a, b
        if n == 1:
            print(a)
```

```
            return
        elif n == 2:
            print(a)
            print(b)
            return
        elif a == 1 & b == 1:
            print(a)
            print(b)
        count = 0
        c = a + b
        print(c)
        a = b
        b = c
        if not n == 0:
            printFibo(n-1)
        else:
            return

    printFibo(10)
```

2. **安装 numpy 和 matplotlib**：进行基础数据分析和绘图。

```
    import matplotlib.pyplot as plt
    import numpy as np

    x = np.linspace(0, 2 * np.pi, 1000)
    y = np.sin(x)

    plt.plot(x, y)
    plt.title('sin wave')
    plt.xlabel('x')
    plt.ylabel('y')

    plt.show()
```

3. **OpenCV 人脸检测**：通过摄像头获取图像，使用 Haar 级联分类器进行人脸检测。
4. **Flask 实时显示**：利用 Flask 框架将处理后的视频流在浏览器中实时展示。
5. **扩展功能**：支持 MP4 格式的导入，并对人脸区域进行高斯模糊处理。

```
    import cv2
    from flask import Flask, render_template, Response, request

    app = Flask(__name__)

    # 加载人脸检测模型
    face_cascade = cv2.CascadeClassifier(cv2.data.haarcascades +
'haarcascade_frontalface_default.xml')

    def generate_frames(source):
```

```
        if source == 'camera':
            cap = cv2.VideoCapture(0)
        else:
            cap = cv2.VideoCapture(source)

        while True:
            # 捕获帧
            success, frame = cap.read()
            if not success:
                break

            # 转为灰度图像
            gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)

            # 检测人脸
            faces = face_cascade.detectMultiScale(gray, scaleFactor=1.  1,
minNeighbors=5, minSize=(30, 30))

            # 在检测到的人脸上应用模糊处理
            for (x, y, w, h) in faces:
                face_region = frame[y:y+h, x:x+w]
                face_region = cv2.GaussianBlur(face_region, (99, 99),   30)
                frame[y:y+h, x:x+w] = face_region

            # 编码帧为 JPEG 格式
            ret, buffer = cv2.imencode('.jpg', frame)
            frame = buffer.tobytes()

            # 生成视频流
            yield (b'--frame\r\n'
                   b'Content-Type: image/jpeg\r\n\r\n' + frame +    b'\r\n')

    @app.route('/')
    def index():
        return render_template('index.html')

    @app.route('/video_feed')
    def video_feed():
        source = request.args.get('source', default='camera')
        return Response(generate_frames(source), mimetype='multipart/   x-mixed-
replace; boundary=frame')

    if __name__ == '__main__':
        app.run(debug=True)
```

以下为HTML文件内容以及网页效果：

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Video Stream</title>
```

```
</head>
<body>
    <h1>人脸识别及高斯模糊视频流</h1>
    <form action="/video_feed" method="get">
        <label for="source">选择视频来源: </label>
        <select name="source" id="source">
            <option value="camera">Camera</option>
            <option
value="C:/Users/32438/Videos/Captures/cxk.mp4">MP4_file</option>
        </select>
        <button type="submit">Start</button>
</body>
</html>
```

人脸识别及高斯模糊视频流

选择视频来源： Camera ▼ Start

1. **ImageTest测试**：导入老师在Gitee上给出的软件包以及代码进行相关图像处理的测试（将高斯双边滤波函数替换为了高斯模糊函数，从而降低CPU负载）

```
    #!/usr/bin/env python3
    """
    Inferemote: a Remote Inference Toolkit for Ascend 310

    """
    ''' Prepare a test '''
    import cv2 as cv

    # 颜色反转/高斯模糊
    def bilateral_filter(image):
        image = cv.bitwise_not(image)
        image = cv.GaussianBlur(image, (99, 99), 30)
        return image

    from inferemote.testing import ImageTest
    class MyTest(ImageTest):
        ''' Define a callback function for inferencing, which will be   called for
every single image '''
        def run(self, image):
            ''' an image must be returned in the same shape '''
            new_img = bilateral_filter(image)
            return new_img

    if __name__ == '__main__':

        t = MyTest(mode='liveweb', threads=1)
        t.start(input='C:/Users/32438/Videos/Captures/cxk.mp4',     mode='show')
```

```
    #Ends
```

四. 主要思路及关键步骤

1. **生成器与实时流的结合**：使用 Python 的 `yield` 生成器函数，使摄像头获取的每一帧图像能被连续处理并返回给 Flask 路由。这种实现方式不仅节省内存，还保证了流的持续性，避免中断。
2. **Flask 路由的设计**：通过 Flask 框架设置不同的路由，既能处理摄像头流，又可以支持上传的 MP4 文件。这一设计提高了系统的灵活性和实用性，能应对多种格式的输入源。
3. **高斯模糊的应用**：在检测到人脸后，对面部区域进行高斯模糊处理。在实际实现中，使用了 OpenCV 的 `GaussianBlur` 函数，在保证隐私的同时，仍能保持其他图像部分的清晰度。
4. **图像处理性能优化**：在初始使用高斯双边滤波处理时，CPU 负载较高，影响了视频帧率。后期通过更换滤波算法，显著提升了视频流的处理效率，使显示更加流畅。

五. 完成情况与结果分析

1. **任务完成情况**

任务中的 Python 3.9 环境部署、基础库安装和图像处理等步骤均已顺利完成，最终实现了摄像头视频流的人脸检测和实时展示功能，并通过 Flask 路由实现了 MP4 和图片格式的导入。同时对检测到的人脸区域应用了高斯模糊处理。

2. **关键结果分析**

- **数据可视化**：通过 `numpy` 和 `matplotlib` 绘制了正弦曲线，验证了 Python 机器学习环境的正确配置。
- **人脸检测与模糊处理**：OpenCV 的人脸检测功能工作正常，检测准确率较高。通过高斯模糊处理，能够有效保护隐私，且整体效果流畅。
- **性能优化**：替换高斯双边滤波后，CPU 占用显著降低，帧率提升明显，确保了实时视频流的连续性和流畅性。

六. 总结

在本次任务中，不仅成功搭建了 Python 3.9 的机器学习环境，还通过 OpenCV 和 Flask 的结合，初步实现了简单的图像处理功能和实时展示。通过任务的完成，我进一步加深了对 Python 的生成器函数、图像处理算法以及 Flask 路由的理解。