# 极市算法SDK说明文档V1.0（C/C++)

| 版本 | 内容说明 | 编写人 |
| --- | --- | --- |
| V1.0 | 提交至极市的算法Linux SDK规范的说明 | 杜鑫 |

# 文档目录



# 1.功能说明

本版本仅适用于C/C++语言开发的算法版本，SDK实现的功能是让接口集成调用规范化，上线极市平台方便用户体验。

# 2.算法SDK包含文件（必须）

- **ji.h** # SDK接口头文件，规范了**初始化，输入，输出，图片分析及视频分析**接口，具体下面2.1说明
- **ji.cpp** # SDK接口实现的文件，实现ji.h 中的所有接口，未使用或未实现的接口在.cpp文件中统一返回int型-2,使用的接口统一返回 0或者-1

- **ji.so** # SDK动态库文件
- **demo.cpp** # 调用ji.cpp实现的接口

## 2.1 ji.h 接口说明

### 2.1.1初始化

```
/*
    可选项，初始化全局参数（例如log4cpp等），LICENSE（如果有），主程序传参，可不使用
    输入参数：
          主程序如果有参数，从初始化函数传入。
    返回:
          0:初始化成功。
          -2：没有使用
          others ： 初始化失败 。
*/
int ji_init(int argc, char** argv);
```

### 2.1.2 创建和释放检测器

检测器是一个C++类，后面的图片或者视频分析接口会以检测器作为输入参数进行图片或者视频的分析。在**ji.cpp**中必须实现下述两个函数**ji_create_predictor()** 和**ji_destory_predictor(void* predictor)**

#### 2.1.2.1 创建检测器

```
/*
    创建一个检测器实例，作为后面图片分析接口和视频分析接口的输入参数
    输入参数：
          无。
    返回：
          实例初始化，即返回检测器的类。
*/

void *ji_create_predictor();
```

#### 2.1.2.2 释放检测器

```
/*
    释放实例函数。
    输入参数：
        predictor ： 待释放的实例。
    返回：
        0 ： 释放成功。
        1 ： 释放失败。
*/
int ji_destory_predictor(void* predictor);
```

### 2.1.3 分析图片/视频

算法分析包括**图片buffer、图片文件、视频、视频单帧**四种类型接口， 开发者需按照算法功能实现相关的接口，并自行实现接口调用的demo演示,方便测试人员进行接口性能测试，输出格式统一为JSON格式，JSON输出格式建议使用cJSON库标准输出，输出格式验证可到bejson进行验证。

**注意**：

- 四个分析接口都需要体现在**ji.cpp**中，未使用或未实现的接口在.cpp文件中统一返回int型-2，使用的接口统一返回 0或者-1。
- 图片、视频单帧接口输入一次返回一次结果（图片、单帧文件保存路径和json)；视频接口输入视频实时返回视频结果保存在本地文件夹,分析完成,根据返回结果长度实例化json,打印所有帧json信息,然后释放json

#### 2.1.3.1图片分析

图片分析接口需要同时实现**图片buffer分析（ji_calc）、图片文件分析（ji_calc_file）**两个接口。

- **分析图片Buffer**

```
/*
 传入图片Buffer作分析，Buffer是由传入的图片转成的二进制流

 输入参数：
   predictor：检测器实例
   buffer: 输入图片文件Buffer,统一用C++标准库进行图片到二进制流转换,转换可参考上面面示例代码，请勿用opencv等其它图像处理库函数对图片进行转换（自行管理图片缓存释放）
   length: 输入图片Buffer长度
   args: 可选项,如图片大小，图片感兴趣区域等绘制
   outfn: 输出文件名称（自行管理图片缓存释放）
   json: 分析图片输出Json信息,主函数中释放JSON

 返回：
 0:分析成功
 -2: 没有使用
 others ： 分析失败
 */
```

```
int ji_calc(void* predictor, const unsigned char* buffer, int length,
        const char* args, const char* outfn, char** json);
```

**参考代码**：[如何把图片文件转换为buffer输入并做测试]

- **分析图片文件**

```
/*
传入图片文件进行分析

 输入参数:
   predictor：检测器实例
   infn: 输入图片文件名称
   args: 可选项，视频感兴趣区域的绘制
   outfn: 输出文件名称
   json: 分析图片输出Json信息，json输出格式见下面

 返回 :
   0:分析成功
   -2:没有使用
   others :分析失败
 */
int ji_calc_file(void* predictor, const char* infn, const char* args,
        const char* outfn, char** json);
```

#### 2.1.3.2 视频分析接口

视频分析接口需要同时实现**视频分析（ji_calc_video_file）**和**分析视频单帧（ji_calc_video_frame）**两个接口。

- **分析视频文件**

```
/*
 传入一个视频文件作分析

 输入参数：
 predictor:在ji_create_predictor创建返回的检测器实例类
 infn: 输入视频地址
 args: 可选项，视频等的绘制
 outfn: 输出视频地址
 event: 分析视频Json信息

 返回：
 0: 分析成功
```

```
 -2: 接口没有被使用
 others: 分析失败
 */
int ji_calc_video_file(void* predictor, const char* infn, const char* args,const char*
outfn, JI_EVENT* event);
```

- **分析视频单帧**

```
/*
分析传入视频文件的每一帧

输入参数：
   predictor: 在ji_create_predictor创建返回的检测器实例类
   inframe: 输入单帧 （自行管理帧释放）
   args: 可选项，单帧分析感兴趣区域的绘制等
   outframe: 输出单帧 （自行管理帧释放）
   event: 分析视频Json信息

 返回：
   0: 分析成功
   -2: 没有使用接口
   others: 分析失败
 */
int ji_calc_video_frame(void* predictor, JI_CV_FRAME* inframe, const char*
args,JI_CV_FRAME* outframe, JI_EVENT* event);
```

### 2.1.4 ji.h完整代码示例

```
#ifndef JI_H
#define JI_H

/*
 * Notes:
 * 1. If use init funciton, no need to release library or model.
 * 2. Please release frame inside the functions(ji_calc_video_frame).
 * 3. Please release filebuffer inside the functions(ji_calc).
 */

/*
 *  算法分析分为图片、视频、视频单帧三种类型接口，请开发者根据自己算法功能选择接口进行实现，并自行实现接口
调用的demo演示，方便测试人员进行接口性能测试
 *  接口文件.h文件无需修改，未使用或未实现的接口在.cpp文件中，并统一返回int型-2，使用的接口统一返回 0或
者-1
 *  接口实现包括ji_init（插件、license初始化），ji_create_predictor（创建检测器实
例），ji_destory_predictor(释放检测器实例)
```

```
 *  ji_calc(图片buffer分析接口),ji_calc_file(图片文件分析接口),ji_calc_video_file(视频分析接口),ji_calc_video_frame(视频单帧分析接口)
 *  图片、视频单帧接口输入一次返回一次结果(图片、单帧文件保存路径和json);视频接口输入视频实时返回视频结果保存在本地文件夹,分析完成,根据返回结果长度实例化json,打印所有帧json信息,然后释放json
 */

extern "C" {



/*
 refer to cv::Mat definition
 @单帧分析输入帧定义
 */
typedef struct {
    int rows; //Number of rows in a 2D array.
    int cols; //Number of columns in a 2D array.
    int type; //Array type. Use CV_8UC1, ..., CV_64FC4 to create 1-4 channel matrices, or CV_8UC(n), ..., CV_64FC(n) to create multi-channel (up to CV_CN_MAX channels) matrices.
    void * data; //Pointer to the user data.
    int step; //Number of bytes each matrix row occupies.
} JI_CV_FRAME;


/*
 event defination
 @算法检测器分析结果输出,根据算法实际使用场景由极视角相关工作人员定义
 */
typedef struct {
    int code; //event code
    char* json; //event
} JI_EVENT;



/*
 @可选项,初始化应用级插件,例如:log4cpp license等
 return:
 0:success
 -2: not used
 others : fail
 */
int ji_init(int argc, char** argv);



/*
 @创建检测器实例
 creat instance and initiallized that can be used to analysis
 */
void *ji_create_predictor();
```

```
/*
 @释放检测器实例
 destory instance
 */
void ji_destory_predictor(void* predictor);


/*
 @分析图片Buffer
 analysis image buffer
 parameters:
 @predictor：检测器实例
 @buffer: 输入图片文件Buffer,统一用C++标准库进行图片到二进制流转换,自行参考
ji_file2buffer(imgfile,buffer)函数,请勿用opencv等其它图像处理库函数对图片进行转换（自行管理图片缓
存释放）
 @length: 输入图片Buffer长度
 @args: 可选项,图片感兴趣区域等绘制
 @outfn: 输出文件名称（自行管理图片缓存释放）
 @json: 分析图片输出Json信息,主函数中释放JSON
 return :
 0:success
 -2: not used
 others : fail
 */
int ji_calc(void* predictor, const unsigned char* buffer, int length,
        const char* args, const char* outfn, char** json);


/*
 @分析图片文件
 analysis image
 parameters:
 @predictor：检测器实例
 @infn: 输入图片文件名称
 @args: 可选项,视频感兴趣区域的绘制
 @outfn: 输出文件名称
 @json: 分析图片输出Json信息,,主函数中释放JSON
 @调用此接口
 @return :
 0:success
 -2: not used
 others : fail
 */
int ji_calc_file(void* predictor, const char* infn, const char* args,
        const char* outfn, char** json);


/*
 analysis video file
 @predictor: 检测器实例
 @infn: 输入视频地址
 @args: 可选项,图片感兴趣区域等的绘制
 @outfn: 输出视频地址
```

```
 @event: 分析视频Json信息,主函数中释放Json
 @return：
 0: success
 -2: not used
 others: fail
 */
int ji_calc_video_file(void* predictor, const char* infn, const char* args,
        const char* outfn, JI_EVENT* event);


/*
 analysis video frame
 @predictor: 检测器实例
 @inframe: 输入单帧 （自行管理帧释放）
 @args: 可选项,单帧分析感兴趣区域的绘制等
 @outframe: 输出单帧 （自行管理帧释放）
 @event: 分析视频Json信息,主函数中释放Json
 @return：
 0: success
 -2: not used
 others: fail
 */
int ji_calc_video_frame(void* predictor, JI_CV_FRAME* inframe, const char* args,
        JI_CV_FRAME* outframe, JI_EVENT* event);

}
#endif
```

## 2.2 ji.cpp 实现示例

```
#include <stdio.h>
#include <string.h>
#include <iostream>
#include <vector>
#include <cstdio>
#include <cstdlib>
#include <fstream>
#include <sstream>
#include <string>
#include "ji.h"
using namespace std;

int ji_init(int argc, char** argv) {
    return -2;
}

void *ji_create_predictor() {

}
```

```
void ji_destory_predictor(void* predictor) {

}

int ji_calc(void* predictor, const unsigned char* buffer, int length,
        const char* args, const char* outfn, char** json) {
    return -2;
}

int ji_calc_file(void* predictor, const char* infn, const char* args,
        const char* outfn, char** json) {
    return -2;
}

int ji_calc_video_file(void* predictor, const char* infn, const char* args,
        const char* outfn, JI_EVENT* event) {
    return -2;
}

int ji_calc_video_frame(void* predictor, JI_CV_FRAME* inframe, const char* args,
        JI_CV_FRAME* outframe, JI_EVENT* event) {
    return -2;
}
```

## 2.3 输出Json说明及样例

SDK运行输出的结果必须采用Json输出形式，且输出内容中不允许存在中文。JSON输出格式建议使用cJSON库标准输出，输出格式验证可到bejson进行验证。下图为示例输出json：

### 2.3.1 图像处理类算法Json实例

```
{
"info":{
    "numOfStyle":"",
    "style":[
    {
    "id":"",
    "top":"1",
    "prob":""
    },
    {
    "id":"",
    "top":"2",
    "prob":""
    },
    {
    "id":"",
    "top":"3",
```

```
    "prob":""
    }
    ]
},
    "status": "#分类图片,分类成功0 分类异常-1#",
    "message": "如果status = 0, 输出#Imagefile classify successed# 如果status = -1, 输出
#Imagefile classify failed#"
}

//
服装风格图片输出json规范示例
info 服装分类信息
    numOfStyle //输出置信度较高风格数量
    style//按照风格数量以及置信度排列输出top信息
    id//所属服装类别号
    top//按照置信度大小 输出相应的top值排序比如最高的top 相应的序号为1
    prob//属于该类别的置信概率值

经典--“1”
时尚--“2”
可爱--“3”
成熟--“4”
```

### 2.3.2 视频分析类算法Json实例

```
{
    "timeStamp": "#当前检测视频时间戳（精确到ms）#",
    "numOfPeople": "#分析区域检测到的人的数量#",
    "status": "#当前检测状态 检测到人1 未检测到人0#",
    "message": "#检测到人message为 #Some People In The Area##未检测到人message为#Nobody In
The Area#",
    "numOfHead":"#检测到的人头数量#",
    "headInfo": [{
        "x": 871,
        "y": 531,
        "width": 38,
        "height": 30,
        "prob": "#检测到为人的概率值#"
    },
    {
        "x": 755,
        "y": 737,
        "width": 48,
        "height": 44,
        "prob": "#检测到为人的概率值#"
    }
]
}
//视频检测人流的Json输出实例
```

```
headInfo:头部坐标


{
    "timeStamp": "置为0#",
    "numOfPeople": "#分析区域检测到的人的数量#",
    "status": "#当前检测状态 检测到人1 未检测到人0#",
    "message": "#检测到人message为 #Some People In The Area##未检测到人message为#Nobody In
The Area#",
    "numOfHead":"#检测到的人头数量#",
    "headInfo": [{
        "x": 871,
        "y": 531,
        "width": 38,
        "height": 30,
        "prob": "#检测到为人的概率值#"
    },
    {
        "x": 755,
        "y": 737,
        "width": 48,
        "height": 44,
        "prob": "#检测到为人的概率值#"
    }
]
}
//视频单帧检测人流的Json输出实例
```

## 2.4 常见错误

- **错误一**.设定输出为常量，输出必须设置为变量，实际输出文件大小不受限制

```
/*
错误写法
*/
char * json = NULL;//new char[204800];
    //memset(json, 0, 204800);
```

- **错误二**.没有实现ji.h中规范的全部接口函数。

```
  /*
  ji.cpp中要求实现的接口函数包括以下七个函数，未使用或未实现的接口在.cpp文件中统一返回int型-2,使用的
接口统一返回 0或者-1
  */

 int ji_init(int argc, char** argv);
```

```
 void *ji_create_predictor()
 void ji_destory_predictor(void* predictor)

 int ji_calc(void* predictor, const unsigned char* buffer, int length,
 const char* args, const char* outfn, char** json)

int ji_calc_file(void* predictor, const char* infn, const char* args,
const char* outfn, char** json)

int ji_calc_video_file(void* predictor, const char* infn, const char* args,const char*
outfn, JI_EVENT* event)

int ji_calc_video_frame(void* predictor, JI_CV_FRAME* inframe, const char*
args,JI_CV_FRAME* outframe, JI_EVENT* event)
```

- **错误三**. 分析图片buffer接口中传入的文件不是Buffer文件，使用了opencv等其它图像处理库函数对图片进行转换

```
  /*
  错误样例
*/
imencode(".png",img encode,data encode)
std::strubg str_encode(data_encode.begin(),data_encode.end())
```

- **其他错误**（待更新）

## 2.5 算法具体实例

### 2.5.1.1 ji-image.cpp，服装图片风格识别的算法实现实例

```
#include <stdio.h>
#include <string.h>
#include <iostream>
#include <vector>
#include <cstdio>
#include <cstdlib>
#include <fstream>
#include <sstream>
#include <string>
#include "ji.h"
using namespace std;
#include "ClothesPredicator.h"
```

```
int ji_init(int argc, char** argv) {
    return -2;
}

void *ji_create_predictor() {
    //detector init
    char bufferPath[100];
    getcwd(bufferPath, 100);
    const string basePath = bufferPath;
    const string dmodel_file = basePath + "/model/deploy.prototxt";
    const string dweights_file = basePath + "/model/_iter_120000.caffemodel";
    const string dmean_file = "";

    ModelFile detectorModelFile(dmodel_file, dweights_file, dmean_file, "", "");

    //classifier init
    const string model_file0 = basePath
            + "/model/ResNet_50_train_val20.prototxt";
    const string trained_file0 = basePath
            + "/model/ResNet_50_train_val23_iter_500000.caffemodel";
    const string mean_file0 = basePath + "/model/imagenet_mean.binaryproto";
    const string label_file0 = basePath + "/model/cloth_shape.txt";
    const string label_file2 = basePath + "/model/cloth_type.txt";

    ModelFile classifierModelFile(model_file0, trained_file0, mean_file0,
            label_file0, label_file2);
    std::vector<ModelFile> modelFile;
    modelFile.push_back(detectorModelFile);
    modelFile.push_back(classifierModelFile);
    //predictor init
    ClothesPredicator *clothesPredicator = new ClothesPredicator(modelFile);
    return clothesPredicator;

}

void ji_destory_predictor(void* predictor) {
    ClothesPredicator *predictor_ = (ClothesPredicator *) predictor;
    delete predictor_;
}

int ji_calc(void* predictor, const unsigned char* buffer, int length,
        const char* args, const char* outfn, char** json) {

    ClothesPredicator *clothesPredicator = (ClothesPredicator *) predictor;
    std::vector<uchar> bufferSet;
    for (int i = 0; i < length; i++) {
        bufferSet.push_back(buffer[i]);
    }
    Json::Value temp;
    clothesPredicator->analysisImgBuffer(bufferSet, length, args, outfn, temp);
```

```
    if (!temp.empty()) {
        char *jsonTemp = new char[temp.toStyledString().size()+1];
        strcpy(jsonTemp, temp.toStyledString().c_str());
        *json = jsonTemp;
    } else {
        return -1;
    }

    return 0;
}

int ji_calc_file(void* predictor, const char* infn, const char* args,
        const char* outfn, char** json) {
    ClothesPredicator *clothesPredicator = (ClothesPredicator *) predictor;
    Json::Value temp;
    clothesPredicator->analysisImgfile(infn, args, outfn, temp);
    if (!temp.empty()) {
        char *jsonTemp = new char[temp.toStyledString().size()+1];
        strcpy(jsonTemp, temp.toStyledString().c_str());
        *json = jsonTemp;
    } else {
        return -1;
    }

    return 0;

}

int ji_calc_video_file(void* predictor, const char* infn, const char* args,
        const char* outfn, JI_EVENT* event) {
    return -2;
}

int ji_calc_video_frame(void* predictor, JI_CV_FRAME* inframe, const char* args,
        JI_CV_FRAME* outframe, JI_EVENT* event) {
    return -2;
}
```

### 2.5.1.2 main-image.cpp,服装图片风格识别demo实现

```
//#include "ji.hpp"
#include "./include/ji_util.hpp"
#include "ji.h"
#include <string.h>
#include <sstream>
#include <glog/logging.h>
using namespace std;
int main(int argc, char* argv[]) {
    /* init model  */
    //image buffer analysis
```

```
    if (argc < 4) {
        std::cout << "Parameter number not right" <<std::endl;
        return -1;
    }
    google::InitGoogleLogging("ev");
    void * predictor = ji_create_predictor();

    char *json = NULL;
    if (std::string(argv[3]) == "1") {
        vector<unsigned char> buffer;
        ji_file2buffer1(argv[1], &buffer);
        while(1){
        ji_calc(predictor, (const unsigned char*) &buffer[0], buffer.size(), "",
                argv[2], &json);
        cout << json << endl;
        delete[] json;
        }
        //image file analysis
    } else if (std::string(argv[3]) == "0") {
        while(1){
        ji_calc_file(predictor, argv[1], "", argv[2], &json);
        cout << json << endl;
        delete[] json;
        }
    } else {
        std::cout << "Error: Choose Interface Wrong" << std::endl;
    }
    ji_destory_predictor(predictor);
    google::ShutdownGoogleLogging();
    return 0;
}
```

#### 2.5.2.1 ji-video.cpp，视频实时检测人员不在岗的算法实现实例

```
#include <stdio.h>
#include <string.h>
#include <iostream>
#include <vector>
#include <cstdio>
#include <cstdlib>
#include <fstream>
#include <sstream>
#include <string>
#include "ji.h"
#include "OffDutyPredicator.h"
using namespace std;
```

```
int ji_init()
{

    return -2;
}


void *ji_create_predictor(){
    std::string evhead_yolo_model_ =
"/home/dx/Desktop/OffDutyV0.2/model/evhead_final.weights";
    std::string evhead_yolo_config_ =
"/home/dx/Desktop/OffDutyV0.2/model/evhead_test.cfg";
    std::string meanPath = "";
    OffDutyPredicator *predictor = new OffDutyPredicator(evhead_yolo_model_,
evhead_yolo_config_, meanPath);
    return predictor;

}

void ji_destory_predictor(void* predictor){
    OffDutyPredicator *predictor_ = (OffDutyPredicator *)predictor;
    delete predictor_;

}

int ji_calc(void* predictor, const unsigned char* buffer, int length,
        const char* args, const char* outfn, char** json) {

    return -2;
}

int ji_calc_file(void* predictor, const char* infn, const char* args,
        const char* outfn, char** json) {

    return -2;
}


int ji_calc_video_file(void* predictor, const char* infn, const char* args, const char*
outfn, JI_EVENT* event){
    OffDutyPredicator *ooffduty = (OffDutyPredicator *)predictor;
    Json::Value temp;
    //Stepover->runByVideo(infn, args, outfn,temp);
    ooffduty->runByVideo(infn, args, outfn,temp);

    if(!temp.empty()){
        event->json = new char[(temp.toStyledString()).size()+1];
        strcpy( event->json,temp.toStyledString().c_str());
        event->json[temp.toStyledString().size()] = '\0';
        event->code = 0;
    }
    else{
        event->code = -1;
```

```
    }

//    std::ofstream outfile;
//    outfile.open("/home/fan/config");
//    std::ostringstream messages;
//    messages<<temp.toStyledString()<<std::endl;
//    outfile<<messages.str()<<std::endl;
//    outfile.close();
    return 0;

}

int ji_calc_video_frame(void* predictor, JI_CV_FRAME* inframe, const char* args,
JI_CV_FRAME* outframe, JI_EVENT* event){
    OffDutyPredicator *ooffduty = (OffDutyPredicator *)predictor;

    cv::Mat inmat(inframe->rows,inframe->cols,inframe->type,inframe->data,inframe-
>step);
    cv::Mat outmat ;
    Json::Value temp;
    ooffduty->runByVideoFrame(inmat,args,outmat,temp);

    if(!temp.empty()){
        event->json = new char[temp.toStyledString().size()+1];
        strcpy(event->json,temp.toStyledString().c_str());
        event->json[temp.toStyledString().size()] = '\0';
        event->code = 0;
        outframe->cols = outmat.cols;
        outframe->rows = outmat.rows;
        outframe->type = outmat.type();
        outframe->data = new char[outmat.total() * outmat.elemSize()];
        memcpy(outframe->data,outmat.data,outmat.total() * outmat.elemSize());
        //outframe->data = outmat.data;
        outframe->step = outmat.step;
    } else{
        event->code = -1;
    }
    return 0;
}
```

### 2.5.2.2 main-video.cpp，视频监测人员不在岗算法demo.cpp实例

```
#include <iostream>
#include "src/ji.h"
#include "src/OffDutyPredicator.h"
int main(int argc, char *argv[]) {
    google::InitGoogleLogging("ev");
    FLAGS_stderrthreshold = google::ERROR;
//    const char *videoFile = "/home/dx/Desktop/OffDutyV0.1/data/test02.mp4";
//    const char *videoFileOutput = "/home/dx/Desktop/OffDutyV0.1/dest/out.flv";
```

```
    if(argc < 4){
        std::cout<<"Warning : parameter number not right"<<std::endl;
        return -1;
    }

    const char *videoFile = argv[1];
    const char *videoFileOutput = argv[2];
    const char *roiErea = "POLYGON((0.1015625 0.10555556,0.9 0.10555556,0.9 0.9,0.1
0.9,0.1 0.1))";

    //TEST FRAME
    if(std::string(argv[3]) == "1") {
        cv::VideoCapture capture;
        capture.open(videoFile);
        cv::VideoWriter writer;
        bool inited = false;
        void * predictor = ji_create_predictor();
        while(true){
            JI_EVENT event;
            event.json = NULL;
            cv::Mat frame;
            capture>>frame;
            if(frame.empty()) break;
            JI_CV_FRAME inf,outf;
            inf.rows = frame.rows;
            inf.cols = frame.cols;
            inf.step = frame.step;
            inf.data = frame.data;
            inf.type = frame.type();
            ji_calc_video_frame(predictor,&inf,roiErea,&outf,&event);

            cv::Mat input__(inf.rows,inf.cols,inf.type,inf.data,inf.step);
            cv::Mat output__(outf.rows,outf.cols,outf.type,outf.data,outf.step);
            if(!inited){
                inited = true;
                writer.open(videoFileOutput,CV_FOURCC('F', 'L', 'V', '1'),
                            capture.get(cv::CAP_PROP_FPS),
                            cv::Size(output__.cols,
                                     output__.rows));
            }
//            cv::imshow("input__",input__);
//            cv::waitKey(1);
//            cv::imshow("output__",output__);
//            cv::waitKey(1);
            writer<<output__;
            if(event.json !=NULL) {
                std::cout << event.json << std::endl;
                delete [] event.json;
            }
            delete [] (uchar*)outf.data;
        }
        writer.release();
        ji_destory_predictor(predictor);
```

```
    }

        //TEST VIDEO
    else if(std::string(argv[3]) == "0") {
        JI_EVENT event;
        event.json = NULL;
        void *predictor = ji_create_predictor();
        ji_calc_video_file(predictor, videoFile, roiErea, videoFileOutput, &event);
        std::cout << event.json << std::endl;
        delete event.json;
        event.json = NULL;
        ji_destory_predictor(predictor);
    }

    else{
        std::cout<< "Warning : parameter wrong"<<std::endl;
    }


    google::ShutdownGoogleLogging();
}
```

## 3.版权说明