

hyperlpr车牌识别分析代码

头文件(.h)

PlateInfo.h

```
1  #ifndef SWIFTPR_PLATEINFO_H
2  #define SWIFTPR_PLATEINFO_H
3
4  #include <opencv2/opencv.hpp>
5
6  namespace pr {
7
8      typedef std::vector<cv::Mat> Character; // 定义Character类型为存储cv::Mat
      的向量
9
10     // 枚举数据, 表示车牌的颜色
11     enum PlateColor { BLUE, YELLOW, WHITE, GREEN, BLACK, UNKNOWN };
12
13     // 枚举数据, 表示车牌类型
14     enum CharType { CHINESE, LETTER, LETTER_NUMS, INVALID };
15
16     class PlateInfo {
17     public:
18         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
19         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
20         float confidence = 0; // 识别得分
21
22         // 构造函数, 根据不同参数设置车牌信息
23         PlateInfo(const cv::Mat& plateData, std::string plateName, cv::Rect
      plateRect, PlateColor plateType)
24             : licensePlate(plateData), name(plateName), ROI(plateRect),
      Type(plateType) {}
25
26         PlateInfo(const cv::Mat& plateData, cv::Rect plateRect, PlateColor
      plateType)
27             : licensePlate(plateData), ROI(plateRect), Type(plateType) {}
28
29         PlateInfo(const cv::Mat& plateData, cv::Rect plateRect)
30             : licensePlate(plateData), ROI(plateRect) {}
31
32         PlateInfo() {}
33
34         cv::Mat getPlateImage() {
35             return licensePlate; // 返回车牌图像
36         }
37
38         void setPlateImage(cv::Mat plateImage) {
39             licensePlate = plateImage; // 设置车牌图像
```

```

40     }
41
42     cv::Rect getPlateRect() {
43         return ROI; // 返回车牌在图像中的矩形区域
44     }
45
46     void setPlateRect(cv::Rect plateRect) {
47         ROI = plateRect; // 设置车牌矩形区域
48     }
49
50     cv::String getPlateName() {
51         return name; // 返回车牌名称
52     }
53
54     void setPlateName(cv::String plateName) {
55         name = plateName; // 设置车牌名称
56     }
57
58     int getPlateType() {
59         return Type; // 返回车牌颜色
60     }
61
62     int setPlateType(PlateColor plateType) {
63         Type = plateType; // 设置车牌颜色
64         return 0;
65     }
66
67     void appendPlateChar(const std::pair<CharType, cv::Mat>& plateChar)
68     {
69         plateChars.push_back(plateChar); // 添加一个车牌字符到plateChars向
70         量
71     }
72
73     void appendPlateCoding(const std::pair<CharType, cv::Mat>&
74     charProb) {
75         plateCoding.push_back(charProb); // 添加一个车牌字符编码到
76         plateCoding向量
77     }
78
79     std::string decodePlateNormal(std::vector<std::string>
80     mappingTable) {
81         std::string decode;
82         for (auto plate : plateCoding) {
83             float* prob = (float*)plate.second.data;
84             if (plate.first == CHINESE) {
85                 decode += mappingTable[std::max_element(prob, prob +
86                 31) - prob]; // 解码中文字符
87                 confidence += *std::max_element(prob, prob + 31);
88             }
89             else if (plate.first == LETTER) {
90                 decode += mappingTable[std::max_element(prob + 41, prob
91                 + 65) - prob]; // 解码字母字符
92                 confidence += *std::max_element(prob + 41, prob + 65);
93             }
94             else if (plate.first == LETTER_NUMS) {

```

```

88         decode += mappingTable[std::max_element(prob + 31, prob
+ 65) - prob]; // 解码字母和数字字符
89         confidence += *std::max_element(prob + 31, prob + 65);
90     }
91     else if (plate.first == INVALID) {
92         decode += '*'; // 无效字符
93     }
94 }
95 name = decode; // 设置车牌名称为解码结果
96
97 confidence /= 7; // 计算平均识别得分
98
99 return decode;
100 }
101
102 private:
103     cv::Mat licensePlate; // 车牌图像
104     cv::Rect ROI; // 车牌在图像中的矩形区域
105     std::string name; // 车牌名称
106     PlateColor type; // 车牌颜色
107 };
108
109 } // namespace pr
110
111 #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

```

1  #ifndef _PLATEDETECT_H_
2  #define _PLATEDETECT_H_
3
4  #include <opencv2/opencv.hpp>
5  #include <vector>
6  #include "PlateInfo.h"
7
8  namespace pr {
9      class PlateDetection {

```

```

10     public:
11         // 构造函数，用于初始化PlateDetection对象
12         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 cascadingstring);
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 `cascadingstring`，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 {
10     public:
11         // 构造函数，用于初始化FineTune对象
12         FineTune(std::string finetune_prototxt, std::string
finetune_caffemodel);
13
14         // 执行微调操作，输入图像img，输出结果图像resImg
15         void Finetune(cv::Mat img, cv::Mat& resImg);
16
17         // 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
18         void to_refine(cv::Mat img, std::vector<cv::Point> pts, cv::Mat&
out);
19
20         // 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
21         void affine_crop(cv::Mat img, std::vector<cv::Point> pts, cv::Mat&
out);
22
23     private:
24         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
25     };
26 } // namespace pr
27
28 #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。

PlateRecognition.h

```

1  #ifndef _PLATERECOGNITION_H_
2  #define _PLATERECOGNITION_H_
3
4  #include <opencv2/dnn.hpp>
5  #include "PlateInfo.h"
6
7  namespace pr {
8      class PlateRecognition {
9      public:
10         // 构造函数，用于初始化PlateRecognition对象

```

```

11     PlateRecognition(std::string Rec_prototxt, std::string
Rec_caffemodel);
12
13     // 进行无分割的车牌识别，输入原始图像src，输出车牌信息plateinfo
14     void segmentation_free_recognition(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 `PlateRecognition` 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。

- `PlateRecognition` 类的构造函数，接受两个参数 `Rec_prototxt` 和 `Rec_caffemodel`，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_recognition` 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 _PIPELINE_H
3  #define _PIPELINE_H
4
5  #include <vector>
6  #include "Finetune.h"
7  #include "Platedetect.h"
8  #include "PlateRecognition.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     PlateRecognition* platerecognition;
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
31     PipelinePR(std::string detect_prototxt, std::string
detect_caffemodel,
32               std::string finetune_prototxt, std::string finetune_caffemodel,
33               std::string platerec_prototxt, std::string platerec_caffemodel,
34               std::string dbstring);
35
36     // 析构函数，用于释放内部对象的资源
37     ~PipelinePR();
38
39     std::vector<std::string> plateRes; // 车牌结果容器
40     std::vector<PlateInfo> RunPipelineAsImage(cv::Mat srcImage, int
IsDB); // 执行车牌识别流程的函数
41 };
42 } // namespace pr
43
44 #endif // !_PIPELINE_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)、`platerecognition` (`PlateRecognition` 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 `PlateRecognition` 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 `RunPipelineAsImage` 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

```

1  #include "../include/Finetune.h"
2  using namespace std;
3  using namespace cv;
4
5  namespace pr {
6      FineTune::FineTune(string finetune_prototxt, string finetune_caffemodel)
7      {
8          // 从给定的 prototxt 和 caffemodel 文件读取深度学习网络
9          FTNet = dnn::readNetFromCaffe(finetune_prototxt,
finetune_caffemodel);
10     }

```

```

10
11     void FineTune::affine_crop(Mat src, vector<Point> pts, Mat &crop)
12     {
13         // 定义源图像上的四个点坐标
14         Point2f dst[4] = {
15             Point2f(0,0),Point2f(160,0),Point2f(160,40),Point2f(0,40) };
16         Point2f srcpt[4] = { Point2f(pts[0]),Point2f(pts[1])
17             ,Point2f(pts[2]) ,Point2f(pts[3]) };
18         // 获取透视变换矩阵
19         Mat _mat = getPerspectiveTransform(srcpt, dst);
20         // 进行透视变换
21         warpPerspective(src, crop, _mat, Size(160, 40));
22     }
23
24     void FineTune::to_refine(Mat src, vector<Point> pts, Mat& crop)
25     {
26         // 定义中心点和宽高
27         float scale = 3.f;
28         int cx = 64; int cy = 24;
29         int cw = 64; int ch = 24;
30         // 定义目标图像上的四个点坐标
31         int tx1 = cx - cw / 2;
32         int ty1 = cy - ch / 2;
33         int tx2 = cx + cw / 2;
34         int ty2 = cy - ch / 2;
35         int tx3 = cx + cw / 2;
36         int ty3 = cy + ch / 2;
37         int tx4 = cx - cw / 2;
38         int ty4 = cy + ch / 2;
39         vector<Point2f> dstp(4);
40         Point2f dst[4] = { (Point2f(tx1*scale, ty1*scale)) ,
41             (Point2f(tx2*scale, ty2*scale)) ,(Point2f(tx3*scale, ty3*scale)) ,
42             (Point2f(tx4*scale, ty4*scale)) };
43         Point2f pt[4] = { Point2f(pts[0]),Point2f(pts[1]) ,Point2f(pts[2])
44             ,Point2f(pts[3]) };
45         // 获取透视变换矩阵
46         Mat _mat = getPerspectiveTransform(pt, dst);
47         // 进行透视变换
48         warpPerspective(src, crop, _mat, Size(120 * scale, 48 * scale));
49     }
50
51     void FineTune::Finetune(Mat src, Mat& dst)
52     {
53         // 将源图像调整为指定大小
54         Mat tof;
55         resize(src, tof, Size(120, 48));
56         // 将图像转换为网络输入的 blob
57         Mat blob = dnn::blobFromImage(tof, 0.0078125, Size(120, 48),
58             Scalar(127.5, 127.5, 127.5), false, false);
59         // 设置网络的输入 blob
60         FTNet.setInput(blob);
61         // 运行网络前向传播, 获取输出 blob
62         Mat outblob = FTNet.forward("conv6-3");
63
64         float *data = outblob.ptr<float>();

```



```

59     vector<Point> pts(4);
60     // 将输出 blob 中的数据转换为四个点的坐标
61     Mat fineMat(Size(2, 4), CV_32F, data);
62     for (int i = 0; i < fineMat.rows; i++)
63     {
64         pts[i].x = fineMat.at<float>(i, 0)*src.cols;
65         pts[i].y = fineMat.at<float>(i, 1)*src.rows;
66     }
67     Mat crop;
68     // 对裁剪的图像进行细化
69     to_refine(src, pts, crop);
70     // 将裁剪后的图像转换为网络输入的 blob
71     blob = dnn::blobFromImage(crop, 0.0078128, Size(120, 48),
Scalar(127.5, 127.5, 127.5), false, false);
72     // 设置网络的输入 blob
73     FTNet.setInput(blob);
74     // 运行网络前向传播, 获取输出 blob
75     outblob = FTNet.forward("conv6-3");
76     data = outblob.ptr<float>();
77     // 将输出 blob 中的数据转换为四个点的坐标
78     Mat fineMat2(Size(2, 4), CV_32F, data);
79     for (int i = 0; i < fineMat.rows; i++)
80     {
81         pts[i].x = fineMat2.at<float>(i, 0)*crop.cols;
82         pts[i].y = fineMat2.at<float>(i, 1)*crop.rows;
83     }
84     // 对裁剪的图像进行仿射变换
85     affine_crop(crop, pts, crop);
86     // 将最终的裁剪图像复制到 dst 中
87     dst = crop.clone();
88 }
89 }

```

Pipeline.cpp

```

1  #include "../include/Pipeline.h"
2
3  namespace pr {
4      PipelinePR::PipelinePR(std::string detect_prototxt, std::string
detect_caffemodel,
5          std::string finetune_prototxt, std::string finetune_caffemodel,
6          std::string platerec_prototxt, std::string platerec_caffemodel,
7          std::string dbstring)
8      {
9          // 创建车牌检测对象
10         platedetection = new PlateDetection(detect_prototxt,
detect_caffemodel);
11         // 创建细化对象
12         finetune = new FineTune(finetune_prototxt, finetune_caffemodel);
13         // 创建车牌识别对象
14         platerecognition = new PlateRecognition(platerec_prototxt,
platerec_caffemodel);
15         // 创建车牌数据库检测对象
16         dbdetection = new DBDetection(dbstring);
17     }

```

```

18
19 PipelinePR::~PipelinePR()
20 {
21     // 释放内存
22     delete platedetection;
23     delete finetune;
24     delete platerecognition;
25     delete dbdetection;
26 }
27
28 cv::Mat DBCropFromImage(const cv::Mat &image){
29     cv::Mat cropped;
30     image.copyTo(cropped);
31     int cropped_w = cropped.cols;
32     int cropped_h = cropped.rows;
33     // 定义上半部分和下半部分的矩形区域
34     cv::Rect up,down;
35     up.y = cropped_h*0.05;up.x = cropped_w*0.2;up.height =
cropped_h*0.35;up.width = cropped_w*0.55;
36     down.y = cropped_h*0.4;down.x = cropped_w*0.05;down.height =
cropped_h*0.6;down.width = cropped_w*0.95;
37     cv::Mat cropUp,cropDown;
38     // 截取上半部分和下半部分的图像
39     cropped(up).copyTo(cropUp);
40     cropped(down).copyTo(cropDown);
41     // 调整图像尺寸
42     cv::resize(cropUp,cropUp,cv::Size(64,40));
43     cv::resize(cropDown,cropDown,cv::Size(96,40));
44     // 创建最终的车牌图像
45     cv::Mat crop = cv::Mat(40,160,CV_8UC3);
46     cropUp.copyTo(crop(cv::Rect(0,0,64,40)));
47     cropDown.copyTo(crop(cv::Rect(64,0,96,40)));
48     return crop;
49 }
50
51 std::vector<PlateInfo> PipelinePR::RunPipelineAsImage(cv::Mat
plateimg,int IsDB)
52 {
53     std::vector<pr::PlateInfo> plates;
54     std::vector<PlateInfo> plateres;
55     if(IsDB==1)
56     {
57         // 调用车牌数据库检测方法
58         dbdetection->DBDetect(plateimg,plates,30,1280);
59     }
60     else
61     {
62         // 调用车牌检测方法
63         platedetection->Detectssd(plateimg, plates);
64     }
65     for (pr::PlateInfo plateinfo : plates) {
66         cv::Mat image = plateinfo.getPlateImage();
67         cv::Mat CropImg;
68
69         if(IsDB==1)

```

```

70         {
71             // 调用车牌数据库识别方法
72             CropImg = DBcropFromImage(image);
73             platerecognition->segmentation_free_recognition(CropImg,
plateinfo);
74         }
75         else
76         {
77             // 调用车牌细化和识别方法
78             finetune->Finetune(image, CropImg);
79             platerecognition->segmentation_free_recognition(CropImg,
plateinfo);
80         }
81         // 将识别结果存入结果集合
82         plateres.push_back(plateinfo);
83     }
84     return plateres;
85 }
86 }

```

PlateDecton.cpp

```

1  #include "../include/Platedetect.h"
2
3  using namespace cv;
4  using namespace std;
5  namespace pr {
6
7      // 车牌检测器构造函数，加载SSD模型
8      PlateDetection::PlateDetection(std::string ssd_prototxt, std::string
ssd_caffemodel)
9      {
10         ssdNet = cv::dnn::readNetFromCaffe(ssd_prototxt, ssd_caffemodel);
11     }
12
13     // 车牌级联分类器构造函数，加载级联分类器模型
14     DBDetection::DBDetection(std::string cascadingstring)
15     {
16         dbcascade.load(cascadingstring);
17     }
18
19     // 使用SSD模型检测车牌
20     void PlateDetection::Detectssd(cv::Mat img, std::vector<pr::PlateInfo>
&plateInfos)
21     {
22         int cols = img.cols;
23         int rows = img.rows;
24
25         // 将图像转换为32位浮点数类型
26         Mat in;
27         img.convertTo(in, CV_32F);
28
29         // 创建输入Blob
30         Mat input(img.size(), CV_32FC3);
31         Mat inputblob1 = input.reshape(1, { 1, 3, rows, cols });

```

```

32     Mat input_blob = dnn::blobFromImages(in, 0.225, size(),
Scalar(103.53, 116.28, 123.675), false);
33     float *blobdata = input_blob.ptr<float>();
34     float *blobdata2 = inputblob1.ptr<float>();
35     {
36         // 将Blob数据拷贝到输入Blob中
37         for (int i = 0; i < rows; i++)
38         {
39             memcpy(blobdata2 + i * cols, blobdata + 3 * i * cols, cols
* sizeof(float));
40             memcpy(blobdata2 + i * cols + rows * cols, blobdata + (1 +
3 * i) * cols, cols * sizeof(float));
41             memcpy(blobdata2 + i * cols + rows * cols * 2, blobdata +
(2 + 3 * i) * cols, cols * sizeof(float));
42         }
43     }
44
45     // 设置输入Blob
46     ssdNet.setInput(inputblob1);
47
48     // 前向传播获取输出Blob
49     Mat outputBlob = ssdNet.forward("detection_out");
50
51     // 解析输出Blob中的检测结果
52     Mat detectmat(outputBlob.size[2], outputBlob.size[3], CV_32F,
outputBlob.ptr<float>());
53     for (int i = 0; i < detectmat.rows; i++)
54     {
55         float confidence = detectmat.at<float>(i, 2);
56
57         // 如果置信度大于0.5, 则认为是一个车牌
58         if (confidence > 0.5)
59         {
60             int x1, x2, y1, y2;
61             Rect rec;
62             Mat cimg;
63             x1 = int(detectmat.at<float>(i, 3) * cols);
64             y1 = int(detectmat.at<float>(i, 4) * rows);
65             x2 = int(detectmat.at<float>(i, 5) * cols);
66             y2 = int(detectmat.at<float>(i, 6) * rows);
67
68             // 对检测框的坐标进行修正, 确保不超出图像边界
69             x1 = max(x1, 0);
70             y1 = max(y1, 0);
71             x2 = min(x2, cols - 1);
72             y2 = min(y2, rows - 1);
73
74             // 提取车牌图像
75             rec.x = x1; rec.y = y1; rec.width = (x2 - x1 + 1);
rec.height = (y2 - y1 + 1);
76             img(rec).copyTo(cimg);
77
78             // 创建PlateInfo对象, 并将车牌图像和位置信息保存到plateInfos向量中
79             PlateInfo plateInfo(cimg, rec);
80             plateInfos.push_back(plateInfo);

```

```

81     }
82 }
83 }
84
85 // 从图像中裁剪指定区域的图像
86 cv::Mat cropFromImage(const cv::Mat &image, cv::Rect rect)
87 {
88     int w = image.cols - 1;
89     int h = image.rows - 1;
90
91     // 确保裁剪区域不超出图像边界
92     rect.x = std::max(rect.x, 0);
93     rect.y = std::max(rect.y, 0);
94     rect.height = std::min(rect.height, h - rect.y);
95     rect.width = std::min(rect.width, w - rect.x);
96
97     // 裁剪图像并返回
98     cv::Mat temp(rect.size(), image.type());
99     cv::Mat cropped;
100     temp = image(rect);
101     temp.copyTo(cropped);
102     return cropped;
103 }
104
105 // 使用级联分类器模型检测车牌
106 void DBDetection::DBDetect(cv::Mat InputImage,
std::vector<pr::PlateInfo> &plateInfos, int min_w, int max_w)
107 {
108     cv::Mat processImage;
109     cv::cvtColor(InputImage, processImage, cv::COLOR_BGR2GRAY);
110     std::vector<cv::Rect> platesRegions;
111
112     // 设置车牌的最小和最大尺寸
113     cv::Size minSize(min_w, min_w / 4);
114     cv::Size maxSize(max_w, max_w / 4);
115
116     // 如果处理的图像为空，则直接返回
117     if (&processImage == NULL)
118         return;
119
120     // 使用级联分类器模型检测车牌
121     dbccascade.detectMultiscale(processImage, platesRegions,
122         1.1, 3, cv::CASCADE_SCALE_IMAGE, minSize, maxSize);
123
124     // 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
125     for (auto plate : platesRegions)
126     {
127         int zeroadd_w = static_cast<int>(plate.width * 0.28);
128         int zeroadd_h = static_cast<int>(plate.height * 0.35);
129         int zeroadd_x = static_cast<int>(plate.width * 0.14);
130         int zeroadd_y = static_cast<int>(plate.height * 0.15);
131
132         // 对车牌区域进行修正
133         plate.x -= zeroadd_x;
134         plate.y -= zeroadd_y;

```

```

135         plate.height += zeroadd_h;
136         plate.width += zeroadd_w;
137
138         // 从原始图像中裁剪出车牌图像
139         cv::Mat plateImage = cropFromImage(InputImage, plate);
140
141         // 创建PlateInfo对象，并将车牌图像和位置信息保存到plateInfos向量中
142         PlateInfo plateInfo(plateImage, plate);
143         plateInfos.push_back(plateInfo);
144     }
145 }
146 }

```

PlateRecognition.cpp

```

1  #include "../include/PlateRecognition.h"
2  #include "../include/Pipeline.h"
3  using namespace std;
4  using namespace cv;
5
6  namespace pr {
7
8      // 构造函数：初始化识别网络
9      PlateRecognition::PlateRecognition(std::string rec_prototxt, std::string
rec_caffemodel)
10     {
11         RecNet = cv::dnn::readNetFromCaffe(rec_prototxt, rec_caffemodel);
12     }
13
14     // 车牌字符分割自由识别函数
15     void PlateRecognition::segmentation_free_recognition(cv::Mat src,
pr::PlateInfo &plateinfo)
16     {
17         float score = 0; // 识别得分
18         string text = ""; // 识别结果
19         Mat src1 = src.clone(); // 克隆原始图像
20         Mat inputMat(Size(40, 160), CV_8UC3); // 创建输入图像的尺寸
21
22         // 将原始图像数据复制到输入图像中
23         for (int j = 0; j < src.rows; j++)
24         {
25             for (int i = 0; i < src.cols; i++)
26             {
27                 inputMat.at<Vec3b>(i, j) = src1.at<Vec3b>(j, i);
28             }
29         }
30
31         // 将输入图像转换为网络可接受的格式
32         Mat blob = dnn::blobFromImage(inputMat, 1 / 255.f, Size(40, 160),
scalar(0, 0, 0), false, false);
33
34         // 设置网络的输入
35         RecNet.setInput(blob);
36
37         // 进行前向传播，获取网络输出

```

```

38     Mat outblob = RecNet.forward();
39
40     int x = outblob.size[2];
41     int y = outblob.size[0];
42
43     float *data = outblob.ptr<float>();
44
45     vector<float> scores(84); // 存储字符的分数
46     vector<int> maxidxs; // 存储每个字符的最大分数索引
47     vector<float> maxscore; // 存储每个字符的最大分数
48
49     // 遍历网络输出，获取每个字符的分数和最大分数索引
50     for (int i = 2; i < 20; i++)
51     {
52         for (int j = 0; j < 84; j++)
53         {
54             scores[j] = data[j * 20 + i];
55         }
56         int idx = max_element(scores.begin(), scores.end()) -
scores.begin();
57         maxidxs.push_back(idx);
58         maxscore.push_back(scores[idx]);
59     }
60
61     int charnum = 0; // 字符数量
62
63     // 将最大分数索引转换为字符，并计算总体得分
64     for (int i = 0; i < maxidxs.size(); i++)
65     {
66         if (maxidxs[i] < pr::CH_PLATE_CODE.size() && (i == 0 ||
(maxidxs[i - 1] != maxidxs[i])))
67         {
68             text += pr::CH_PLATE_CODE[maxidxs[i]];
69             score += maxscore[i];
70             charnum++;
71         }
72     }
73
74     // 计算平均得分
75     if (charnum > 0)
76     {
77         score /= charnum;
78     }
79
80     // 设置PlateInfo对象的车牌名称和识别得分
81     plateinfo.setPlateName(text);
82     plateinfo.confidence = score;
83 }
84 }

```

测试代码

```
1 #include "../include/Pipeline.h"
2
3 // 测试车牌识别管道的函数
4 void TEST_PIPELINE()
5 {
6     // 创建车牌识别的管道对象，并传入相关模型文件的路径
7     pr::PipelinePR prc("../lpr/model/mininet_ssd_v1.prototxt",
8     "../lpr/model/mininet_ssd_v1.caffemodel",
9     "../lpr/model/refinenet.prototxt",
10    "../lpr/model/refinenet.caffemodel",
11    "../lpr/model/SegmenationFree-Inception.prototxt",
12    "../lpr/model/SegmenationFree-Inception.caffemodel",
13    "../lpr/model/cascade_double.xml");
14
15    // 读取待识别的图像
16    cv::Mat img = cv::imread("../lpr/res/test.jpg");
17
18    // 运行车牌识别管道，得到识别结果
19    std::vector<pr::PlateInfo> res = prc.RunPipelineAsImage(img, 0);
20
21    // 遍历识别结果
22    for (auto st : res) {
23        if (st.confidence > 0.75) {
24            // 打印识别出的车牌名称和置信度
25            std::cout << st.getPlateName() << " " << st.confidence <<
26            std::endl;
27
28            // 获取车牌的矩形区域并在图像上绘制矩形框
29            cv::Rect region = st.getPlateRect();
30            cv::rectangle(img, cv::Point(region.x, region.y),
31            cv::Point(region.x + region.width, region.y + region.height),
32            cv::Scalar(255, 255, 0), 2);
33        }
34    }
35
36    // 显示带有车牌矩形框的图像
37    //cv::imshow("image", img);
38    //cv::waitKey(0);
39 }
40
41 // 主函数
42 int main()
43 {
44     // 调用测试函数，运行车牌识别的示例程序
45     TEST_PIPELINE();
46     return 0;
47 }
```

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.RunPipelineAsImage()` 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。