华中科技大学

Python 编程技术 小组项目设计报告

第七组：《图像人脸识别与计数系统》

人员信息

指导教师	陈建文
组长	王艺澄
组员	武晨曦
	胡玮承
	张俊哲
	胡波
	刘柏煜
	钟轶群
	郑帅
	龙文振
	易俊哲

组长联系电话：18813096937

目录

(一) 项目背景与目标.....1

一、 项目背景：.....1
二、 项目目标：.....1

(二) 团队成员与分工.....3

(三) 技术选型与实现.....4

(四) 系统设计与实现.....5

一、 系统架构设计.....5
二、 关键技术与实现.....5
(1) Tkinter 模块.....5
(2) TkinterDnD2 模块.....5
(3) messagebox 模块.....6
(4) OpenCV（cv2）模块.....6
(5) PIL（Pillow）模块.....7
(6) Ultralytics YOLO 模块.....7
(7) 主函数和事件处理.....7
(8) 拖放功能实现.....8
(9) YOLO 模型简介：.....8

(五) 代码分析与结果.....9

一、 代码结构概述.....9
(1) 导入必要的库.....9
(2) 定义 YOLO 模型.....9
(3) 编写人脸检测函数.....9
(4) 创建 GUI 窗口.....11
(5) 显示图像和结果.....12
(6) 设置拖放功能.....12
(7) 处理拖放事件.....12
(8) 运行应用程序.....13

(六) 场景应用：.....14

(七) 心得体会.....15

(一) 项目背景与目标

一、 项目背景：

随着人工智能与计算机视觉技术的飞速发展，图像处理和模式识别技术在各个领域得到了广泛应用。特别是在安全监控、人机交互、社交媒体分析等领域，对图像中人脸的检测与计数需求日益增长。传统的人脸检测方法往往需要大量的人工干预，效率低下且易出错。因此，开发一种高效、准确、自动化的人脸检测与计数系统显得尤为重要。

近年来，Python 编程语言凭借其简洁的语法、丰富的库资源以及强大的社区支持，在数据科学、机器学习和计算机视觉等领域占据了主导地位。特别是 OpenCV、dlib、face_recognition 等库，为人脸检测提供了强大的工具。利用这些库，可以快速地构建出高效的人脸检测与计数系统，满足实际应用需求。

再考虑到最近学校也正利用线上方式查看各课堂到课率、抬头率等情况，我们小组决定以“图像主体计数系统”为题，利用 python 开发实现图片人脸的检测和计数。

二、 项目目标：

本项目旨在利用 Python 编程语言和相关计算机视觉库，开发一个能够自动检测和计数图像中人脸数量的系统。具体目标如下:

1. **实现自动化人脸检测**：通过集成先进的计算机视觉算法，系统能够自动识别并定位图像中的人脸位置，无需人工干预。

2. **准确计数人脸数量**：在检测到人脸的基础上，系统能够准确统计出图像中的人脸数量，为后续的分析和处理提供可靠的数据支持。

3. **优化性能与准确性**：通过算法优化和参数调整，提高系统的检测速度和准确性，确保在不同场景和光照条件下都能获得稳定的表现。

4. **提供用户友好的界面**：设计简洁直观的用户界面，使得用户能够轻松上传图像并查看检测结果，降低使用门槛。

5. **扩展性与可维护性**：系统设计应具备良好的扩展性和可维护性，以便未来能够方便地添加新的功能或进行算法升级。

通过实现上述目标，本项目将为安全监控、人机交互、社交媒体分析等领域提供一种高效、准确的人脸检测与计数解决方案，推动相关技术的进一步发展与应用。

(二) 团队成员与分工

- **组长：王艺澄**

负责前端代码编写，包括图像读取、显示和 GUI 等；负责项目整体规划与进度监督，协调团队成员的工作，确保项目按时完成

- **组员**：

易俊哲：给项目提出建议，并参与汇报答辩

武晨曦：负责后端代码编写，包括置信度设置、Yolo 模型等

胡玮承：进行 PPT 所需材料的收集工作

张俊哲：进行 PPT 的制作

龙文振：给项目提出建议，并参与汇报答辩

胡波：负责收集材料并编写项目报告

郑帅：协助组长进行项目管理，并进行部分代码的编写

刘柏煜：负责收集材料并编写项目报告

钟轶群：协助组长进行项目管理，并进行部分代码的编写

(三) 技术选型与实现

1. **Python 语言**：作为项目的主要编程语言，Python 具有简单易学、库丰富等优点，非常适合用于快速开发。

2. **Tkinter 库**：用于创建 GUI 界面。Tkinter 是 Python 的标准 GUI 库，提供了丰富的控件和布局管理功能，能够满足我们的需求。

3. **OpenCV 库**：用于图像处理。OpenCV 是一个开源的计算机视觉库，提供了丰富的图像处理函数和算法，包括图像的读取、转换、显示等。

4. **Pillow 库**：即 PIL（Python Imaging Library）的分支，用于处理图像，特别是将 OpenCV 处理的图像转换为 Tkinter 可以显示的格式。

5. **TkinterDnD 库**：实现了拖放功能，使用户可以通过拖放图片到 GUI 中进行人脸检测。

6. **Ultralytics YOLO**：一个流行的目标检测库，提供了 YOLO 算法的预训练模型和预测函数，简化了人脸检测的实现。

(四) 系统设计与实现

一、 系统架构设计

系统采用模块化设计，分为以下几个模块：

1. GUI 模块：负责用户界面的创建和事件处理。
2. 图像处理模块：负责图像的读取、处理和显示。
3. 人脸检测模块：利用 YOLO 算法进行人脸检测，并返回检测结果。
4. 结果显示模块：在 GUI 中显示检测到的人脸数量和检测后的图像。

二、 关键技术与实现

（1） Tkinter 模块

Tkinter 是 Python 的标准 GUI（图形用户界面）库，用于创建和管理图形界面。它提供了丰富的控件（如按钮、文本框、标签等）和布局管理工具，使得开发者能够轻松地创建出直观的用户界面。

在代码中，Tkinter 用于创建主窗口、标签（用于显示检测结果和图片）以及管理窗口的布局。例如：

- root = TkinterDnD.Tk(): 创建一个 Tkinter 主窗口，并启用拖放功能（通过 TkinterDnD2 扩展）。
- panel = tk.Label(root): 创建一个标签控件，用于显示处理后的图片。
- result_label = tk.Label(root, text="请拖入图片", font=("Arial", 16)): 创建一个标签控件，用于显示检测结果（如人脸数量）。

（2） TkinterDnD2 模块

TkinterDnD2 是一个 Tkinter 的扩展，用于实现拖放功能。它允许用户将文

件从文件管理器或其他应用程序拖放到 Tkinter 应用程序的窗口中。

在代码中，TkinterDnD2 用于处理拖放事件，并获取拖放的图片文件路径。例如：

- root.drop_target_register(DND_FILES): 注册拖放目标，允许拖放文件。
- root.dnd_bind('<<Drop>>', drop): 绑定拖放事件到 drop 函数，当文件被拖放到窗口中时，将调用该函数。

（3） messagebox 模块

messagebox 是 Tkinter 中的一个子模块，用于显示消息框。它提供了多种类型的消息框，如错误框、信息框、警告框等，用于向用户显示信息或获取用户输入。

在代码中，messagebox 用于在发生错误时向用户显示错误信息。例如：

- messagebox.showerror("错误", "无法读取图片"): 当无法读取图片时，显示一个错误消息框。

（4） OpenCV（cv2）模块

OpenCV 是一个开源的计算机视觉和机器学习软件库，提供了大量的图像处理和分析功能。它广泛用于图像和视频处理、人脸识别、物体检测等领域。

在代码中，OpenCV 用于读取图片文件，并将其转换为 YOLO 模型可以处理的格式。例如：

- img = cv2.imread(image_path): 读取图片文件。
- result_img = results[0].plot(): 使用 YOLO 模型处理后的图像，通过 plot 方法转换为可用于显示的图像。

（5） PIL（Pillow）模块

PIL（Python Imaging Library）是一个强大的图像处理库，但由于其不再维护，现在通常使用其分支 Pillow。Pillow 提供了广泛的图像文件格式支持、高效的内部表示和相当强大的图像处理功能。

在代码中，Pillow 用于将 OpenCV 处理的图像（BGR 格式）转换为 Tkinter 可以显示的图像（RGB 格式），并调整图像大小以适应 GUI 窗口。例如：

- img=Image.fromarray(cv2.cvtColor(output_img, cv2.COLOR_BGR2RGB))：将 BGR 格式的图像转换为 RGB 格式。
- img.thumbnail((400, 400))：调整图像大小，使其不超过 400x400 像素。

（6） Ultralytics YOLO 模块

Ultralytics YOLO 是一个基于 YOLO（You Only Look Once）算法的开源目标检测库。它提供了预训练的模型，可以轻松地用于各种目标检测任务，包括人脸检测。

在代码中，Ultralytics YOLO 用于加载预训练的人脸检测模型，并对图片进行人脸检测。例如：

- model = YOLO('yolov8n-face.pt', verbose=False)：加载预训练的人脸检测模型。
- results = model.predict(img)：使用模型对图片进行预测，获取检测结果。

（7） 主函数和事件处理

- detect_faces(image_path)：这是一个自定义函数，用于读取图片、进行人脸检测，并返回检测到的人脸数量和带有检测框的图像。
- drop(event)：这是一个事件处理函数，当文件被拖放到窗口中时调用。

它获取拖放的图片路径，调用 detect_faces 函数进行人脸检测，并在 GUI 界面上显示结果。

（8） 拖放功能实现

利用 TkinterDnD 库实现拖放功能。首先，注册拖放目标，并绑定拖放事件处理函数。在事件处理函数中，读取拖放的图片文件，并进行人脸检测。最后，更新 GUI 界面，显示检测结果。

（9） YOLO 模型简介：

1. YOLO（You Only Look Once）是一种基于深度学习的目标检测算法，其核心理念是“只看一次”图像，即可同时预测多个目标的位置和类别。

2. YOLO 将目标检测任务转化为一个单一的回归问题，从而大大简化了检测流程，提高了检测速度。yolov8n-face.pt 是 YOLO 系列的一个轻量级版本，专门针对人脸检测进行了优化，具有更高的检测精度和更快的检测速度。

3. 在代码中，通过 YOLO('yolov8n-face.pt', verbose=False)加载预训练的人脸检测模型。verbose=False 参数用于控制模型加载时的输出信息，避免不必要的日志干扰。

(五) 代码分析与结果

一、 代码结构概述

代码由几个主要部分组成：导入必要的库、定义 YOLO 模型、编写人脸检测函数、创建 GUI 窗口和处理拖放事件。以下是对这些部分的详细解释。

（1） 导入必要的库

首先，代码导入了所需的 Python 库。

```
import tkinter as tk
from tkinter import messagebox

import cv2
from PIL import Image, ImageTk
from tkinterdnd2 import DND_FILES, TkinterDnD
from ultralytics import YOLO
```

1. tkinter 和 messagebox 用于创建 GUI 窗口和显示消息框。
2. cv2（OpenCV）用于图像处理，包括读取图像、显示图像和颜色空间转换。
3. PIL（Pillow）和 ImageTk 用于处理图像并将其转换为 Tkinter 可以显示的格式。
4. tkinterdnd2 和 TkinterDnD 用于实现拖放功能。
5. ultralytics.YOLO 用于加载 YOLO 模型并进行人脸检测。

（2） 定义 YOLO 模型

接下来，代码定义了一个 YOLO 模型实例，用于人脸检测。

```
model = YOLO('yolov8n-face.pt', verbose=False)
```

这里使用了一个预训练的 YOLOv8 模型（yolov8n-face.pt），该模型经过优化，专门用于人脸检测。verbose=False 参数表示在加载模型时不显示详细信息。

（3） 编写人脸检测函数

detect_faces 函数是代码的核心部分，它接收一个图像路径作为输入，并返回检测到的人脸数量和带有检测框的图像。

```
def detect_faces(image_path):
try:
img = cv2.imread(image_path)
if img is None:
messagebox.showerror("错误", "无法读取图片")
return 0, None

results = model.predict(img)
result_img = results[0].plot() # 获取带检测框的图像
face_count = 0

for result in results:
boxes = result.boxes.xyxy # 边界框坐标

scores = result.boxes.conf # 置信度分数

classes = result.boxes.cls # 类别索引


class_names = [model.names[int(cls)] for cls in classes]

for box, score, class_name in zip(boxes, scores, class_names):
if class_name == 'face' and score > 0.3:
face_count += 1
print(f"{face_count}")
print("finish test")

return face_count, result_img

except Exception as e:
messagebox.showerror("错误", f"处理图片时发生错误: {str(e)}")
return 0, None
```

1. 首先，使用 cv2.imread 读取图像。如果图像无法读取，则显示错误消息框并返回。

2. 然后，使用 model.predict 进行人脸检测，并获取检测结果。results[0].plot() 用于生成带有检测框的图像。

3. 接下来，遍历检测结果，提取边界框坐标、置信度分数和类别索引。

4. 使用 model.names 将类别索引转换为类别名称。

5. 遍历所有检测到的对象，如果类别名称是“face”且置信度分数大于 0.3，则增加人脸计数器。

6. 最后，打印人脸数量和完成测试的消息，并返回人脸数量和带有检测框的图像。

（4） 创建 GUI 窗口

接下来，代码创建了一个 Tkinter GUI 窗口，用于显示检测结果和接收拖放的图像文件。

```
# 创建 GUI 窗口
root = TkinterDnD.Tk()
root.attributes("-topmost", True)
root.title("人脸计数系统")

# 获取屏幕尺寸
screen_width = root.winfo_screenwidth()
screen_height = root.winfo_screenheight()

# 计算窗口在屏幕中的位置
window_width = 600
window_height = 400
x = (screen_width - window_width) // 2
y = (screen_height - window_height) // 2

# 设置窗口位置和尺寸
root.geometry(f"{window_width}x{window_height}+{x}+{y}")
```

1. 使用 TkinterDnD.Tk()创建一个支持拖放功能的 Tkinter 窗口。

2. 设置窗口标题为“人脸计数系统”。

3. 获取屏幕尺寸并计算窗口在屏幕中的位置，以确保窗口居中显示。

4. 设置窗口的宽度、高度和位置。

（5） 显示图像和结果

接下来，代码创建了用于显示图像和结果的标签。

```
    panel = tk.Label(root)
    panel.pack(padx=10, pady=10)


    result_label = tk.Label(root, text="请拖入图片", font=("Arial", 16))
    result_label.pack(padx=10, pady=10)
```

1. panel 用于显示带有检测框的图像。

2. result_label 用于显示检测到的人脸数量。

（6） 设置拖放功能

最后，代码设置了拖放功能，以允许用户将图像文件拖放到 GUI 窗口中。

```
    # 设置拖放功能
    root.drop_target_register(DND_FILES)
    root.dnd_bind('<<Drop>>', drop)
```

1. 使用 drop_target_register 注册拖放目标。

2. 使用 dnd_bind 绑定<<Drop>>事件到 drop 函数。

（7） 处理拖放事件

drop 函数处理拖放事件，并调用 detect_faces 函数进行人脸检测。

```
    def drop(event):
    file_path = event.data
    count, output_img = detect_faces(file_path)
```

```
        result_label.config(text=f"检测到 {count} 个人")

        if output_img is not None:
        img = Image.fromarray(cv2.cvtColor(output_img, cv2.COLOR_BGR2RGB))
        img.thumbnail((400, 400)) # 调整图像大小
        imgtk = ImageTk.PhotoImage(image=img)
        panel.imgtk = imgtk # 保留引用，防止垃圾回收
        panel.config(image=imgtk)
```

1. 从事件中获取拖放的图像文件路径。
2. 调用 detect_faces 函数进行人脸检测，并获取检测到的人脸数量和带有检测框的图像。
3. 更新 result_label 以显示检测到的人脸数量。
4. 如果检测到图像，则将其转换为 Tkinter 可以显示的格式，并显示在 panel 中。注意，需要保留对 PhotoImage 对象的引用（panel.imgtk = imgtk），以防止它被垃圾回收。

（8） 运行应用程序

最后，使用 root.mainloop()启动 Tkinter 事件循环，以运行 GUI 应用程序。

```
        root.mainloop()
```

(六) 场景应用：

在实际应用场景中，这款系统可以被部署在监控中心的大屏幕上，或者作为一款独立的软件应用在监控人员的电脑上。在一个人流密集的公共场所，如大型购物中心、机场候机厅或火车站候车室，当监控人员需要检测某个区域的人脸数量时，只需简单地将该区域的监控截图或实时拍摄的图像拖动到系统指定的框内，系统便会立即启动人脸检测算法。在短短几秒钟内，系统就能准确识别出图像中的人脸数量，并在界面上清晰展示每个人脸的位置和计数结果。如果检测到异常数量的人脸聚集，系统还会自动发出警报，提醒监控人员及时关注并采取相应的措施，确保公共安全与秩序。

此外，这款系统还可以后续开发数据处理和分析等功能，以实现记录并保存每次检测的结果，包括人脸数量、位置信息、检测时间等，为后续的数据分析和决策提供有力的支持。

(七) 心得体会

在开发这个基于 Python 编程语言和相关库开发的自动人脸检测与计数系统的项目过程中，我们小组深刻体会到了技术创新的魅力和挑战并存的过程。这个项目不仅让我们小组对 python 语言有了更深入的了解，还让我们认识了有更多功能的库，锻炼了我们的编程实践和问题解决能力。

在项目的实现过程中充满了探索与学习的乐趣。从最初对人脸检测技术的懵懂无知，到逐渐熟悉 OpenCV、pillow 等库的使用，再到最终成功构建出能够自动检测和计数人脸数量的系统，每一步都凝聚着我们小团队的智慧和汗水。在这个过程中，我们深刻感受到了技术积累的重要性，以及不断学习和探索新知识的必要性。

项目的实践也让我们对 Python 编程语言的强大功能有了更深刻的认识。Python 的简洁语法和丰富库资源为项目的快速开发提供了极大的便利，也更加坚信了它在数据处理、机器学习等领域的广泛应用前景。

此外，项目的团队合作也让我们受益匪浅。在项目的开发过程中，我们遇到了一些技术难题和挑战，被其中某些代码的实现困住难以继续开发，但是通过团队的共同努力和协作，我们成功地克服了这些困难。这也让我们学会了在调试代码过程中保持冷静，耐心的思考问题所在，慢慢地解决一个个问题。

最后，这个项目的成功实现让我们对 python 的应用前景充满了期待。我相信，通过不断的学习和实践，我们可以利用 python 开发出更多具有实际应用价值的系统软件，为人们的生活带来更多的便利和安全。