OASIS Programing Guide

NXP SOLUTIONS

SECURE CONNECTIONS
FOR A SMARTER WORLD

Table of Contents

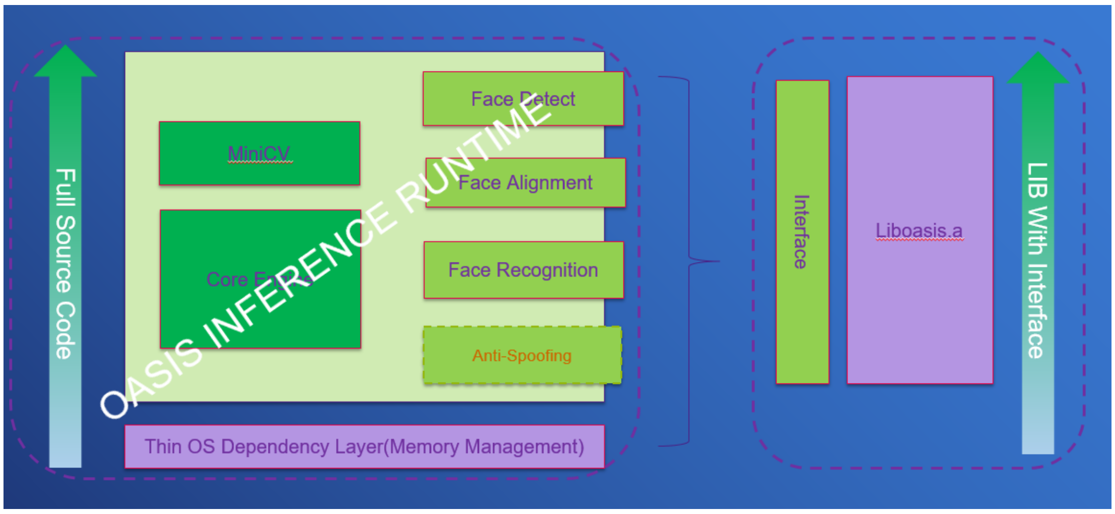
1 Introduction....4
2 High level architecture....4
3 Release....4
4 Core Data Structure....4
4.2 Version Information....4
4.3 Face ROI and Feature Dimension....5
4.4 Face Box....5
4.5 Face....5
4.6 Pixel Format....5
5 APIs....6
5.1 face_detect_init....6
5.2 face_detect_exit....6
5.3 face_detect....6
5.4 face_recognize_init....7
5.5 face_recognize_exit....7
5.6 face_align....7
5.7 face_recognize....7
5.8 face_recognize....8
6 Sample code....8
6.1 Initialize the Pipeline....8
6.2 Face Detection, Alignment and Recognition....9
6.3 Cleanup the pipeline....9

version	Author	Description
0.1	Dongsheng Zhang (dongsheng.zhang@nxp.com)	initial version

1 Introduction

Oasis is a cross OS and Platform AI/ML library for NXP MCU&MPU AI solutions. It includes the CNN inference engine, computer vision utilities and different AI vison functionalities such as Face Detection, Face Alignment, Face recognition, etc.

2 High Level Architecture



3 Release

The oasis library is released in static library format with the header files.

4 Core Data Structure

4.2 Version Information

Name	Type	Description
VERSION_MAJOR	int	The major version number
VERSION_MINOR	int	The minor version number

4.3 Face ROI and Feature Dimension

Name	Type	Description
FACE_W	int	The width of the face ROI for recognition
FACE_H	int	The height of the face ROI for recognition
FACE_FEATURES	int	The feature size of the face ROI

4.4 Face Box

Name	Type	Description
rect	int array	The left, top, right, bottom of the face bounding box
fld	Float arry	The x, y axes position of the 5 face landmark points

4.5 Face

Name	Type	Description
bbox	Face Box	The bounding box and landmark of a face
name	string	The name of the face
feature	Float vector	The feature of the face

4.6 Pixel Format

Name	Type	Description
PIXEL_RGB	enum	The RGB format
PIXEL_BGR	enum	The BGR format

5 APIs

5.1 face_detect_init

Function	**int face_detect_init(const unsigned char* param1, const unsigned char* model1, const unsigned char* param2, const unsigned char* model2, const unsigned char* param3, const unsigned char* model3, int minFace, bool onlyMaxFace)**
Description	Initialize the face detect pipeline for the face detection.
Input Param	Parma*: the face detect model parameters Model*: the face detect model minFace: the minimum face to be detected onlyMaxFace: flag to indicate if only detect the Maximum face in the frame
Return Value	0: Success Else: error code

5.2 face_detect_exit

Function	**int face_detect_exit(void)**
Description	Cleanup the face detect pipeline

5.3 face_detect

Function	**int face_detect(unsigned char* data, int type, int w, int h, std::vector<Face>& faces)**
Description	Detect the faces in the frame data
Input Param	data：the pixel frame data type: the pixel format w: the width of the pixels h: the height of the pixels.
Output Param	faces: faces which was detected in the frame
Return Value	The face count which was detected in the frame

5.4 face_recognize_init

Function	**int face_recognize_init(const unsigned char* param, const unsigned char* model);**
Description	Initialize the face recognize pipeline
Input Param	param：the face recognize model parameter model: the face recognize model
Return Value	0: Success Else: error code

5.5 face_recognize_exit

Function	**int face_recognize_exit();**
Description	Cleanup the face recognize pipeline

5.6 face_align

Function	**int face_align(unsigned char* frame, int w, int h, FBox& bbox, unsigned char* data)**
Description	Get the aligned face ROI data in the frame for recognition according to the detected face boxes.
Input Param	frame：frame data in RGB/BGR format w: width of the frame in pixels h: height of the frame in pixels bbox: the detected face bounding box
Output Param	data: the aligned face ROI data
Return Value	the count of aligned face ROI

5.7 face_recognize

Function	**int face_recognize(const unsigned char* face, int type, std::vector<float>& feature)**
Description	Extract the face feature from the aligned face ROI
Input Param	face：the aligned face ROI data type: the pixel format of the face ROI data
Output Param	feature: the feature of the face
Return Value	1: success extracted the feature

	Else: error code

5.8 face_recognize

Function	**int face_recognize(unsigned char* frame, int type, int w, int h, std::vector<oasis::Face>& faces)**
Description	Extract the face feature from the original frame
Input Param	frame：the pixel frame data type: the pixel format w: the width of the pixels h: the height of the pixels. faces: the detected faces
Output Param	faces: the extracted feature will be stored in the feature of the faces
Return Value	The face count of successfully extracted feature

6 Sample code

6.1 Initialize the Pipeline

```
#include "face_detect.h"
#include "face_recognize.h"
#include "face_detect_model.h"
#include "face_recognize_model.h"

int Oasis_Init()
{
    int ret = 0;
    memset(&gFaceRecBuf, 0x0, sizeof(gFaceRecBuf));

    ret = face_detect_init(det12x12_param, det12x12_bin, det24x24_param, det24x24_bin,
                           det48x48_param,    det48x48_bin,    FACEREC_MINFACE,
FACEREC_MAXFACE);

    if (ret) {
        LOGE("[ERROR]:FaceRec_Init failed\n");
        return ret;
    }
    ret = face_recognize_init(rec_param, rec_bin);

    if (ret) {
        LOGE("[ERROR]:FaceRec_Init failed\n");
        return ret;
    }


    return ret;
}
```

6.2 Face Detection, Alignment and Recognition

```
#include "face_detect.h"
#include "face_recognize.h"

#define REC_RECT_WIDTH 320
#define REC_RECT_HEIGHT 240

std::vector<Face> recgFaces;
std::vector<float> feature;
uint8_t    snapshot[FACE_WIDTH * FACE_HEIGHT * 3];

int fcount = face_detect((uint8_t*)frame, oasis::PIXEL_RGB, REC_RECT_WIDTH,
REC_RECT_HEIGHT, recgFaces);

fcount = face_align((uint8_t*)frame, REC_RECT_WIDTH, REC_RECT_HEIGHT,
recgFaces[0].bbox, snapshot);

fcount = face_recognize(gFaceRecBuf.snapshot, oasis::PIXEL_RGB, feature);
```

6.3 Cleanup the Pipeline

```
#include "face_detect.h"
#include "face_recognize.h"

int Oasis_Exit()
{
    int ret = 0;
    ret = face_detect_exit();

    if (ret) {
        LOGE("[ERROR]:face_detect_exit failed\n");
    }

    ret = face_recognize_exit();

    if (ret) {
        LOGE("[ERROR]:face_recognize_exit failed\n");
    }

    return ret;
}
```