基于人工智能的图像处理技术
PBLF2022

姓名：李子诺

学号：2021091201012

第一章 人工智能概述

1.1 人工智能的发展与现状

人类正在进入人工智能时代，全世界产业已经认识到了人工智能的重要性。发达国家将部署人工智能以提升产业竞争力作为一个重要的发展战略。，

人工智能的概念于上个世纪五、六十年代提出。第二次世界大战以前，图灵开始研究机器是否可以思考。在不久之后，一位叫马文·明斯基的研究生于 1951 年提出了关于思维如何萌发并形成的一些基本理论，并建造了世界上第一个神经网络模拟器，名叫 Snare。Snare 的目的是学习如何穿过迷宫。明斯基在 Snare 的基础上，解决了使机器能基于过去行为的知识预测当前行为的结果这一问题，并以“神经网络和脑模型问题”为题完成了博士论文并取得博士学位。

1956 年，约翰·麦卡锡、马文·明斯基、克劳德·香农等科学家在达特茅斯学院举行了一场延续两个月的会议。这次会议上正式出现了“人工智能”这个术语，这被人们看作人工智能正式诞生的标志。在这次会议后的十余年时间里，人工智能被广泛应用于数学和自然语言领域，用来解决代数、几何等问题。但是在七十年代，由于美国科研人员对项目难度的低估，导致社会舆论的压力，经费也被转移到了其他项目上。人工智能迎来了一次低谷。当时人工智能的发展主要受到三个因素的制约，一是那个时候计算机的计算能力有

限，导致很多程序无法得到实际应用。二是深度学习需要大量的数据，而那个时候缺乏足够训练神经网络的数据库。三是早期的人工智能只能解决特定的问题，无法解决复杂的问题。

从上个世纪九十年代开始，随着 AI 技术和神经网络技术的发展，人们开始对 AI 有了初步理性的认识，人工智能再次开始发展。1997 年 5 月 11 日，IBM 研发的计算机系统“深蓝”于国际象棋比赛中战胜了世界冠军卡斯帕罗夫，引发了公众对人工智能的关注。

最近几年，深度学习大热，Google DeepMind 开发智能围棋程序 AlphaGo 战胜世界冠军李世石，被成为计算机历史上的“登月事件”。

1.2 人工智能的应用

人工智能已经被应用到我们生活的方方面面中。例如指纹识别，人脸采集和识别，人体动作识别。人工智能和各行业相结合，形成了智能金融、智能零售、智能交通、智能教育、智能医疗、智能制造、智能健康管理等行业。

1.3 人工智能技术与分类

人工智能的研究主要集中在六大领域，机器学习、自然语言处理、知识表示、机器人学、自动推理和计算机视觉。

机器学习涉及概率论、统计学、算法复杂度理论等多门学科。研究计算机模拟人类的学习行为，并且不停地改善自身的性能。机器学习不仅在基于知识的系统中得到应用，而且在自然语言理解、非单调推理、机器视觉、模式识别等许多领域也得到了广泛应用。一个系统是否具有学习能力已成为是否具有“智能”的一个标志。机器学习的研究主要分为两类研究方向：第一类是传统机器学习的研究，该类研究主要是研究学习机制，注重探索模拟人的

学习机制；第二类是大数据环境下机器学习的研究，该类研究主要是研究如何有效利用信息，注重从巨量数据中获取隐藏的、有效的、可理解的知识。

自然语言处理研究实现人与计算机之间用自然语言进行有效通信的理论和方法。自然语言处理主要应用于机器翻译、舆情监测、自动摘要、观点提取、文本分类、问题回答、文本语义对比、语音识别、中文 OCR 等方面。

第二章 虚拟机与 Ubuntu 系统的安装

在进行项目学习时，在 Win11 操作系统下安装了虚拟机软件 VMware，在 VMware 中使用 Ubuntu 20.04.4 LTS 镜像。

虚拟机是指通过软件模拟的具有完整硬件系统功能的、运行在一个完全隔离环境中的完整计算机系统。在实体计算机中能够完成的工作在虚拟机中都能够实现。通过虚拟机安装其他操作系统，可以避免直接安装双系统的麻烦。通过虚拟机，也能实现在一台机器上同时运行多个操作系统，能够在不同操作系统之间进行文件复制、网络共享。也可以进行不同环境下的软件测试。由于虚拟机和宿主机隔离，虚拟机可以保证宿主机数据的安全。

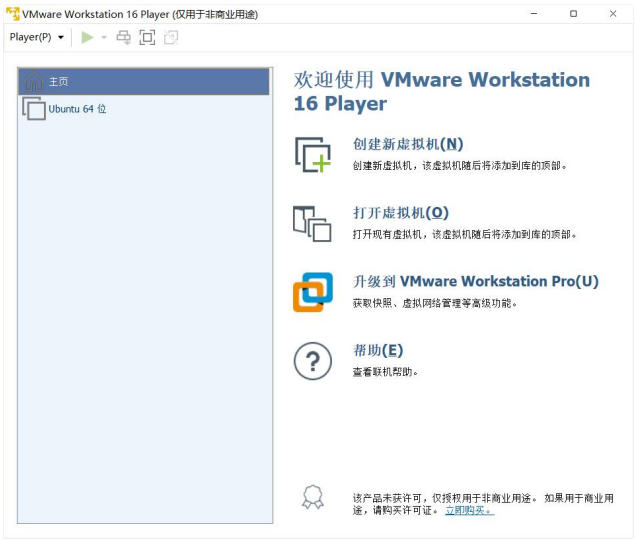

图 1VMware 主界面

安装好 VMware Workstation 16 Player 之后，选择创建新虚拟机，

选择 Ubuntu 光盘映像文件，输入用户名、密码、安装位置等信息来开始安装。

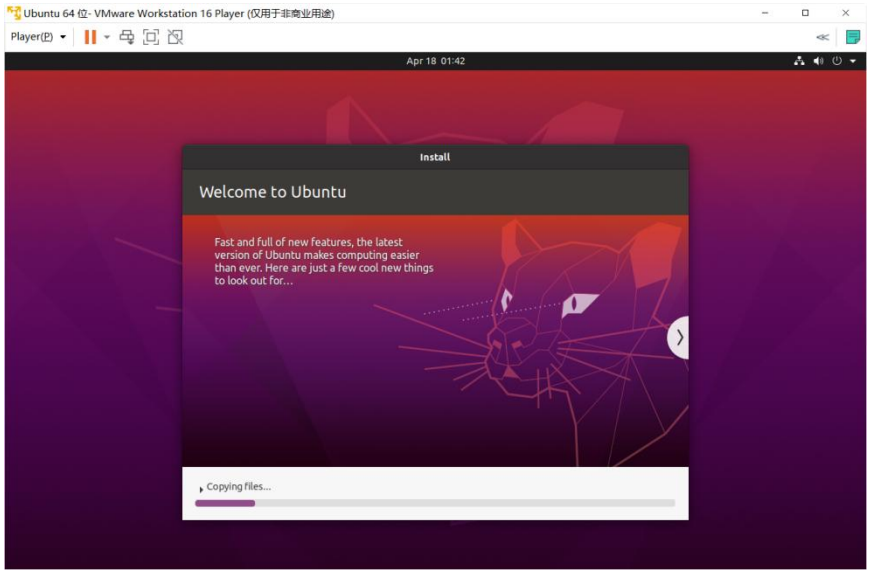

图 2 安装界面

本次项目学习用到的 Ubuntu 是 Linux 操作系统的一种。Linux，是一套免费使用和自由传播的类 Unix 操作系统，具有免费、开源、可扩展性强、安全可靠、系统性能稳定等特点。Ubuntu 的目标在于为一般用户提供一个最新的、同时又相当稳定的主要由自由软件构建而成的操作系统。Ubuntu 具有庞大的社区力量，用户可以方便地从社区获得帮助。

Ubuntu 最大的特点在于强调并且鼓励人们使用、完善并传播开源软件。在本次学习人工智能图像处理时，所用到的 OpenCv 和 EasyPr 都是免费并且开源的。所以我们得以快速、高效、便捷地基于开源项目实现人脸和车牌识别。

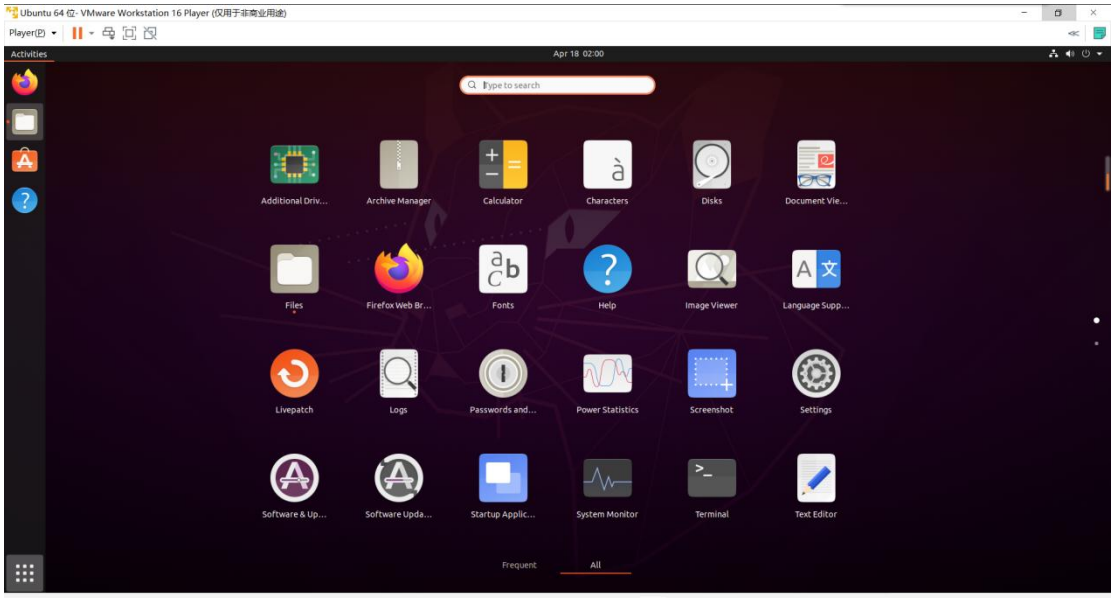

图 3Ubuntu 桌面

同时我们可以看到，Ubuntu 拥有美观的桌面环境。在初次使用 Ubuntu 时，可以获得和 Windows 系统相似的直观体验，容易上手学习。在使用 Ubuntu 时，也能随时打开命令行终端输入命令。

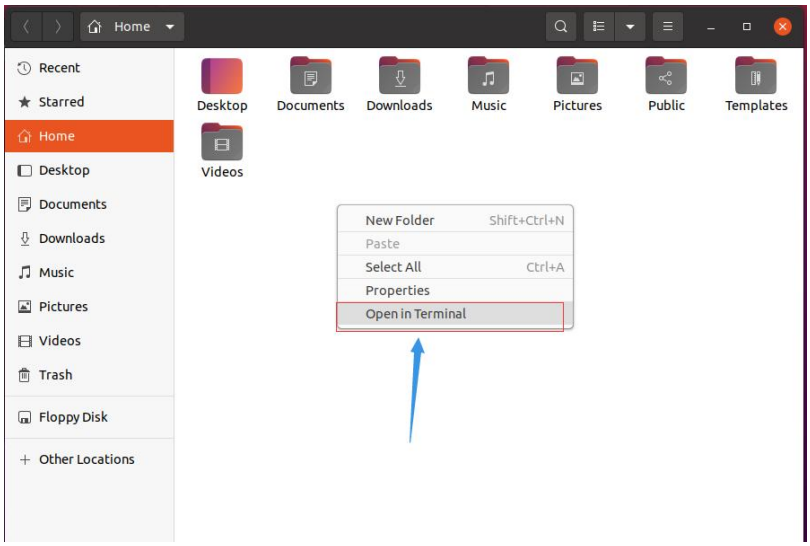

图 4 在指定路径下打开终端

第三章 OpenCV 的安装与调试

3.1CMake 概要与安装

CMake 是一个跨平台的安装工具，可以用简单的语句来描述所有平台的

安装编译功能。CMake 时”Cross platform make”的缩写，意为在所有平台下都能产生标准的构建档，例如 Ubuntu 的 Makefile,然后再按照一般的方式编译使用。这使得开发者可以方便地在不同环境下进行代码的部署。

由于 Ubuntu 的默认软件源列表中有 CMake,直接在终端中输入指令即可完成 CMake 3.16.3 的安装。

```
sudo apt-get install cmake
```

3.2OpenCV 概要与安装

OpenCV 是一个开源的计算机视觉库，它提供了很多函数，这些函数非常高效地实现了计算机视觉算法。OpenCV 支持各种编程语言，如 C++, Python, Java 等，可在不同的平台上使用，包括 Windows，Linux，OS X，Android 和 iOS。OpenCV 支持图像和视频的读写、数字图像的处理、目标识别与跟踪等功能。

OpenCV 在 Ubuntu 20.04 软件源中可用。可以直接通过命令行安装。

```
sudo apt-get update
sudo apt-get install libopencv-dev python3-opencv
```

以上命令会更新软件源至最新版本，并且安装 OpenCV 和所有依赖的安装包。

要通过源码安装 OpenCV，先通过 VMware Tools 将 OpenCV 的源码复制入虚拟机中并解压。

进入源码文件夹后输入指令

```
mkdir build
cd build
cmake ..
make -j8
sudo make install
```

完成 OpenCV 安装。

3.3 C++与 Python 环境

本机使用的 C++编译器是 GCC 9.4.0 版本。GCC 是 GNU 计划的编译器套件，可以处理 C、C++、Java 等语言。GCC 已经被大多数类 Unix 操作系统采纳为标准编译器。它的移植版本 MinGW 在 Windows 的环境下也能使用。

本机使用的 Python 版本为 3.8.10

3.4 OpenCV 调试

3.4.1 OpenCV 样例程序测试

进入~/opencv-3.4.8/samples/cpp/example_cmake 中，打开终端。使用 CMake 和 Make 编译程序。运行程序并成功打开摄像头。

```
cmake .
make
./opencv_example
```

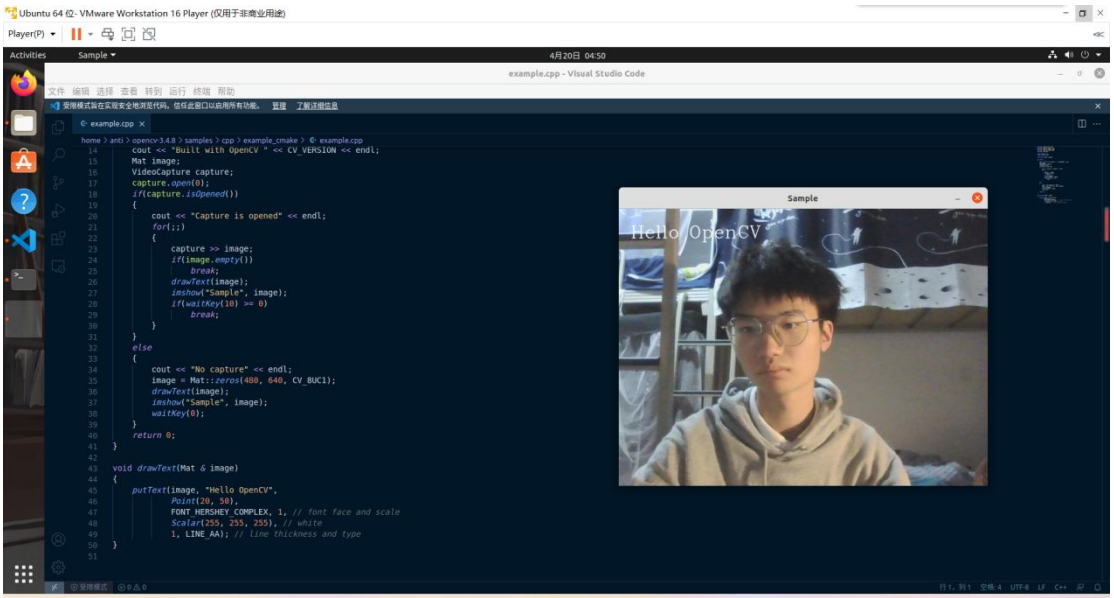

图 5 OpenCV 例程

3.4.2 OpenCV 读取并展示图片和视频

编写读取图片的程序如下。

```
#include <opencv2/opencv.hpp>
```

```cpp
#include <opencv2/opencv.hpp>
#include <iostream>
#include "opencv2/highgui/highgui.hpp"

using namespace cv;
using std::cout;
using std::endl;

int main(int argc, char **argv)
{
 if (argc != 2)
 {
  cout << "No Image File Input" << endl;
  return 0;
 }
 Mat img = imread(argv[1]);
 if (img.empty())
 {
  cout << "Can't Find Image!" << endl;
  return 0;
 }

 namedWindow("Image_test", WINDOW_AUTOSIZE);
 imshow("Image_test", img);
 waitKey(0);
 destroyWindow("Image_test");

 return 0;
}
```

其中，将图片路径通过命令行参数传入程序，并且使用 OpenCV 提供的 imread 函数读入一个 Mat 类。并且新建一个窗口输出。最后销毁窗口。

再通过编写 CMakeLists.txt，生成 Makefile 进行编译。

```
cmake_minimum_required(VERSION 2.8)
project(ImageTest)
find_package(OpenCV REQUIRED)
set(CMAKE_RUNTIME_OUTPUT_DIRECTORY ${CMAKE_SOURCE_DIR})
```

```
add_executable(ImageTest ImageTest.cpp)
target_link_libraries(ImageTest LINK_PRIVATE ${OpenCV_LIBS})
```

`cmake_minimum_required(VERSION 2.8)`指定了 cmake 的最低版本号为 2.8。`project(ImageTest)`指定了工程名为 ImageTest。`find_package(OpenCV REQUIRED)`快速引入文件依赖模块。`set(CMAKE_RUNTIME_OUTPUT_DIRECTORY ${CMAKE_SOURCE_DIR})`指定生成程序的路径。`add_executable`添加了一个可执行文件构筑目标。`target_link_libraries`将目标文件和第三方库链接。

```
mkdir build
cd build
cmake ..
make
cd ..
./ImageTest A.png
```

运行程序结果如图。

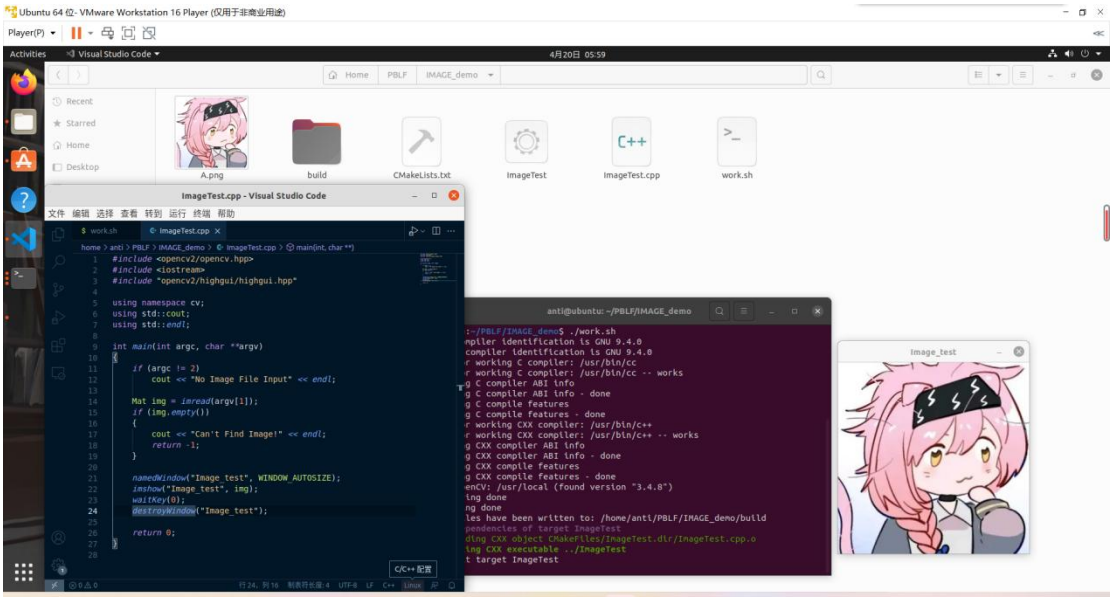

图 6 图片读取

使用 OpenCV 读取视频操作和读取图片相似。不同的是，需要使用 VideoCapture 读取视频以创建一个流，将流的每一帧输入图片类再输出，形成视频的效果。

以下为 Video_test.cpp 内容

```
#include <opencv2/opencv.hpp>
#include <iostream>
using namespace cv;
using std::cout;
using std::endl;
int main(int argv, char **argc)
{
    if (argv != 2)
    {
        cout << "NO PATH" << endl;
        return 0;
    }
    namedWindow("Video_test", 960 * 540);
    VideoCapture cap;
    cap.open(argc[1]);
    Mat frame;
    for (;;)
    {
        cap >> frame;
        if (frame.empty())
            break;
        imshow("Lizn", frame);
        if (waitKey(33) >= 0)
            break;
    }
    return 0;
}
```

编写 CMakeLists.txt 如下

```
cmake_minimum_required(VERSION 2.8)
project(VideoTest)
find_package(OpenCV REQUIRED)
set(CMAKE_RUNTIME_OUTPUT_DIRECTORY ${CMAKE_SOURCE_DIR})
add_executable(VideoTest Video_test.cpp)
target_link_libraries(VideoTest LINK_PRIVATE ${OpenCV_LIBS})
```

编写 Work.sh 如下以进行测试

```
mkdir build
cd build
cmake ..
make
```

```
cd ..
./VideoTest Test_Video.mp4
```

运行后成功播放视频

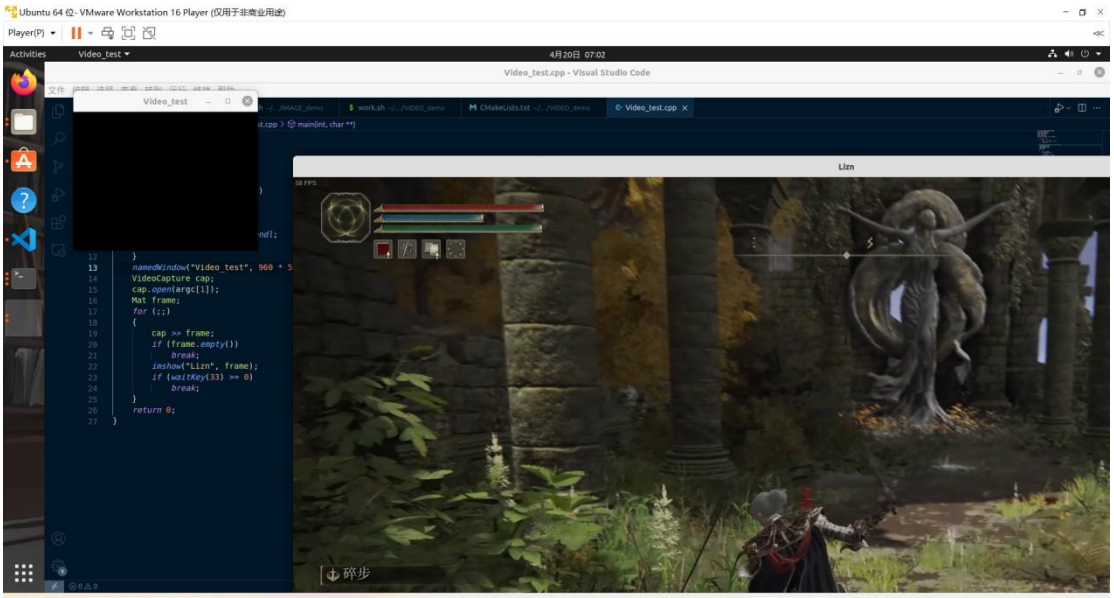

图 7 视频播放

第四章 车牌识别

4.1 车牌识别应用与技术概要

4.4.1 车牌识别应用

一、在公路上的摄像头可以捕捉闯红绿灯、超速等车的图像并且识别车牌。并交由交警部门处理。也可以协助警察逮捕逃逸车辆。

二、小区门口的车牌自动识别车牌号，并在数据库中检索信息、处理。

三、现在很多车辆都装上了 ETC（电子不停车收费系统），司机在通过高速路口时，可以不被人工处理。摄像头拍照，传到电脑上识别后自动放行。在出高速时也能实现自动放行、收费。

4.4.2 车牌识别技术概要

车牌识别首先需要有摄像头对指定区域进行监控，并且将图像上传到系统中心。接下来需要对图片进行预处理，并提取出车牌矩形图片的大概位置。

接下来进行字符分割和字符匹配，最后得到车牌。

车牌识别需要识别不同地区和不同类型的车牌，比如蓝底、黄底、绿底等不同拍照。在获得汽车图片后，用 SVM 模型得到车牌图片。

SVM 全称支持向量机，是一种按照监督学习对数据进行二分的广义线性分类器。SVM 一般有硬间隔支持向量机、软间隔支持向量机、非线性支持向量机。其本质都是通过将数据进行二元分类来进行边界决策。

SVM 在各领域的模式识别问题中有应用，包括人像识别 、文本分类 、手写字符识别、生物信息学等。

在车牌定位时，采用传统的边缘检测算法会出现较大偏差。利用颜色再定位，对 Sobel 定位后的区域进行边界缩小，可以提升定位的准确率。在 EasyPR 中，车牌定位提供了文字定位算法 CMER，边缘定位 SOBEL 和颜色定位算法 COLOR。也可以设置联合使用这些算法。

例如设置同时使用三种探测方式。

```
1. CPlateRecognize pr;
2. pr.setDetectType(PR_DETECT_COLOR | PR_DETECT_SOBEL| PR_DETECT_CM
   SER);
```

接下来对车牌字符进行分割。包括的步骤有车牌图块灰度化、二值化、投影分析、去上下边框、根据阈值进行分割。得到一系列字符块。

接下来对分割后得到的图块进行特征提取，用于字符训练和识别。

EasyPR 采用了 ANN 模型，也即人工神经网络进行字符训练。神经网络是一 种运算模型，由大量的节点（或称神经元）之间相互联接构成。每个节点代表一种特定的输出函数。每两个节点间的连接都代表一个对于通过该连接信号的加权值，称之为权重，神经网络就是通过这种方式来模拟人类的记忆。网络的输出则取决于网络的结构、网络的连接方式、权重和激活函

数。而网络自身通常都是对自然界某种算法或者函数的逼近，也可能是对一种逻辑策略的表达。

4.2 车牌识别工程实现

车牌识别工程利用了 OpenCV 和 EasyPR 的源码。EasyPR 是一个开源的中文车牌识别项目。支持简单、高效地识别车牌。

在此工程中，使用了一个 bool solve(char *filename)函数对指定单张图片进行处理。若不能识别车牌或者打开图片返回值 false，否则返回 true。

主函数通过命令行传参，实现一次程序对多张图片进行识别。如果出现错误输出提示信息。

```
int main(int argc, char *argv[])
{
    for (int i = 1; i < argc; i++)
    {
        if (!solve(argv[i]))
            cout << "couldn't handle " << argv[i] << endl;
    }
}
```

接下来对图片进行处理。

```
    CPlateRecognize pr;
    pr.setResultShow(true);
    pr.setDetectType(PR_DETECT_CMSER | PR_DETECT_SOBEL | PR_DETECT_COLOR);
    pr.setMaxPlates(5);
    vector<CPlate> plateVec;
```

CPlateRecognize pr 声明了一个 EasyPR 提供的车牌识别类。接下来会利用 pr 进行车牌识别操作。

pr.setResultShow(true)设置车牌识别的结果会显示出来。

pr.setDetectType(PR_DETECT_CMSER | PR_DETECT_SOBEL | PR_DETECT_COLOR)设置了车牌识别的模式为文字识别、边缘识别、颜色识别三种。

pr.setMaxPlates(5)设置一张图片车牌最多为 5 张。如果识别结果多于 5 个，会取最有可能是车牌的 5 个。

vector<CPlate> plateVec 声明了一个元素为 CPlate 的向量，CPlate 存储了车牌信息。

Mat src = imread(filename)读取函数参数为路径的图片在 src 中。

接着，使用 plateRecognize 识别车牌。并把结果存放在 plateVec 中。

```
int result = pr.plateRecognize(src, plateVec);
```

如果 result 值为 0，识别失败。Result 值不为 0 则识别成功，显示车牌信息。

接着显示车牌识别信息。

```
    int result = pr.plateRecognize(src, plateVec);
    if (result != 0)
    {
        return false;
    }
    namedWindow(filename, WINDOW_AUTOSIZE);
    for (int i = 0; i < plateVec.size(); i++)
    {
        CPlate plate = plateVec.at(i);
         Mat plateMat = plate.getPlateMat();

         imshow(filename, plateMat);
         waitKey(0);
```

```
        string license = plate.getPlateStr();
        if (license.length() < 7)
            continue;
        cnt++;
        cout << license << endl;
    }
    cout << cnt << " Plate Have Been Found" << endl;
    destroyWindow(filename);
    return true;
```

使用 CMake 生成 Makefile 文件。

```
cmake_minimum_required(VERSION 3.0.0)
project(car)
set(CMAKE_CXX_STANDARD 11)
set(CMAKE_CXX_STANDARD_REQUIRED ON)
find_package(OpenCV 3.2.0 REQUIRED)
include_directories(.)
include_directories(include)
include_directories(${OpenCV_INCLUDE_DIRS})
add_subdirectory(thirdparty)
set(SOURCE_FILES
        src/core/core_func.cpp
        src/core/chars_identify.cpp
        src/core/chars_recognise.cpp
        src/core/chars_segment.cpp
        src/core/feature.cpp
        src/core/plate_detect.cpp
        src/core/plate_judge.cpp
        src/core/plate_locate.cpp
        src/core/plate_recognize.cpp
        src/core/params.cpp

        src/train/ann_train.cpp
        src/train/annCh_train.cpp
        src/train/svm_train.cpp
        src/train/train.cpp
        src/train/create_data.cpp

        src/util/util.cpp
        src/util/program_options.cpp
        src/util/kv.cpp
        )
```

```
add_library(easypr STATIC ${SOURCE_FILES})
set(CMAKE_RUNTIME_OUTPUT_DIRECTORY ${CMAKE_SOURCE_DIR})
set(CMAKE_MODULE_PATH ${PROJECT_SOURCE_DIR}/build)
set(EXECUTABLE_NAME "app")
add_executable(${EXECUTABLE_NAME} app.cpp)
target_link_libraries(${EXECUTABLE_NAME} easypr thirdparty ${OpenCV_LIBS})
```

build.sh 如下

```
mkdir build
cd build
cmake ..
make
```

接下来传入图片进行测试。

```
./app res/test_5.png res/test_2.png res/test_3.png res/test_4.png
```

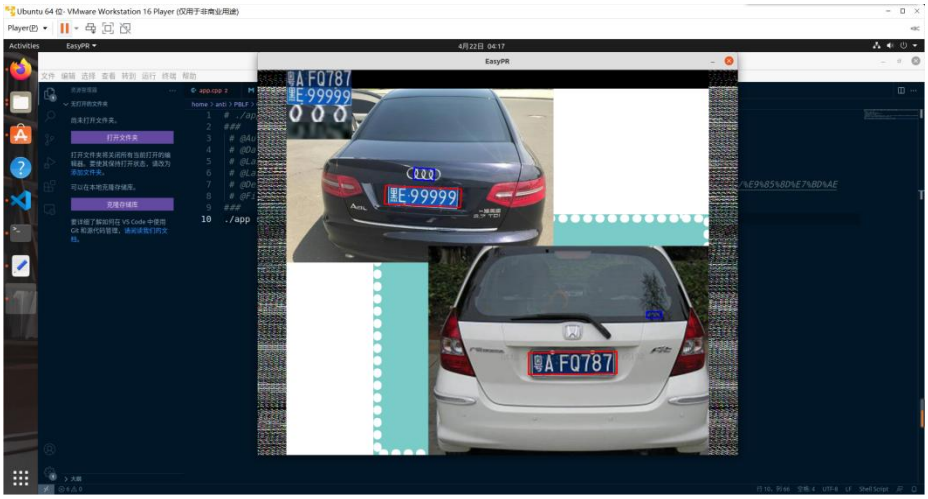

图 8 车牌识别 1

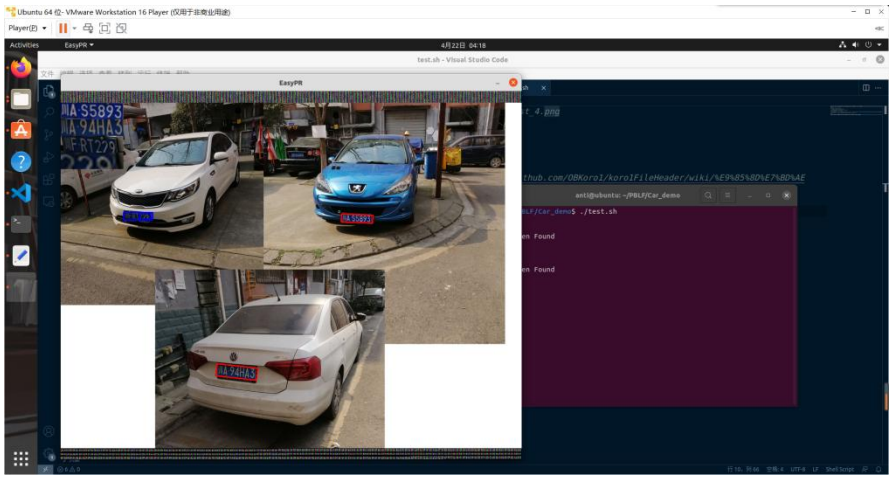

图 9 车牌识别 2

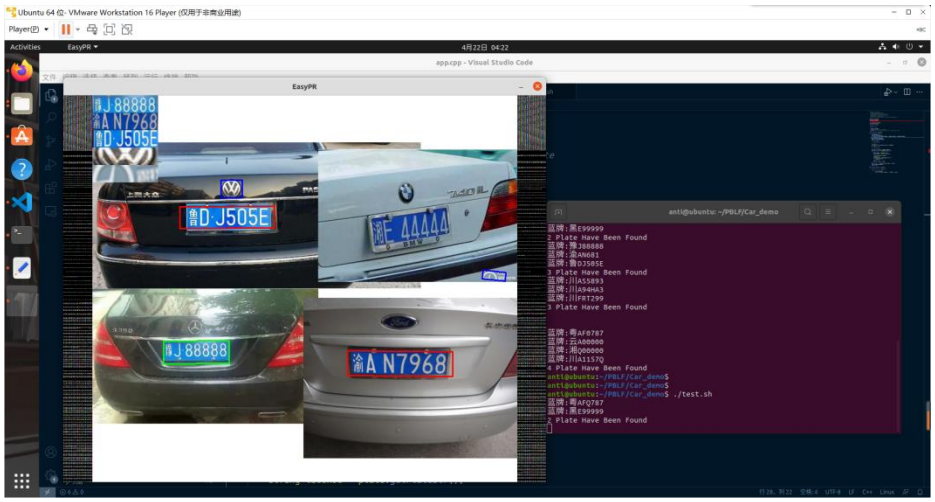

图 10 车牌识别 3

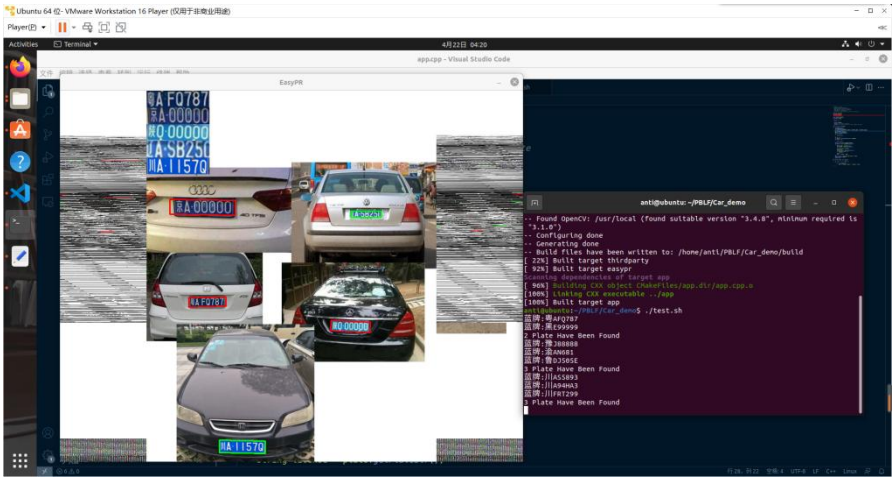

图 11 车牌识别 4

```
[100%] Built target app
anti@ubuntu:~/PBLF/Car_demo$ ./test.sh
蓝牌:粤AFQ787
蓝牌:黑E99999
2 Plate Have Been Found
蓝牌:豫J88888
蓝牌:渝AN681
蓝牌:鲁DJ505E
3 Plate Have Been Found
蓝牌:川AS5893
蓝牌:川A94HA3
蓝牌:川FRT299
3 Plate Have Been Found


蓝牌:粤AF0787
蓝牌:云A00000
蓝牌:湘Q00000
蓝牌:川A1157Q
4 Plate Have Been Found
anti@ubuntu:~/PBLF/Car_demo$
anti@ubuntu:~/PBLF/Car_demo$
```

图 12 识别结果

完整代码如下

```
1. /*
2.  * @Author: lizinuo
3.  * @Date: 2022-04-20 09:59:30
```

```
 * @LastEditTime: 2022-04-22 04:20:27
 * @LastEditors: Please set LastEditors
 * @Description: 车牌识别工程
 * @FilePath: /undefined/home/anti/PBLF/Car_demo/app.cpp
 */
#include <iostream>
#include <string>
#include "easypr.h"

using namespace easypr;
using std::string;
// author: Lizinuo
/**
 * @brief CarRec
 *
 * @param filename
 * @return true Success recognition
 * @return false Cant Open Picture Or cant recognize any plate
 */
bool solve(char *filename)
{
    CPlateRecognize pr;
    pr.setResultShow(true);
    pr.setDetectType(PR_DETECT_CMSER | PR_DETECT_SOBEL | PR_DETECT_COLOR);
    pr.setMaxPlates(5);
    vector<CPlate> plateVec;
    Mat src = imread(filename);
    if (src.empty())
    {
        return false;
    }
    int cnt = 0;
    int result = pr.plateRecognize(src, plateVec);
    if (result != 0)
    {
        return false;
    }
    namedWindow(filename, WINDOW_AUTOSIZE);
    for (int i = 0; i < plateVec.size(); i++)
    {
        CPlate plate = plateVec.at(i);
```

```
45.         Mat plateMat = plate.getPlateMat();
46.
47.         imshow(filename, plateMat);
48.         waitKey(0);
49.         string license = plate.getPlateStr();
50.         if (license.length() < 7)
51.             continue;
52.         cnt++;
53.         cout << license << endl;
54.     }
55.     cout << cnt << " Plate Have Been Found" << endl;
56.     destroyWindow(filename);
57.     return true;
58. }
59. int main(int argc, char *argv[])
60. {
61.     for (int i = 1; i < argc; i++)
62.     {
63.         if (!solve(argv[i]))
64.             cout << "couldn't handle " << argv[i] << endl;
65.     }
66. }
```

第五章 人脸识别

5.1 人脸识别应用与技术概要

人脸识别应用在人脸识别机器人、人脸识别闸机、智能门锁、人脸对比、人脸考勤机、人脸门禁等场景。人员出入小区可以自动抓拍扫描记录，省时省力。通过人脸识别，可以对小区常住人口和流动人口进行分类识别。为小区安防工作带来极大便利。在无人零售店、生活超市中，人们无需携带手机或现金，就能刷脸进行支付。在宿舍楼安装人脸识别摄像头，可以对学生出入情况进行记录，抓拍陌生人，方便管理。在地铁站，可以进行人脸识别乘车。火车站会通过人脸识别和身份证图片比较，快速进行身份认证。

本次人脸识别使用了 dlib 库。Dlib 是一个机器学习的开源库，包含了

机器学习的很多算法。利用 dlib.get_frontal_face_detector()可以实现人脸的检测

人脸识别首先预处理。先将图片灰度化，把彩色图片转化为灰度图。然后进行几何变换，由于成像、采集角度等原因可能使采集到的人脸有一定的变形，可以对图像进行缩放、翻转、仿射、映射等几何变换最大程度地消除。几何变换不改变图像的像素值，而是将像素进行坐标变换，改变像素之间的排列关系，进而将注意力集中在图像内容本身的特征。几何变换由于不改变图片的像素值，无法解决由于灯光等环境因素导致图像呈现出不同情况。需要对灰度阶进行限制，尽量将采集图像和底库照片的灰度阶统一，从肉眼上图片可能会有些失真，但不影响计算机的处理和识别。这种处理方法称为「阈值处理」。图像在形成、传输过程中受到干扰，需要对检测出现的噪声进行过滤，这个过程叫做图像滤波。

接下来进行人脸检测，本次使用了基于模板匹配人脸的检测技术。从数据库中提取人脸模板，接着采取一定的模板匹配策略，使抓取人脸图像与从模板库提取图片相匹配，由相关性的高低和所匹配的模板大小确定人脸的大小和位置信息。Dlib 的 shape_predictor_68_face_landmarks.dat 是已经训练好的人脸关键点检测器。使用了 GBDT，一种基于回归树的人脸对齐算法来提取特征点。

获得特征点之后，需要将特征点转换为特征向量。face_descriptor = facerec.compute_face_descriptor(img, shape)计算出了特征向量。接下来计算欧几里得距离，欧氏距离指两个空间中的向量之间的直线距离，欧氏距离越接近，说明两个向量的差异越小。假设有向量

$x(x_1, x_2, x_3, x_4...)$,$y(y_1, y_2, y_3, y_4...)$。那么它们的欧氏距离为$\sqrt{\sum_{i=1}^{n}(x_i - y_i)^2}$

计算两张人脸特征向量的欧氏距离，欧氏距离越小，说明两张图片中的人脸越相似，当欧氏距离小于某一个值时，认为是同一个人。

5.2 人脸识别工程实现

将人脸图片放入人脸库 candidate-face 中。运行 candidate_train.py 获得人脸库特征信息。

```
# -*- coding: UTF-8 -*-
import sys,os,dlib,numpy
import cv2
# 1.人脸关键点检测器
predictor_path = "shape_predictor_68_face_landmarks.dat"
# 2.人脸识别模型
face_rec_model_path = "dlib_face_recognition_resnet_model_v1.dat
"
# 3.候选人脸文件夹
faces_folder_path = "candidate-face"
# 4.需识别的人脸
img_path = "test-face/0001_IR_allleft.jpg"
# 5.识别结果存放文件夹
faceRect_path = "faceRec"
# 1.加载正脸检测器
detector = dlib.get_frontal_face_detector()
# 2.加载人脸关键点检测器
sp = dlib.shape_predictor(predictor_path)

# 3. 加载人脸识别模型
facerec = dlib.face_recognition_model_v1(face_rec_model_path)
# 候选人脸描述子 list
candidates = []

filelist = os.listdir(faces_folder_path)
count = 0
for fn in filelist:
        count = count+1
descriptors = numpy.zeros(shape=(count,128))
```

```
n = 0
for file in filelist:
    f = os.path.join(faces_folder_path,file)
    #if os.path.splitext(file)[1] == ".jpg" #文件扩展名
    print("Processing file: {}".format(f))
    img = cv2.imread(f)
    # 1.人脸检测
    dets = detector(img, 1)

    for k, d in enumerate(dets):
        # 2.关键点检测
        shape = sp(img, d)

        # 3.描述子提取，128D 向量
        face_descriptor = facerec.compute_face_descriptor(img, shape)
        # 转换为numpy array
        v = numpy.array(face_descriptor)

        descriptors[n] = v

        # descriptors.append(v)
        candidates.append(os.path.splitext(file)[0])

    n += 1

    for d in dets:
        # print("faceRec locate:",d)
        # print(type(d))
        # 使用opencv 在原图上画出人脸位置
        left_top = (dlib.rectangle.left(d), dlib.rectangle.top(d))
        right_bottom = (dlib.rectangle.right(d), dlib.rectangle.bottom(d))
        cv2.rectangle(img, left_top, right_bottom, (0, 255, 0), 2, cv2.LINE_AA)

```

```
    # cv2.imwrite(os.path.join(faceRect_path,file), img)
numpy.save('candidates.npy',descriptors)
file= open('candidates.txt', 'w')
for candidate in candidates:
    file.write(candidate)
    file.write('\n')
file.close()
```

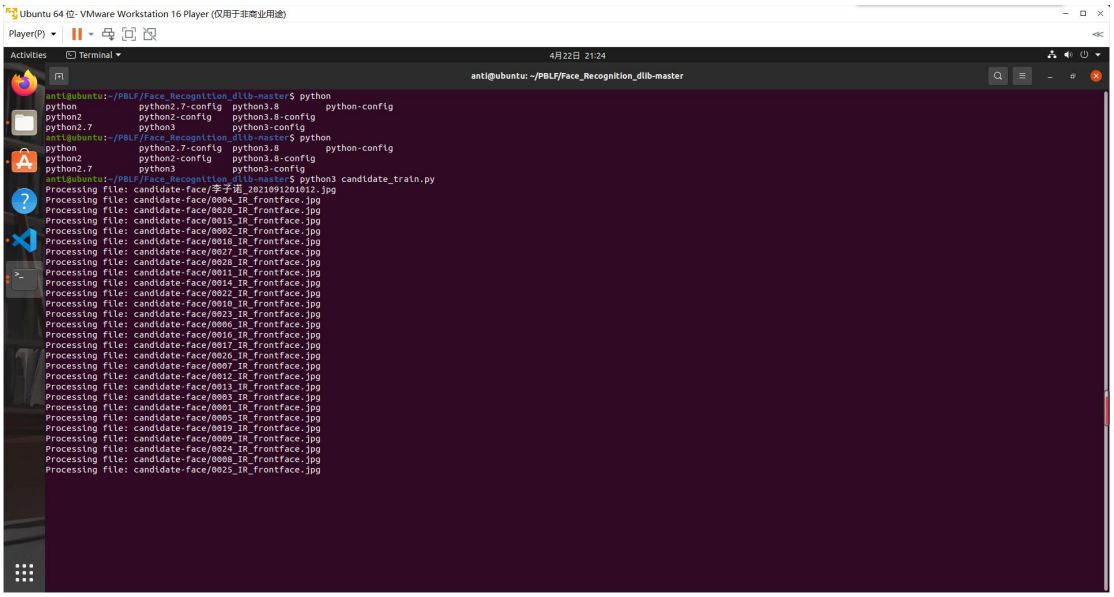

图 13 获得人脸库信息

结果存储在 candidates.npy 和 candidates.txt 中。

接下来打开摄像头，通过摄像头捕捉图像进行识别。

```
# -*- coding: UTF-8 -*-

import dlib,numpy
import cv2
import time
from PIL import Image, ImageDraw, ImageFont
# 1.人脸关键点检测器
predictor_path = "shape_predictor_68_face_landmarks.dat"
# 2.人脸识别模型
face_rec_model_path = "dlib_face_recognition_resnet_model_v1.dat"
# 3.候选人文件
candidate_npydata_path = "candidates.npy"
candidate_path = "candidates.txt"
# 4.储存截图目录
path_screenshots = "screenShots/"
```

```
16.
17.
18. # 加载正脸检测器
19. detector = dlib.get_frontal_face_detector()
20. # 加载人脸关键点检测器
21. sp = dlib.shape_predictor(predictor_path)
22. # 加载人脸识别模型
23. facerec = dlib.face_recognition_model_v1(face_rec_model_path)
24.
25.
26. # 候选人脸描述子 list
27. # 读取候选人数据
28. npy_data=numpy.load(candidate_npydata_path)
29. descriptors=npy_data.tolist()
30. # 候选人名单
31. candidate = []
32. file=open(candidate_path, 'r')
33. list_read = file.readlines()
34. for name in list_read:
35.     name = name.strip('\n')
36.     candidate.append(name)
37.
38. # 创建 cv2 摄像头对象
39. cv2.namedWindow("camera", 1)
40. cap = cv2.VideoCapture(0)
41. cap.set(3, 480)
42. # 截图 screenshots 的计数器
43. cnt = 0
44. def cv2ImgAddText(img, text, left, top, textColor=(255, 255, 0),
    textSize=20):
45.     if (isinstance(img, numpy.ndarray)):  # 判断是否 OpenCV 图片类
    型
46.         img = Image.fromarray(cv2.cvtColor(img, cv2.COLOR_BGR2R
    GB))
47.     # 创建一个可以在给定图像上绘图的对象
48.     draw = ImageDraw.Draw(img)
49.     # 字体的格式
50.     fontStyle = ImageFont.truetype("fonts/truetype/arphic/ukai.
    ttc", textSize, encoding="utf-8")
```

```
    # 绘制文本
    draw.text((left, top), text, textColor, font=fontStyle)
    # 转换回OpenCV格式
    return cv2.cvtColor(numpy.asarray(img), cv2.COLOR_RGB2BGR)
while (cap.isOpened()):  #isOpened()  检测摄像头是否处于打开状态
    ret, img = cap.read()  #把摄像头获取的图像信息保存之 img 变量
    if ret == True:       #如果摄像头读取图像成功
        # 添加提示
        cv2.putText(img, "press 'S': screenshot", (20, 420), cv2.FONT_HERSHEY_PLAIN, 1, (255, 255, 255), 1, cv2.LINE_AA)
        cv2.putText(img, "press 'Q': quit", (20, 440), cv2.FONT_HERSHEY_PLAIN, 1, (255, 255, 255), 1, cv2.LINE_AA)

        # img_gray = cv2.cvtColor(im_rd, cv2.COLOR_RGB2GRAY)
        dets = detector(img, 1)
        if len(dets) != 0:
            # 检测到人脸
            for k, d in enumerate(dets):
                # 关键点检测
                shape = sp(img, d)
                # 遍历所有点圈出来
                for pt in shape.parts():
                    pt_pos = (pt.x, pt.y)
                    cv2.circle(img, pt_pos, 2, (0, 255, 0), 1)
                face_descriptor = facerec.compute_face_descriptor(img, shape)
                d_test2 = numpy.array(face_descriptor)
                # 计算欧式距离
                dist = []
                for i in descriptors:
                    dist_ = numpy.linalg.norm(i - d_test2)
                    dist.append(dist_)
                num = dist.index(min(dist))  # 返回最小值

                left_top = (dlib.rectangle.left(d), dlib.rectangle.top(d))
                right_bottom = (dlib.rectangle.right(d), dlib.rectangle.bottom(d))
                cv2.rectangle(img, left_top, right_bottom, (0, 255, 0), 2, cv2.LINE_AA)
                text_point = (dlib.rectangle.left(d), dlib.rectangle.top(d) - 5)
```

```
                #cv2.putText(img, candidate[num][0:4], text_point, cv2.FONT_HERSHEY_PLAIN, 2.0, (255, 255, 255), 1, 1)  # 标出face
                textimg = cv2ImgAddText(img,candidate[num],dlib.rectangle.left(d),dlib.rectangle.top(d)-30,(255, 255, 0), textSize=20)
            cv2.putText(img, "facesNum: " + str(len(dets)), (20, 50),  cv2.FONT_HERSHEY_PLAIN, 1.5, (0, 0, 0), 2, cv2.LINE_AA)
        else:
            # 没有检测到人脸
            cv2.putText(img, "facesNum:0", (20, 50),  cv2.FONT_HERSHEY_PLAIN, 1.5, (0, 0, 0), 2, cv2.LINE_AA)

        k = cv2.waitKey(1)
        # 按下 's' 键保存
        if k == ord('s'):
            cnt += 1
            print(path_screenshots + "screenshot" + "_" + str(cnt) + "_" + time.strftime("%Y-%m-%d-%H-%M-%S", time.localtime()) + ".jpg")
            cv2.imwrite(path_screenshots + "screenshot" + "_" + str(cnt) + "_" + time.strftime("%Y-%m-%d-%H-%M-%S", time.localtime()) + ".jpg", img)

        # 按下 'q' 键退出
        if k == ord('q'):
            break
        cv2.imshow("camera", textimg)
        #cv2.imshow("camera", img)

# 释放摄像头
cap.release()
cv2.destroyAllWindows()
```

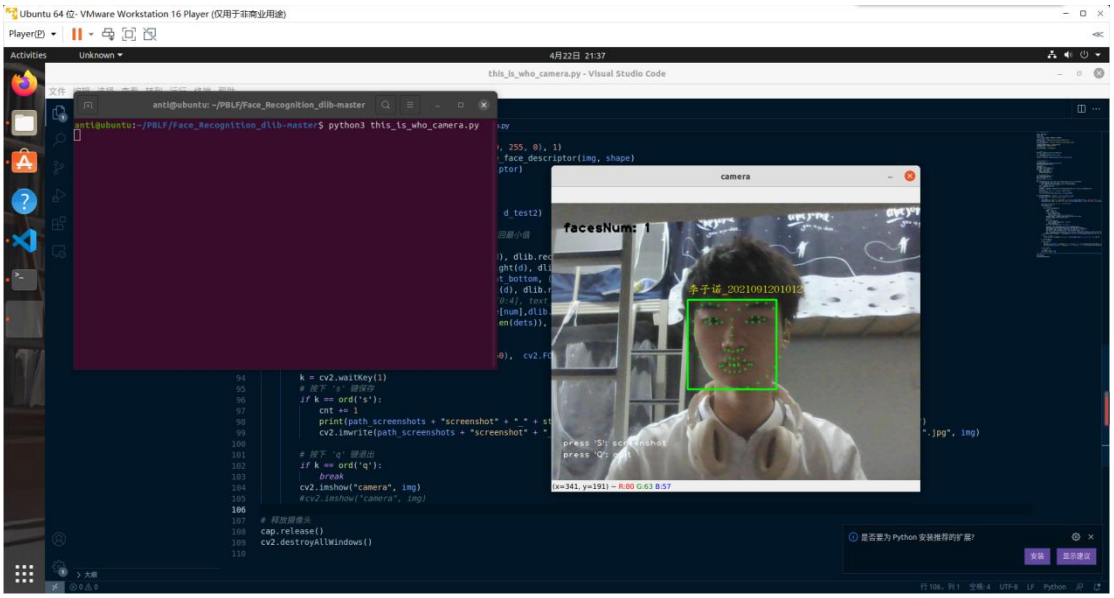

图 14 人脸识别结果

第六章 总结

通过本次学习，我学会了 VMWare 的安装与使用，一些基本的 Linux 操作，以及如何通过编译源码安装 OpenCV

在工程中，我学会了 CMake 的基本使用方法，通过编写 CMakeLists 生成了 Makefile 文件。接着，我学会了用 OpenCV 打开图片和视频。

在车牌识别和人脸识别中，我知道了识别过程都要先进行图片预处理、目标检测、匹配等过程。通过学习，我了解到了 SVM、ANN、特征点提取、特征向量比较等人工智能方面的内容。最后，通过结合开源代码和自己编程，实现了简单的车牌识别和人脸识别。

我建议课程可以增加更多可选择的项目工程来做，比如通过图片生成字符画。

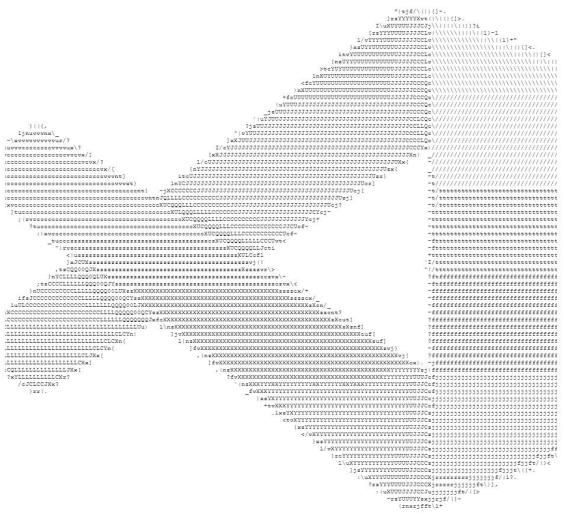

图 15 字符画

也可以加入更多对算法的介绍，讲解更多算法的原理很实现。

总之，我认为这门课程既有理论内容，也有动手实践的环节。让我对计算机视觉内容有了基本的了解。最大的遗憾是课程时间较短，没有对人工智能的算法进行深入的了解。

参考资料

[1] 杨磊.浅谈自然语言处理[J].中国信息技术教育,2022(01):73-76.

[2] 梁玥.基于 SVM 模型的车牌识别应用研究[J].电脑编程技巧与维护,2021(11):115-117.DOI:10.16184/j.cnki.comprg.2021.11.043.

[3] [美]Adrian Kaehler，Gary Bradski.学习 OpenCV3(中文版)[M].刘昌祥等.北京：清华大学出版社, 2018.