# 实验十二、hyperlpr代码分析

**计算机学院 熊明 20305055**

## 一、实验目的

1. 理解hyperlpr的代码大致思路
2. 学习opencv的基本语法
3. 了解神经网络的结构模型

## 二、实验原理

1. opencv简介

   OpenCV 计算机视觉开源库，在OpenCV2.x版本以后，越来越多的函数实现了MatLab所具有的功能，函数名都一样（如 imread, imshow,imwriter等）。

   在计算机内存中，数字图像以矩阵的形式存储和运算。在早期的OpenCV1.x版本中，图像的处理是通过IplImage结构来实现的。早期的OpenCV是用C语言编写，因此提供的借口也是C语言接口，其源代码完全是C的编程风格。IplImage结构是OpenCV矩阵运算的基本数据结构。

   到OpenCV2.x版本，OpenCV开源库引入了面向对象编程思想，大量源代码用C++重写，Mat类(Matrix的缩写) 是OpenCV用于处理图像而引入的一个封装类。从功能上讲，Mat类在IplImage结构的基础上进一步增强，并且，由于引入C++高级编程特性，Mat类的扩展性大大提高，查看Mat类的定义，会发现其设计实现十分全面而具体，基本覆盖计算机视觉对于图像处理的基本要求。

2. hyperlpr2简介

   HyperLPR是一个开源的车牌识别系统，专门用于自动识别和识别车辆的车牌。它基于深度学习技术，利用计算机视觉和图像处理算法来实现高精度的车牌识别。HyperLPR具有以下主要功能和特点：

   1. 车牌识别：HyperLPR可以从图像或视频中准确地检测和识别车辆的车牌。它能够处理各种场景下的车牌，包括不同角度、光照条件和车牌类型。
   2. 高速处理：HyperLPR经过优化，能够在实时或近实时的速度下进行车牌识别。这使得它适用于需要快速处理大量车辆数据的应用场景，如交通监控和停车管理系统。
   3. 多语言支持：HyperLPR支持多种语言的车牌识别，包括中文、英文和其他一些语言。它能够自动检测并识别不同语言的车牌。
   4. 精准度高：HyperLPR经过大量数据集的训练和调优，具有较高的识别准确度。它能够有效地处理复杂的场景和噪声干扰，提供可靠的车牌识别结果。
   5. 易于集成：HyperLPR提供了简单易用的API和接口，便于开发人员将其集成到自己的应用程序或系统中。它支持多种编程语言和平台。

   总而言之，HyperLPR是一个功能强大的车牌识别系统，可以广泛应用于交通管理、安防监控、智能停车等领域，提供高效、准确的车牌识别服务。

## 三、实验内容

# hyperlpr车牌识别分析代码

## 头文件(.h)

### PlateInfo.h

```
#ifndef SWIFTPR_PLATEINFO_H
#define SWIFTPR_PLATEINFO_H

#include <opencv2/opencv.hpp>

namespace pr {

    typedef std::vector<cv::Mat> Character; // 定义Character类型为存储cv::Mat的向量

    // 枚举数据，表示车牌的颜色
    enum PlateColor { BLUE, YELLOW, WHITE, GREEN, BLACK, UNKNOWN };

    // 枚举数据，表示车牌类型
    enum CharType { CHINESE, LETTER, LETTER_NUMS, INVALID };

    class PlateInfo {
    public:
        std::vector<std::pair<CharType, cv::Mat>> plateChars; // 存储车牌字符的类型和图像矩阵的向量
        std::vector<std::pair<CharType, cv::Mat>> plateCoding; // 存储车牌字符编码的类型和图像矩阵的向量
        float confidence = 0; // 识别得分

        // 构造函数，根据不同参数设置车牌信息
        PlateInfo(const cv::Mat& plateData, std::string plateName, cv::Rect plateRect, PlateColor plateType)
            : licensePlate(plateData), name(plateName), ROI(plateRect), Type(plateType) {}

        PlateInfo(const cv::Mat& plateData, cv::Rect plateRect, PlateColor plateType)
            : licensePlate(plateData), ROI(plateRect), Type(plateType) {}

        PlateInfo(const cv::Mat& plateData, cv::Rect plateRect)
            : licensePlate(plateData), ROI(plateRect) {}

        PlateInfo() {}

        cv::Mat getPlateImage() {
            return licensePlate; // 返回车牌图像
        }

        void setPlateImage(cv::Mat plateImage) {
            licensePlate = plateImage; // 设置车牌图像
        }

        cv::Rect getPlateRect() {
            return ROI; // 返回车牌在图像中的矩形区域
```

```
        }

        void setPlateRect(cv::Rect plateRect) {
            ROI = plateRect; // 设置车牌矩形区域
        }

        cv::String getPlateName() {
            return name; // 返回车牌名称
        }

        void setPlateName(cv::String plateName) {
            name = plateName; // 设置车牌名称
        }

        int getPlateType() {
            return Type; // 返回车牌颜色
        }

        int setPlateType(PlateColor plateType) {
            Type = plateType; // 设置车牌颜色
            return 0;
        }

        void appendPlateChar(const std::pair<CharType, cv::Mat>& plateChar)
{
            plateChars.push_back(plateChar); // 添加一个车牌字符到plateChars向
量
        }

        void appendPlateCoding(const std::pair<CharType, cv::Mat>&
charProb) {
            plateCoding.push_back(charProb); // 添加一个车牌字符编码到
plateCoding向量
        }

        std::string decodePlateNormal(std::vector<std::string>
mappingTable) {
            std::string decode;
            for (auto plate : plateCoding) {
                float* prob = (float*)plate.second.data;
                if (plate.first == CHINESE) {
                    decode += mappingTable[std::max_element(prob, prob +
31) - prob]; // 解码中文字符
                    confidence += *std::max_element(prob, prob + 31);
                }
                else if (plate.first == LETTER) {
                    decode += mappingTable[std::max_element(prob + 41, prob
+ 65) - prob]; // 解码字母字符
                    confidence += *std::max_element(prob + 41, prob + 65);
                }
                else if (plate.first == LETTER_NUMS) {
                    decode += mappingTable[std::max_element(prob + 31, prob
+ 65) - prob]; // 解码字母和数字字符
                    confidence += *std::max_element(prob + 31, prob + 65);
                }
```

```
                else if (plate.first == INVALID) {
                    decode += '*'; // 无效字符
                }
            }
            name = decode; // 设置车牌名称为解码结果

            confidence /= 7; // 计算平均识别得分

            return decode;
        }

    private:
        cv::Mat licensePlate; // 车牌图像
        cv::Rect ROI; // 车牌在图像中的矩形区域
        std::string name; // 车牌名称
        PlateColor Type; // 车牌颜色
    };

} // namespace pr

#endif // SWIFTPR_PLATEINFO_H
```

这段代码定义了一个名为`PlateInfo`的类，用于存储车牌信息。它包含了车牌图像、车牌矩形区域、车牌名称和车牌颜色等属性。同时，还定义了用于操作和获取这些属性的成员函数。

- `PlateInfo`类中的`plateChars`和`plateCoding`成员变量是用来存储车牌字符的类型和对应的图像矩阵的向量。`confidence`表示识别得分。
- 该类还提供了构造函数的重载，可以根据传入的参数来初始化车牌信息。`getPlateImage()`函数返回车牌图像，`setPlateImage()`函数设置车牌图像。`getPlateRect()`函数返回车牌在图像中的矩形区域，`setPlateRect()`函数设置车牌矩形区域。`getPlateName()`函数返回车牌名称，`setPlateName()`函数设置车牌名称。`getPlateType()`函数返回车牌颜色，`setPlateType()`函数设置车牌颜色。
- `appendPlateChar()`函数用于将车牌字符添加到`plateChars`向量中，`appendPlateCoding()`函数用于将车牌字符编码添加到`plateCoding`向量中。

`decodePlateNormal()`函数根据传入的映射表对车牌字符编码进行解码，并返回解码后的车牌名称。解码过程根据字符类型选择概率最大的字符编码进行解码，并累加识别得分。

**Platedetect.h**

```
#ifndef _PLATEDETECT_H_
#define _PLATEDETECT_H_

#include <opencv2/opencv.hpp>
#include <vector>
#include "PlateInfo.h"

namespace pr {
    class PlateDetection {
    public:
        // 构造函数，用于初始化PlateDetection对象
        PlateDetection(std::string ssd_prototxt, std::string
ssd_caffe_model);
```

```
13
14          // 使用SSD进行车牌检测，输入图像inputImg，输出车牌信息plateInfos
15          void Detectssd(cv::Mat inputImg, std::vector<pr::PlateInfo>&
    plateInfos);
16
17      private:
18          cv::dnn::Net ssdNet;  // 使用OpenCV的dnn模块的Net类进行SSD网络的定义和操作
19      };
20
21      class DBDetection {
22      public:
23          // 构造函数，用于初始化DBDetection对象
24          DBDetection(std::string cascadestring);
25
26          // 使用DBCascade进行车牌检测，输入图像inputImg，输出车牌信息plateInfos，最小
    宽度min_w和最大宽度max_w
27          void DBDetect(cv::Mat inputImg, std::vector<pr::PlateInfo>&
    plateInfos, int min_w, int max_w);
28
29      private:
30          cv::CascadeClassifier dbcascade;  // 使用OpenCV的CascadeClassifier类进
    行DBCascade的定义和操作
31      };
32  }  // namespace pr
33
34  #endif  // !_PLATEDETECT_H_
35
```

这段代码定义了两个类：`PlateDetection`和`DBDetection`，位于`pr`命名空间中。这两个类用于车牌检测。

- `PlateDetection`类用于基于SSD进行车牌检测。它的构造函数接受两个参数`ssd_prototxt`和`ssd_caffe_model`，用于初始化SSD网络。`Detectssd`函数用于执行车牌检测，接受一个输入图像`inputImg`，并将检测到的车牌信息存储在输出的车牌信息容器`plateInfos`中。
- `DBDetection`类用于基于DBCascade进行车牌检测。它的构造函数接受一个参数`cascadestring`，用于初始化DBCascade。`DBDetect`函数用于执行车牌检测，接受一个输入图像`inputImg`，并将检测到的车牌信息存储在输出的车牌信息容器`plateInfos`中。还可以通过`min_w`和`max_w`参数指定车牌的最小宽度和最大宽度。

私有成员变量`ssdNet`是一个OpenCV的`Net`类对象，用于定义和操作SSD网络。

私有成员变量`dbcascade`是一个OpenCV的`CascadeClassifier`类对象，用于定义和操作DBCascade。

**Finetune.h**

```
1  #ifndef _FINETUNE_H_
2  #define _FINETUNE_H_
3
4  #include <vector>
5  #include <opencv2/dnn.hpp>
6  #include <opencv2/opencv.hpp>
7
8  namespace pr {
9      class FineTune {
```

```
    public:
        // 构造函数，用于初始化FineTune对象
        FineTune(std::string finetune_prototxt, std::string
finetune_caffemodel);

        // 执行微调操作，输入图像img，输出结果图像resImg
        void Finetune(cv::Mat img, cv::Mat& resImg);

        // 对图像进行细化处理，输入原始图像img，标注点集pts，输出细化后的图像out
        void to_refine(cv::Mat img, std::vector<cv::Point> pts, cv::Mat&
out);

        // 对图像进行仿射裁剪，输入原始图像img，标注点集pts，输出裁剪后的图像out
        void affine_crop(cv::Mat img, std::vector<cv::Point> pts, cv::Mat&
out);

    private:
        cv::dnn::Net FTNet;  // 使用OpenCV的dnn模块的Net类进行微调网络的定义和操作
    };
}  // namespace pr

#endif  // !_FINETUNE_H_
```

这段代码是一个名为`FineTune`的类的定义，位于`pr`命名空间中。它包含了用于图像微调的一些函数和私有成员变量。

- `FineTune`类的构造函数，接受两个参数`finetune_prototxt`和`finetune_caffemodel`，用于初始化微调网络。它会加载指定的模型配置文件和权重文件。
- `Finetune`函数用于执行微调操作。它接受一个输入图像`img`，并将结果存储在输出图像`resImg`中。
- `to_refine`函数对图像进行细化处理。它接受一个原始图像`img`和一个标注点集`pts`，并将细化后的结果存储在输出图像`out`中。
- `affine_crop`函数对图像进行仿射裁剪。它接受一个原始图像`img`和一个标注点集`pts`，并将裁剪后的结果存储在输出图像`out`中。
- 私有成员变量`FTNet`是一个OpenCV的`Net`类对象，用于定义和操作微调网络。

**PlateRecognation.h**

```
#ifndef _PLATERECOGNATION_H_
#define _PLATERECOGNATION_H_

#include <opencv2/dnn.hpp>
#include "PlateInfo.h"

namespace pr {
    class PlateRecognation {
    public:
        // 构造函数，用于初始化PlateRecognation对象
        PlateRecognation(std::string Rec_prototxt, std::string
Rec_cafffemodel);

        // 进行无分割的车牌识别，输入原始图像src，输出车牌信息plateinfo
        void segmentation_free_recognation(cv::Mat src, pr::PlateInfo&
plateinfo);
```

```
15
16      private:
17          cv::dnn::Net RecNet;  // 使用OpenCV的dnn模块的Net类进行识别网络的定义和操
    作
18      };
19  }  // namespace pr
20
21  #endif  // !_PLATERECOGNATION_H_
```

这段代码是一个名为`PlateRecognation`的类的定义，位于`pr`命名空间中。它包含了进行车牌识别的一些函数和私有成员变量。

- `PlateRecognation`类的构造函数，接受两个参数`Rec_prototxt`和`Rec_cafffemodel`，用于初始化识别网络。它会加载指定的模型配置文件和权重文件。
- `segmentation_free_recognation`函数用于进行无分割的车牌识别。它接受一个原始图像`src`，并将识别结果存储在输出的车牌信息对象`plateinfo`中。
- 私有成员变量`RecNet`是一个OpenCV的`Net`类对象，用于定义和操作识别网络。

**Pipeline.h**

```
1   #pragma warning(disable:4430)  // 禁用警告4430
2   #ifndef _PIPLINE_H
3   #define _PIPLINE_H
4
5   #include <vector>
6   #include "Finetune.h"
7   #include "Platedetect.h"
8   #include "PlateRecognation.h"
9   //#include "PlateColor.h"
10
11  using namespace std;  // 引用命名空间std
12  using namespace cv;   // 引用命名空间cv
13
14  namespace pr {
15      // 建立常量向量数组 CH_PLATE_CODE，表示中国车牌所出现的字符
16      const std::vector<std::string> CH_PLATE_CODE{
17          "京", "沪", "津", "渝", "冀", "晋", "蒙", "辽", "吉", "黑", "苏", "浙",
    "皖", "闽", "赣", "鲁", "豫", "鄂", "湘", "粤", "桂",
18          "琼", "川", "贵", "云", "藏", "陕", "甘", "青", "宁", "新", "0", "1",
    "2", "3", "4", "5", "6", "7", "8", "9", "A",
19          "B", "C", "D", "E", "F", "G", "H", "J", "K", "L", "M", "N", "P",
    "Q", "R", "S", "T", "U", "V", "W", "X",
20          "Y", "Z", "港", "学", "使", "警", "澳", "挂", "军", "北", "南", "广",
    "沈", "兰", "成", "济", "海", "民", "航", "空"
21      };
22
23      class PipelinePR {
24      public:
25          PlateDetection* platedetection;
26          FineTune* finetune;
27          PlateRecognation* platerecognation;
28          DBDetection* dbdetection;
29
30          // 构造函数，用于初始化 PipelinePR 对象，接受多个参数用于初始化内部对象
```

```
        PipelinePR(std::string detect_prototxt, std::string
detect_caffemodel,
            std::string finetune_prototxt, std::string finetune_caffemodel,
            std::string platerec_prototxt, std::string platerec_caffemodel,
            std::string dbstring);

        // 析构函数，用于释放内部对象的资源
        ~PipelinePR();

        std::vector<std::string> plateRes;  // 车牌结果容器
        std::vector<PlateInfo> RunPiplineAsImage(cv::Mat srcImage, int
IsDB);  // 执行车牌识别流程的函数
    };
}  // namespace pr

#endif  // !_PIPLINE_H
```

这段代码定义了一个名为 `PipelinePR` 的类，位于 `pr` 命名空间中。它用于执行车牌识别的流程。

- `PipelinePR` 类包含了几个指向其他类对象的指针成员，分别是 `platedetection`（PlateDetection 类的对象指针）、`finetune`（FineTune 类的对象指针）、`platerecognation`（PlateRecognation 类的对象指针）和 `dbdetection`（DBDetection 类的对象指针）。
- 构造函数 `PipelinePR` 接受多个参数，用于初始化内部对象。具体参数包括 `detect_prototxt` 和 `detect_caffemodel`（用于初始化 `PlateDetection` 对象）、`finetune_prototxt` 和 `finetune_caffemodel`（用于初始化 `FineTune` 对象）、`platerec_prototxt` 和 `platerec_caffemodel`（用于初始化 `PlateRecognation` 对象）以及 `dbstring`（用于初始化 `DBDetection` 对象）。
- 析构函数 `~PipelinePR` 用于释放内部对象的资源。
- 成员函数 `RunPiplineAsImage` 执行车牌识别的流程，接受输入图像 `srcImage` 和参数 `IsDB`，并将识别结果存储在成员变量 `plateRes` 和返回的 `PlateInfo` 向量中。

## 源文件(.cpp)

### FineTune.cpp

```
#include "../include/Finetune.h"
using namespace std;
using namespace cv;

namespace pr {
    FineTune::FineTune(string finetune_prototxt, string finetune_caffemodel)
{
        // 从给定的 prototxt 和 caffemodel 文件读取深度学习网络
        FTNet = dnn::readNetFromCaffe(finetune_prototxt,
finetune_caffemodel);
    }

    void FineTune::affine_crop(Mat src, vector<Point> pts, Mat &crop)
    {
        // 定义源图像上的四个点坐标
        Point2f dst[4] = {
Point2f(0,0),Point2f(160,0),Point2f(160,40),Point2f(0,40) };
```

```
        Point2f srcpt[4] = { Point2f(pts[0]),Point2f(pts[1])
,Point2f(pts[2]) ,Point2f(pts[3]) };
        // 获取透视变换矩阵
        Mat _mat = getPerspectiveTransform(srcpt, dst);
        // 进行透视变换
        warpPerspective(src, crop, _mat, Size(160, 40));
    }

    void FineTune::to_refine(Mat src, vector<Point> pts, Mat& crop)
    {
        // 定义中心点和宽高
        float scale = 3.f;
        int cx = 64; int cy = 24;
        int cw = 64; int ch = 24;
        // 定义目标图像上的四个点坐标
        int tx1 = cx - cw / 2;
        int ty1 = cy - ch / 2;
        int tx2 = cx + cw / 2;
        int ty2 = cy - ch / 2;
        int tx3 = cx + cw / 2;
        int ty3 = cy + ch / 2;
        int tx4 = cx - cw / 2;
        int ty4 = cy + ch / 2;
        vector<Point2f> dstp(4);
        Point2f dst[4] = { (Point2f(tx1*scale, ty1*scale)) ,
(Point2f(tx2*scale, ty2*scale)) ,(Point2f(tx3*scale, ty3*scale)) ,
(Point2f(tx4*scale, ty4*scale)) };
        Point2f pt[4] = { Point2f(pts[0]),Point2f(pts[1]) ,Point2f(pts[2])
,Point2f(pts[3]) };
        // 获取透视变换矩阵
        Mat _mat = getPerspectiveTransform(pt, dst);
        // 进行透视变换
        warpPerspective(src, crop, _mat, Size(120 * scale, 48 * scale));
    }

    void FineTune::Finetune(Mat src, Mat& dst)
    {
        // 将源图像调整为指定大小
        Mat tof;
        resize(src, tof, Size(120, 48));
        // 将图像转换为网络输入的 blob
        Mat blob = dnn::blobFromImage(tof, 0.0078125, Size(120, 48),
Scalar(127.5, 127.5, 127.5), false, false);
        // 设置网络的输入 blob
        FTNet.setInput(blob);
        // 运行网络前向传播，获取输出 blob
        Mat outblob = FTNet.forward("conv6-3");

        float *data = outblob.ptr<float>();
        vector<Point> pts(4);
        // 将输出 blob 中的数据转换为四个点的坐标
        Mat fineMat(Size(2, 4), CV_32F, data);
        for (int i = 0; i < fineMat.rows; i++)
        {
            pts[i].x = fineMat.at<float>(i, 0)*src.cols;
```

```
            pts[i].y = fineMat.at<float>(i, 1)*src.rows;
        }
        Mat crop;
        // 对裁剪的图像进行细化
        to_refine(src, pts, crop);
        // 将裁剪后的图像转换为网络输入的 blob
        blob = dnn::blobFromImage(crop, 0.0078128, Size(120, 48),
Scalar(127.5, 127.5, 127.5), false, false);
        // 设置网络的输入 blob
        FTNet.setInput(blob);
        // 运行网络前向传播，获取输出 blob
        outblob = FTNet.forward("conv6-3");
        data = outblob.ptr<float>();
        // 将输出 blob 中的数据转换为四个点的坐标
        Mat fineMat2(Size(2, 4), CV_32F, data);
        for (int i = 0; i < fineMat.rows; i++)
            {
            pts[i].x = fineMat2.at<float>(i, 0)*crop.cols;
            pts[i].y = fineMat2.at<float>(i, 1)*crop.rows;
        }
        // 对裁剪的图像进行仿射变换
        affine_crop(crop, pts, crop);
        // 将最终的裁剪图像复制到 dst 中
        dst = crop.clone();
    }
}
```

**Pipeline.cpp**

```
#include "../include/Pipeline.h"

namespace pr {
    PipelinePR::PipelinePR(std::string detect_prototxt, std::string
detect_caffemodel,
        std::string finetune_prototxt, std::string finetune_caffemodel,
        std::string platerec_prototxt, std::string platerec_caffemodel,
        std::string dbstring)
    {
        // 创建车牌检测对象
        platedetection = new PlateDetection(detect_prototxt,
detect_caffemodel);
        // 创建细化对象
        finetune = new FineTune(finetune_prototxt, finetune_caffemodel);
        // 创建车牌识别对象
        platerecognation = new PlateRecognation(platerec_prototxt,
platerec_caffemodel);
        // 创建车牌数据库检测对象
        dbdetection = new DBDetection(dbstring);
    }

    PipelinePR::~PipelinePR()
    {
        // 释放内存
        delete platedetection;
        delete finetune;
```

```
        delete platerecognation;
        delete dbdetection;
    }

    cv::Mat DBcropFromImage(const cv::Mat &image){
        cv::Mat cropped;
        image.copyTo(cropped);
        int cropped_w = cropped.cols;
        int cropped_h = cropped.rows;
        // 定义上半部分和下半部分的矩形区域
        cv::Rect up,down;
        up.y = cropped_h*0.05;up.x = cropped_w*0.2;up.height = cropped_h*0.35;up.width = cropped_w*0.55;
        down.y = cropped_h*0.4;down.x = cropped_w*0.05;down.height = cropped_h*0.6;down.width = cropped_w*0.95;
        cv::Mat cropUp,cropDown;
        // 截取上半部分和下半部分的图像
        cropped(up).copyTo(cropUp);
        cropped(down).copyTo(cropDown);
        // 调整图像尺寸
        cv::resize(cropUp,cropUp,cv::Size(64,40));
        cv::resize(cropDown,cropDown,cv::Size(96,40));
        // 创建最终的车牌图像
        cv::Mat crop = cv::Mat(40,160,CV_8UC3);
        cropUp.copyTo(crop(cv::Rect(0,0,64,40)));
        cropDown.copyTo(crop(cv::Rect(64,0,96,40)));
        return crop;
    }

    std::vector<PlateInfo> PipelinePR::RunPiplineAsImage(cv::Mat plateimg,int IsDB)
    {
        std::vector<pr::PlateInfo> plates;
        std::vector<PlateInfo> plateres;
        if(IsDB==1)
        {
            // 调用车牌数据库检测方法
            dbdetection->DBDetect(plateimg,plates,30,1280);
        }
        else
        {
            // 调用车牌检测方法
            platedetection->Detectssd(plateimg, plates);
        }
        for (pr::PlateInfo plateinfo : plates) {
            cv::Mat image = plateinfo.getPlateImage();
            cv::Mat CropImg;

            if(IsDB==1)
            {
                // 调用车牌数据库识别方法
                CropImg = DBcropFromImage(image);
                platerecognation->segmentation_free_recognation(CropImg, plateinfo);
            }
```

```
            else
            {
                // 调用车牌细化和识别方法
                finetune->Finetune(image, CropImg);
                platerecognation->segmentation_free_recognation(CropImg, plateinfo);
            }
            // 将识别结果存入结果集合
            plateres.push_back(plateinfo);
        }
        return plateres;
    }
}
```

**PlateDection.cpp**

```
#include "../include/Platedetect.h"

using namespace cv;
using namespace std;
namespace pr {

    // 车牌检测器构造函数，加载SSD模型
    PlateDetection::PlateDetection(std::string ssd_prototxt, std::string ssd_caffemodel)
    {
        ssdNet = cv::dnn::readNetFromCaffe(ssd_prototxt, ssd_caffemodel);
    }

    // 车牌级联分类器构造函数，加载级联分类器模型
    DBDetection::DBDetection(std::string cascadestring)
    {
        dbcascade.load(cascadestring);
    }

    // 使用SSD模型检测车牌
    void PlateDetection::Detectssd(cv::Mat img, std::vector<pr::PlateInfo> &plateInfos)
    {
        int cols = img.cols;
        int rows = img.rows;

        // 将图像转换为32位浮点数类型
        Mat in;
        img.convertTo(in, CV_32F);

        // 创建输入Blob
        Mat input(img.size(), CV_32FC3);
        Mat inputblob1 = input.reshape(1, { 1, 3,rows,cols });
        Mat input_blob = dnn::blobFromImages(in, 0.225, Size(), Scalar(103.53, 116.28, 123.675), false);
        float *blobdata = input_blob.ptr<float>();
        float *blobdata2 = inputblob1.ptr<float>();
        {
            // 将Blob数据拷贝到输入Blob中
```

```
            for (int i = 0; i < rows; i++)
            {
                memcpy(blobdata2 + i * cols, blobdata + 3 * i * cols, cols * sizeof(float));
                memcpy(blobdata2 + i * cols + rows * cols, blobdata + (1 + 3 * i) * cols, cols * sizeof(float));
                memcpy(blobdata2 + i * cols + rows * cols * 2, blobdata + (2 + 3 * i) * cols, cols * sizeof(float));
            }
        }

        // 设置输入Blob
        ssdNet.setInput(inputblob1);

        // 前向传播获取输出Blob
        Mat outputBlob = ssdNet.forward("detection_out");

        // 解析输出Blob中的检测结果
        Mat detectmat(outputBlob.size[2], outputBlob.size[3], CV_32F, outputBlob.ptr<float>());
        for (int i = 0; i < detectmat.rows; i++)
        {
            float confidence = detectmat.at<float>(i, 2);

            // 如果置信度大于0.5，则认为是一个车牌
            if (confidence > 0.5)
            {
                int x1, x2, y1, y2;
                Rect rec;
                Mat cimg;
                x1 = int(detectmat.at<float>(i, 3) * cols);
                y1 = int(detectmat.at<float>(i, 4) * rows);
                x2 = int(detectmat.at<float>(i, 5) * cols);
                y2 = int(detectmat.at<float>(i, 6) * rows);

                // 对检测框的坐标进行修正，确保不超出图像边界
                x1 = max(x1, 0);
                y1 = max(y1, 0);
                x2 = min(x2, cols - 1);
                y2 = min(y2, rows - 1);

                // 提取车牌图像
                rec.x = x1; rec.y = y1; rec.width = (x2 - x1 + 1); rec.height = (y2 - y1 + 1);
                img(rec).copyTo(cimg);

                // 创建PlateInfo对象，并将车牌图像和位置信息保存到plateInfos向量中
                PlateInfo plateInfo(cimg, rec);
                plateInfos.push_back(plateInfo);
            }
        }
    }

    // 从图像中裁剪指定区域的图像
    cv::Mat cropFromImage(const cv::Mat &image, cv::Rect rect)
```

```
    {
        int w = image.cols - 1;
        int h = image.rows - 1;

        // 确保裁剪区域不超出图像边界
        rect.x = std::max(rect.x, 0);
        rect.y = std::max(rect.y, 0);
        rect.height = std::min(rect.height, h - rect.y);
        rect.width = std::min(rect.width, w - rect.x);

        // 裁剪图像并返回
        cv::Mat temp(rect.size(), image.type());
        cv::Mat cropped;
        temp = image(rect);
        temp.copyTo(cropped);
        return cropped;
    }

    // 使用级联分类器模型检测车牌
    void DBDetection::DBDetect(cv::Mat InputImage,
std::vector<pr::PlateInfo> &plateInfos, int min_w, int max_w)
    {
        cv::Mat processImage;
        cv::cvtColor(InputImage, processImage, cv::COLOR_BGR2GRAY);
        std::vector<cv::Rect> platesRegions;

        // 设置车牌的最小和最大尺寸
        cv::Size minSize(min_w, min_w / 4);
        cv::Size maxSize(max_w, max_w / 4);

        // 如果处理的图像为空，则直接返回
        if (&processImage == NULL)
            return;

        // 使用级联分类器模型检测车牌
        dbcascade.detectMultiScale(processImage, platesRegions,
            1.1, 3, cv::CASCADE_SCALE_IMAGE, minSize, maxSize);

        // 对检测到的车牌区域进行进一步处理和修正，并将结果保存到plateInfos向量中
        for (auto plate : platesRegions)
        {
            int zeroadd_w = static_cast<int>(plate.width * 0.28);
            int zeroadd_h = static_cast<int>(plate.height * 0.35);
            int zeroadd_x = static_cast<int>(plate.width * 0.14);
            int zeroadd_y = static_cast<int>(plate.height * 0.15);

            // 对车牌区域进行修正
            plate.x -= zeroadd_x;
            plate.y -= zeroadd_y;
            plate.height += zeroadd_h;
            plate.width += zeroadd_w;

            // 从原始图像中裁剪出车牌图像
            cv::Mat plateImage = cropFromImage(InputImage, plate);

```

```
                // 创建PlateInfo对象，并将车牌图像和位置信息保存到plateInfos向量中
                PlateInfo plateInfo(plateImage, plate);
                plateInfos.push_back(plateInfo);
            }
        }
    }
```

**PlateRecognation.cpp**

```
#include "../include/PlateRecognation.h"
#include "../include/Pipeline.h"
using namespace std;
using namespace cv;

namespace pr {

    // 构造函数：初始化识别网络
    PlateRecognation::PlateRecognation(std::string rec_prototxt, std::string rec_caffemodel)
    {
        RecNet = cv::dnn::readNetFromCaffe(rec_prototxt, rec_caffemodel);
    }

    // 车牌字符分割自由识别函数
    void PlateRecognation::segmentation_free_recognation(cv::Mat src, pr::PlateInfo &plateinfo)
    {
        float score = 0; // 识别得分
        string text = ""; // 识别结果
        Mat src1 = src.clone(); // 克隆原始图像
        Mat inputMat(Size(40, 160), CV_8UC3); // 创建输入图像的尺寸

        // 将原始图像数据复制到输入图像中
        for (int j = 0; j < src.rows; j++)
        {
            for (int i = 0; i < src.cols; i++)
            {
                inputMat.at<Vec3b>(i, j) = src1.at<Vec3b>(j, i);
            }
        }

        // 将输入图像转换为网络可接受的格式
        Mat blob = dnn::blobFromImage(inputMat, 1 / 255.f, Size(40, 160), Scalar(0, 0, 0), false, false);

        // 设置网络的输入
        RecNet.setInput(blob);

        // 进行前向传播，获取网络输出
        Mat outblob = RecNet.forward();

        int x = outblob.size[2];
        int y = outblob.size[0];

        float *data = outblob.ptr<float>();
```

```
        vector<float> scores(84); // 存储字符的分数
        vector<int>maxidxs; // 存储每个字符的最大分数索引
        vector<float> maxscore; // 存储每个字符的最大分数

        // 遍历网络输出，获取每个字符的分数和最大分数索引
        for (int i = 2; i < 20; i++)
        {
            for (int j = 0; j < 84; j++)
            {
                scores[j] = data[j * 20 + i];
            }
            int idx = max_element(scores.begin(), scores.end()) -
scores.begin();
            maxidxs.push_back(idx);
            maxscore.push_back(scores[idx]);
        }

        int charnum = 0; // 字符数量

        // 将最大分数索引转换为字符，并计算总体得分
        for (int i = 0; i < maxidxs.size(); i++)
        {
            if (maxidxs[i] < pr::CH_PLATE_CODE.size() && (i == 0 ||
(maxidxs[i - 1] != maxidxs[i])))
            {
                text += pr::CH_PLATE_CODE[maxidxs[i]];
                score += maxscore[i];
                charnum++;
            }
        }

        // 计算平均得分
        if (charnum > 0)
        {
            score /= charnum;
        }

        // 设置PlateInfo对象的车牌名称和识别得分
        plateinfo.setPlateName(text);
        plateinfo.confidence = score;
    }
}
```

**测试代码**

```
#include "../include/Pipeline.h"

// 测试车牌识别管道的函数
void TEST_PIPELINE()
{
    // 创建车牌识别的管道对象，并传入相关模型文件的路径
    pr::PipelinePR prc("../lpr/model/mininet_ssd_v1.prototxt",
"../lpr/model/mininet_ssd_v1.caffemodel",
```

```
                        "../lpr/model/refinenet.prototxt",
"../lpr/model/refinenet.caffemodel",
                        "../lpr/model/SegmenationFree-Inception.prototxt",
"../lpr/model/SegmenationFree-Inception.caffemodel",
                        "../lpr/model/cascade_double.xml");

    // 读取待识别的图像
    cv::Mat img = cv::imread("../lpr/res/test.jpg");

    // 运行车牌识别管道，得到识别结果
    std::vector<pr::PlateInfo> res = prc.RunPiplineAsImage(img, 0);

    // 遍历识别结果
    for (auto st : res) {
        if (st.confidence > 0.75) {
            // 打印识别出的车牌名称和置信度
            std::cout << st.getPlateName() << " " << st.confidence <<
std::endl;

            // 获取车牌的矩形区域并在图像上绘制矩形框
            cv::Rect region = st.getPlateRect();
            cv::rectangle(img, cv::Point(region.x, region.y),
cv::Point(region.x + region.width, region.y + region.height),
cv::Scalar(255, 255, 0), 2);
        }
    }

    // 显示带有车牌矩形框的图像
    //cv::imshow("image", img);
    //cv::waitKey(0);
}

// 主函数
int main()
{
    // 调用测试函数，运行车牌识别的示例程序
    TEST_PIPELINE();
    return 0;
}
```

这段代码首先创建了一个车牌识别的管道对象`prc`，并传入了相关的模型文件路径。然后，它加载一张待识别的图像`img`。接下来，通过调用`prc.RunPiplineAsImage()`函数运行车牌识别管道，并将识别结果存储在`res`向量中。

在遍历识别结果时，如果识别得分大于0.75，则打印出识别出的车牌名称和置信度，并使用OpenCV的函数`cv::rectangle()`在图像上绘制识别到的车牌矩形框。

最后，代码可以选择显示带有车牌矩形框的图像（注释部分），或直接返回程序结束。整体作用是演示了如何使用车牌识别的管道进行图像中的车牌识别，并对识别结果进行处理和展示。

# vs实现测试程序

使用软件版本：

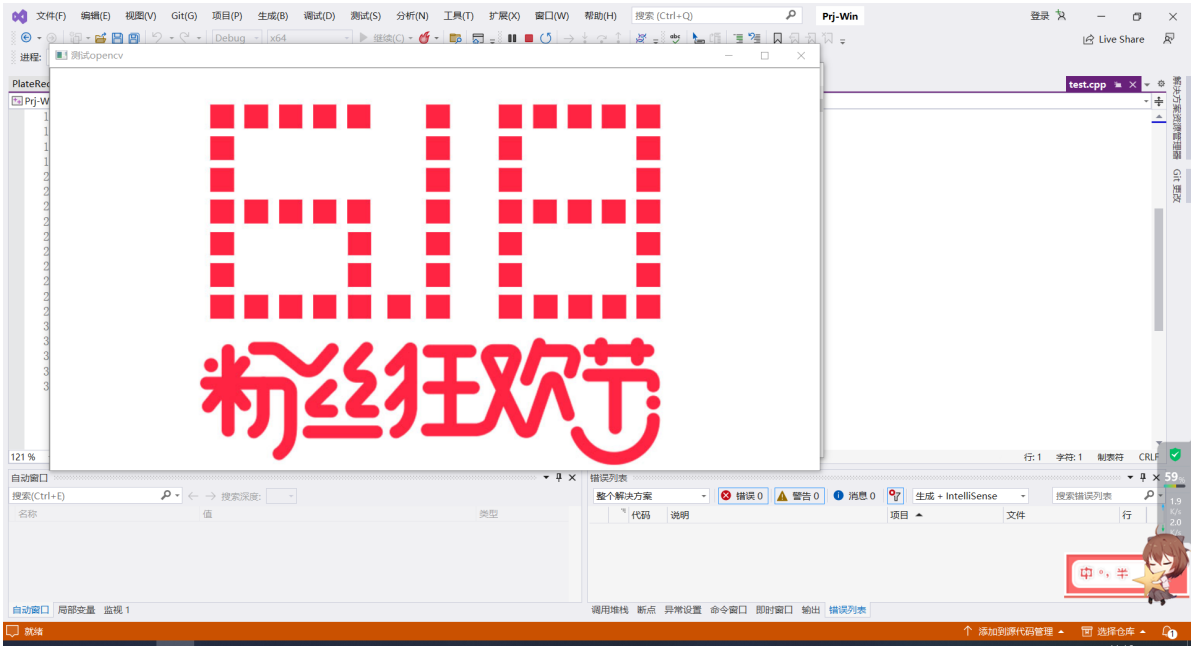
1. Visual Studio 2022
2. opencv-4.00
3. HyperLPR-2

测试opencv：

使用测试代码

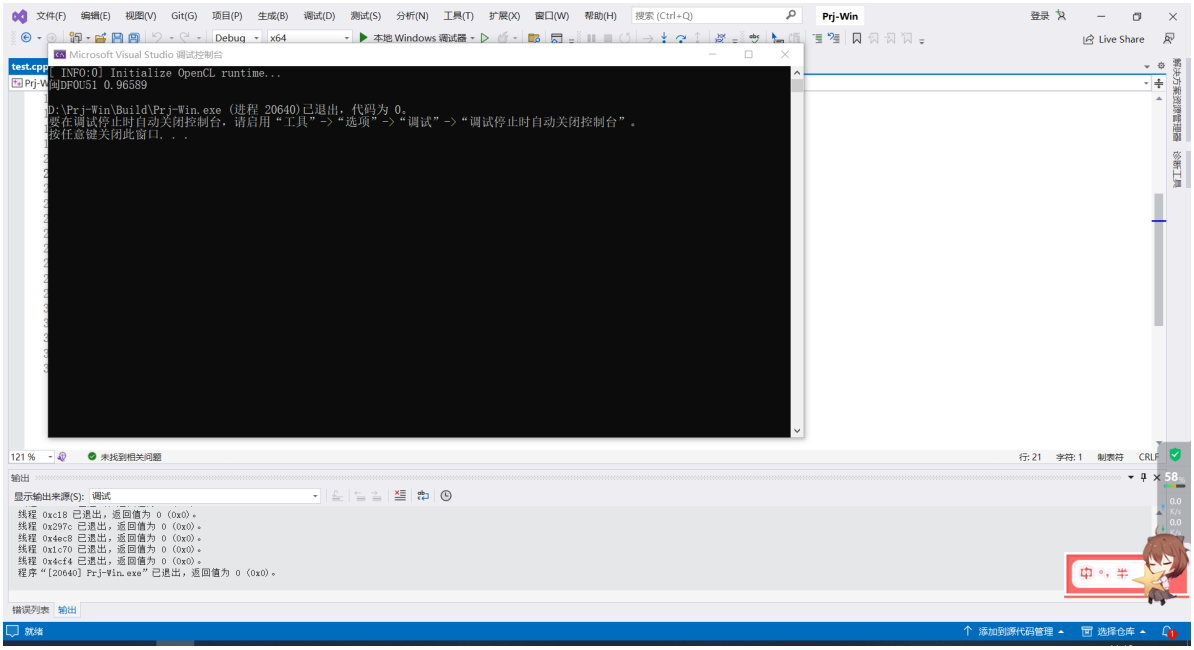
```
//读取图片并显示
#include "stdio.h"
#include<iostream>
#include <opencv2/core/core.hpp>
#include <opencv2/highgui/highgui.hpp>
using namespace cv;
//using namespace std;
int main()
{
    Mat img = imread("E:\\ps\\work\\1\\618.jpg");
    namedWindow("测试opencv");
    imshow("测试opencv", img);
    waitKey(6000);
}
```

结果如下，测试opencv运行成功




测试hyperlpr：

利用上述分析过的测试代码(TEST_PIPELINE)，运行结果如下：




对比测试图片可以看出，运行成功。