人脸识别设计文档

徐　硕 201710733231

赵国宏 201710733239

王伟旭 201710733227

第一部分

设计概述 /Design Introduction

1.1 设计目的

在 21 世纪这个信息时代，随着信息技术的飞速发展，深刻地影响着人们的生活方式和工作方式，其中图像处理技术给人们的生活中带来了诸多的便利，为人们解决了很多问题。人脸识别系统作为安全系统的重要部分，它在企业、电子护照及身份证、信息安全等领域起到了无可替代的作用。人脸识别是指在图片或视频流中识别出人脸，并对该人脸进行一系列相关操作技术。然而，传统的人脸识别系统存由于姿势、光照或遮挡等原因，在非强迫环境下的人脸识别和对齐是一项具有挑战性的问题。因此，本项目基于 Xilinx 公司的 Zynq SOC 器件和 OV5640 摄像头模组，利用多任务级联卷积神经网络（Multi-task Convolutional Neural Network， MTCNN）实现了人脸检测与识别系统。

1.2 应用领域

人脸识别应用的领域极为广泛，典型的应用场景可以归纳为以下几个方面。

1.2.1 身份认证场景

这是人脸识别技术最典型的应用场景之一。生活中的门禁系统、手机解锁等都可以归类为该种类别。

1.2.2 证件验证场景

证件验证与身份识别认证相似，也可称为人脸验证。它是判断证件中的人脸图像与被识别人的人脸是否相同的场景。

1.2.3 人脸检索场景

人脸检索与身份证验证类似，其主要功能是将人脸检索的图片进行“一对多”的图像对比验证。

1.3 适用范围

本项目在嵌入式 Zynq-7000 SOC 平台、OV5640 摄像头和 Linux 系统环境下构建人脸识别系统。该系统可移植性强、应用场景广泛，例如身份认证场景、证件验证场景、人脸检索场景、人脸分类场景、交互式应用场景等。

第二部分

系统组成及功能说明 /System Construction & Function Description

2.1 系统介绍

人脸检测与识别的整体架构如图 1 所示。系统主要由 xc7z020 SOC 器件、OV5640 摄像头模组、外部 DDR3 动态存储器、HDMI 显示接口等构成。系统基于软硬件协同设计的思想进行设计，主要分为四个部分，分别是图像采集、图像预处理、人脸检测、界面及图像显示。图像采集和图像预处理主要由系统硬件部分实现，人脸检测与识别和图像显示由系统软件部分实现。

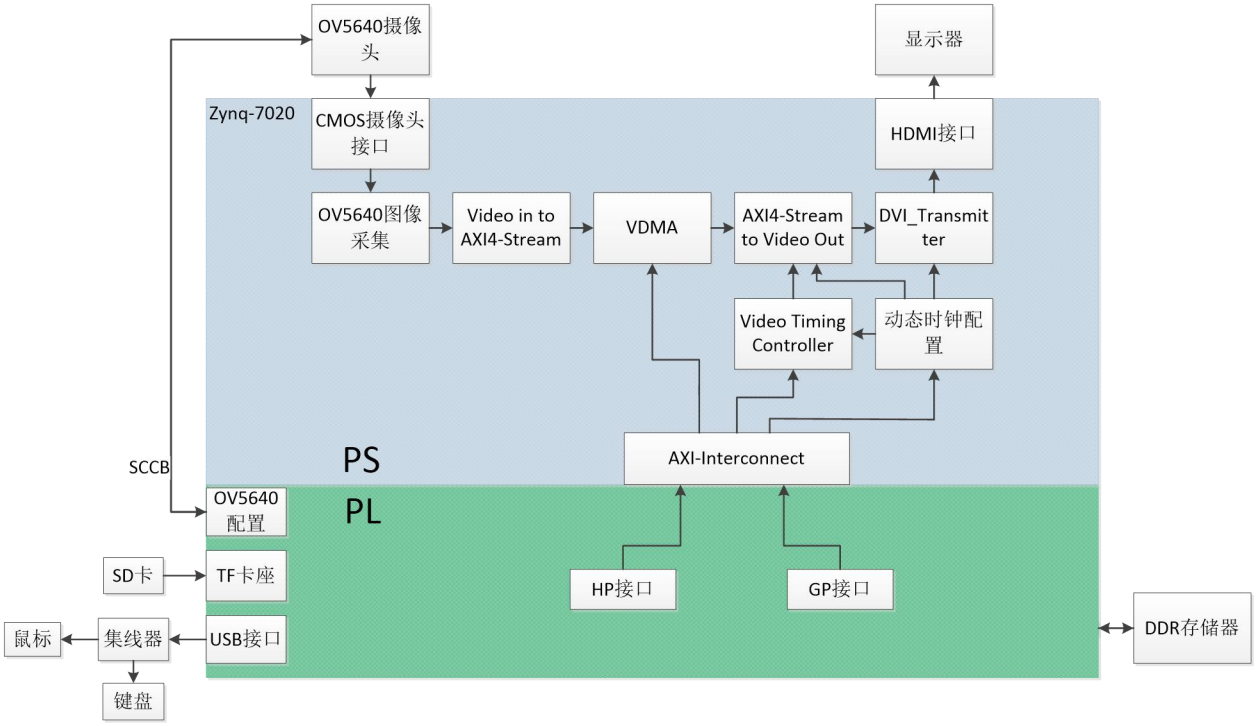

图 1 系统整体架构

2.2 各模块介绍

2.2.1 xc7z020 SOC 器件

xc7z020 SOC 器件作为整个系统的核心，包括处理器系统（Processing System，PS）和可编程逻辑（Programmable Logic，PL）两部分。PS 部分集成了 Cortex-A9 双核硬核

处理器，PL 部分提供了海量的可编程逻辑单元和 DSP 资源。

（1） PS 实现的主要功能

处理器系统 PS 部分实现了主要功能包括：运行 Linux 系统；调用自定义的 IP 模块实现与 PL 部分进行数据交互；运行系统应用程序。

（2） PL 实现的主要功能

可编程逻辑 PL 部分实现了主要功能包括：调用 OV5640 摄像头并获取图像数据；将图像数据通过 VDMA 传输至 DDR3 动态存储器保存；将 DDR3 中保存的图像数据通过 VDMA 传回至图像处理模块，并把处理里后的数据重写，写回到 DDR3 存储器；定制 HDMI 模块用于实现图像数据的显示。

2.2.2 OV5640 摄像头

系统中使用的摄像头是 OmniVision 公司的 OV5640 传感器模组。它是一款 1/4 英寸单芯片图像传感器，其感光阵列达到 2592*1944（即 500W 像素），能实现最快 15fps QSXVGA（2592*1944）或者 90fpsVGA（640*480）分辨率的图像采集。该传感器模组提供了自动图像控制功能，包括自动曝光控制、自动白平衡、和自动带宽滤波和自动聚焦控制等功能。本系统中 OV5640 传感器模组负责图像数据的采集，即人脸图像的采集。

2.2.3 多任务级联卷积神经网络

多任务级联卷积神经网络（Multi-task Cascaded Convolutional Networks，MTCNN）由三个阶段组成，具体包括：第一阶段，通过 CNN 快速产生候选窗体。第二阶段，通过更复杂一点的 CNN 精炼候选窗体，丢弃大量重叠窗体。第三阶段，使用更强大的 CNN，实现候选窗体去留，并显示人脸关键点定位。3 个阶段的流程如图 2 所示。需要指出的是，MTCNN 网络在进行人脸检测前，先进行多尺度变换处理，即将一幅人脸图片缩放为不同尺寸的图片，这就是所谓的图像金字塔。这些不同的尺寸的图像将作为 3 个阶段的输入数据进行训练。

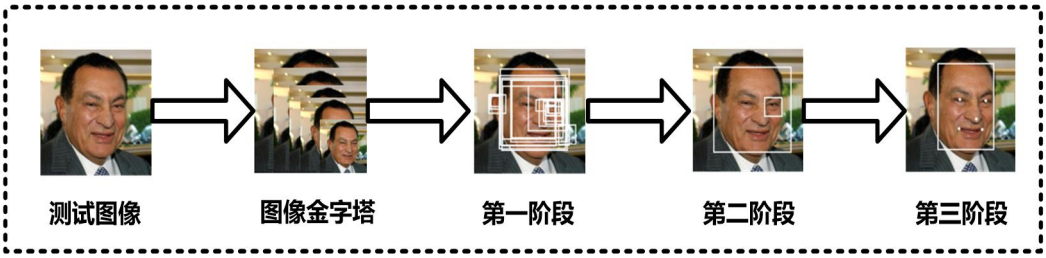

图 2 MTCNN 三阶段的流程示例

MTCNN 由 P-Net（Proposal Network）、R-Net（Refinement Network）、O-Net（Output Network）3 个网络结构组成。对于一个给定的图像，首先将其调整到不同的比例，以构建一个图像金字塔，作为三级级联框架的输入。MTCNN 工作流程如图 3 所示。

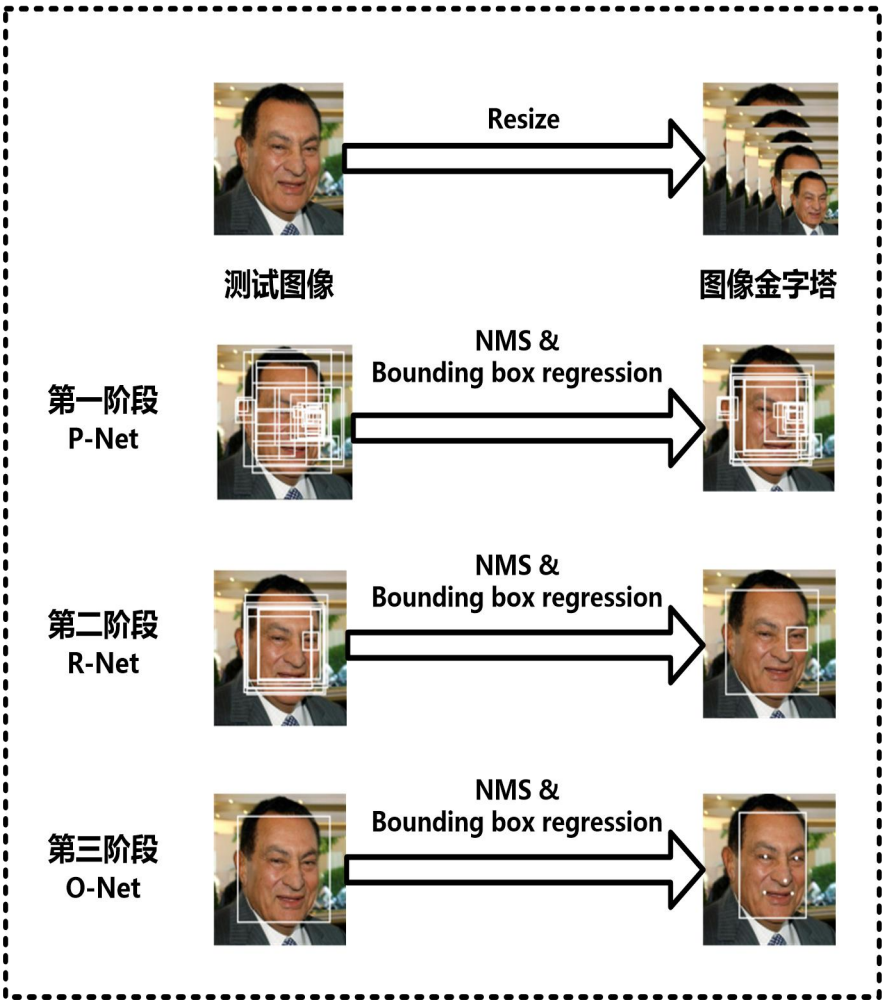

图 3 MTCNN 三个阶段工作流程

第一阶段使用 P-Net 卷积神经网络获得了人脸区域的候选窗口和边界框的回归向量。同时，对候选窗口根据边界框进行校准。然后，利用非极大值抑制算法去除重叠窗体。

第二阶段使用 R-Net 卷积神经网络进行操作，将经过 P-Net 确定的包含候选窗体的图片在 R-Net 网络中训练，然后使用全连接网络进行分类。通过边界框向量微调候选窗体，最后还是利用非极大值抑制算法去除重叠窗体。

第三阶段使用 O-Net 卷积神经网络进行操作，该网络比 R-Net 层多一层卷积层，所以处理的结果会更加精细。作用和 R-Net 层作用一样。但是，该层对人脸区域进行了更多的监督，同时还会标定人脸关键点的位置。

第三部分

完成情况及性能参数 /Final Design & Performance Parameters

本项目在 Zynq-7000 SOC 器件上，通过 OV5640 摄像头采集图像，并在 Linux 系统环境下利用 MTCNN 网络实现人脸检测与识别。本项目主要完成了硬件系统设计、Linux 系统移植、上位机图形界面设计。

本项目硬件系统是基于 Xilinx 公司的 Vivado 2017.4 进行设计，整个硬件系统的设计结构及 IP 如图 4 所示。

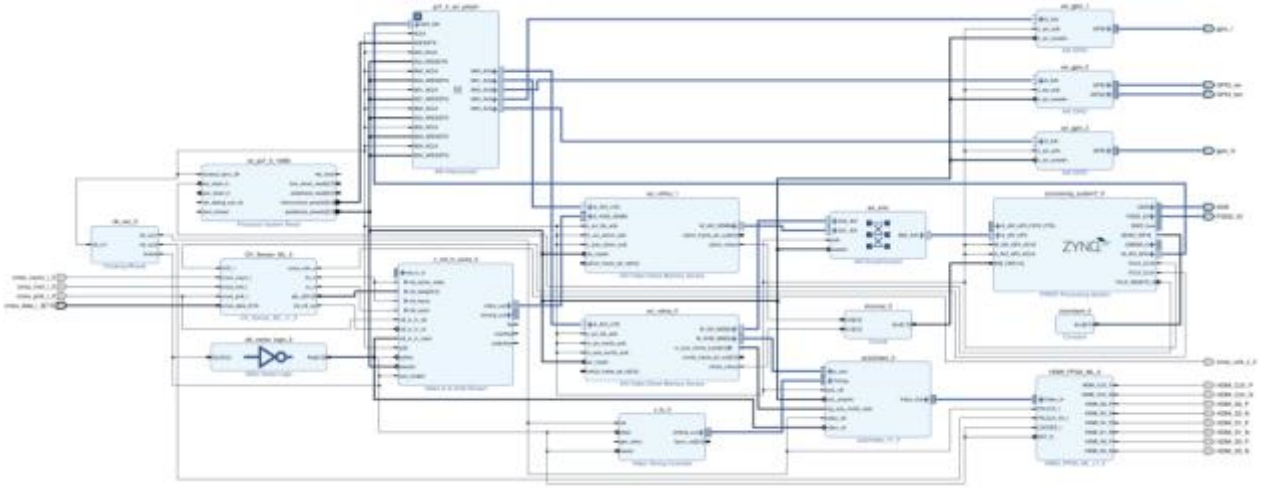

图 4 人脸识别系统的硬件系统设计

本项目移植的 Linux 系统是 Debian GNU/Linux，系统在 Zynq-7000 SOC 开发板运行情况如图 5 所示.

图 5 板载 Debian GNU/Linux 的运行情况

本项目上位机图形界面采用 QT Creator 集成开发工具进行嵌入式软件设计和代码编写，实现对图像的显示及控制功能，运行结果如图 6 所示。

图 6 上位机图形界面

此外，本次项目中我们使用了训练好的 MTCNN 网络来进行人脸检测与识别。MTCNN 网络在刚诞生的时候其性能表现最优。表 1 是 CNN 与 MTCNN 处理图像的速度与验证精度对比。随着人工智能和神经网络的不断发展，MTCNN 当下已经不是最优的网络了，但是该网络是一个非常有意义的架构，它将人脸检测与人脸特征点定位结合起来，而人脸特征点又可以用来实现人脸矫正。

表 1 MTCNN 处理图像的速度与验证精度对比

Group	CNN	300 × Forward Propagation	Validation Accuracy
Group1	12-Net	0.038s	94.4%
	P-Net	0.031s	94.6%
Group2	24-Net	0.738s	95.1%
	R-NET	0.458s	95.4%
Group3	48-Net	3.577s	93.2%
	O-Net	1.347s	95.4%

第四部分

总结 /Conclusions

4.1 主要创新点

1. 在 Xilinx 公司 Zynq-7000 SOC 器件上搭建嵌入式开发系统，包括编译环境的搭建，移植了 Debian9 系统，该系统体积小、稳定性好。

2. 系统硬件设计部分，利用 Zynq-7000 SOC 器件上的可编程逻辑(PL)部分定制了相关 IP，实现系统各模块的控制逻辑，包括视频图像数据的传输控制以及 HDMI 高清显示控制。

3. ARM 部分软件设计，包括采用 QT Creator 的上位机图形界面设计，以及使用训练好的 MTCNN 网络结构实现视频图像中人脸检测与识别功能。

4.2 可扩展之处

本系统基于 Zynq-7000 SOC 器件、OV5640 摄像头和 Linux 系统实现了人脸识别系统。由于人脸识别算法采用 MTCNN 网络结构，该网络同时实现人脸检测和人脸特征点的标定。因此，本系统可以应用到人脸特征检测与人脸检索的环境中，例如人脸分类、表情识别、人脸跟踪，从而为用户有针对性地推荐一些感兴趣的人。

4.3 心得体会

在此次竞赛之前，我们只是初步了解 FPGA 的设计，通过此次竞赛，使我们由开始浅陋的了解到现在有了很大的进步。在此次设计中，我们自主学习了 FPGA 和各种编程语言的知识，为了让我们的作品更加完善，我们花费了大量的时间与精力。一开始选择人脸识别这一课题，我们觉得无从下手，到了现在对它已经很了解，就像交朋友那样，由陌生到熟悉。其次移植系统时搭建环境很费时间和精力，我们最初使用 Ubuntu 系统，但是尝试了很久没有成功，最后转战 Debian 系统，从中寻到了一线生机。在此次项目中我们得到了锻炼，最大的收获还是我们一起去解决问题，努力拼搏，永不言弃。人生路漫漫，有付出就必定有收获，当我们回头再看现在之时，能够看出我们青春时努力拼搏的身影，这都将成为我们美好珍贵的回忆。

第五部分

附录

源程序代码

```
//
// Created by Lonqi on 2017/11/18.
//
#pragma once

#ifndef __MTCNN_NCNN_H__
#define __MTCNN_NCNN_H__
#include "net.h"
```

```
//#include <opencv2/opencv.hpp>
#include <string>
#include <vector>
#include <time.h>
#include <algorithm>
#include <map>
#include <iostream>
#include <math.h>
using namespace std;
//using namespace cv;
struct Bbox
{
	float score;
	int x1;
	int y1;
	int x2;
	int y2;
	float area;
	float ppoint[10];
	float regreCoord[4];
};


class MTCNN {

public:
	MTCNN(const string &model_path);
	MTCNN(const std::vector<std::string> param_files, const std::vector<std::string> bin_files);
	~MTCNN();

	void SetMinFace(int minSize);
	void detect(ncnn::Mat& img_, std::vector<Bbox>& finalBbox);
	void detectMaxFace(ncnn::Mat& img_, std::vector<Bbox>& finalBbox);
  //	void detection(const cv::Mat& img, std::vector<cv::Rect>& rectangles);
private:
	void generateBbox(ncnn::Mat score, ncnn::Mat location, vector<Bbox>& boundingBox_, float scale);
	void nmsTwoBoxs(vector<Bbox> &boundingBox_, vector<Bbox> &previousBox_, const float
overlap_threshold, string modelname = "Union");
	void nms(vector<Bbox> &boundingBox_, const float overlap_threshold, string modelname="Union");
	void refine(vector<Bbox> &vecBbox, const int &height, const int &width, bool square);
	void extractMaxFace(vector<Bbox> &boundingBox_);

	void PNet(float scale);
	void PNet();
	void RNet();
```

```
        void ONet();

        ncnn::Net Pnet, Rnet, Onet;
        ncnn::Mat img;

        const float nms_threshold[3] = {0.5f, 0.7f, 0.7f};
        const float mean_vals[3] = {0, 0, 0};
        const float norm_vals[3] = {1, 1, 1};
        const int MIN_DET_SIZE = 12;
        std::vector<Bbox> firstPreviousBbox_, secondPreviousBbox_, thirdPreviossBbox_;
        std::vector<Bbox> firstBbox_, secondBbox_,thirdBbox_;
        int img_w, img_h;

private://部分可调参数
        const float threshold[3] = { 0.6f, 0.7f, 0.7f };
        int minsize = 40;
        const float pre_facetor = 0.709f;

};


#endif //__MTCNN_NCNN_H__
//
// Created by Longqi on 2017/11/18..
//

/*
 * TO DO : change the P-net and update the generat box
 */
#define NOMINMAX
#include "mtcnn.h"

bool cmpScore(Bbox lsh, Bbox rsh) {
        if (lsh.score < rsh.score)
                return true;
        else
                return false;
}

bool cmpArea(Bbox lsh, Bbox rsh) {
        if (lsh.area < rsh.area)
                return false;
        else
                return true;
}
```

```
//MTCNN::MTCNN(){}
MTCNN::MTCNN(const string &model_path) {

    std::vector<std::string> param_files = {
        model_path+"/det1.param",
        model_path+"/det2.param",
        model_path+"/det3.param"
    };

    std::vector<std::string> bin_files = {
        model_path+"/det1.bin",
        model_path+"/det2.bin",
        model_path+"/det3.bin"
    };

    Pnet.load_param(param_files[0].data());
    Pnet.load_model(bin_files[0].data());
    Rnet.load_param(param_files[1].data());
    Rnet.load_model(bin_files[1].data());
    Onet.load_param(param_files[2].data());
    Onet.load_model(bin_files[2].data());
}

MTCNN::MTCNN(const std::vector<std::string> param_files, const std::vector<std::string> bin_files){
    Pnet.load_param(param_files[0].data());
    Pnet.load_model(bin_files[0].data());
    Rnet.load_param(param_files[1].data());
    Rnet.load_model(bin_files[1].data());
    Onet.load_param(param_files[2].data());
    Onet.load_model(bin_files[2].data());
}


MTCNN::~MTCNN(){
    Pnet.clear();
    Rnet.clear();
    Onet.clear();
}
void MTCNN::SetMinFace(int minSize){
    minsize = minSize;
}
void MTCNN::generateBbox(ncnn::Mat score, ncnn::Mat location, std::vector<Bbox>& boundingBox_, float scale){
    const int stride = 2;
    const int cellsize = 12;
```

```
    //score p
    float *p = (float *)score.data;//score.data + score.cstep;
    //float *plocal = location.data;
    Bbox bbox;
    float inv_scale = 1.0f/scale;
    //std::cout << score.h << std::endl;
    //std::cout << score.w << std::endl;
    for(int row=0;row<score.h;row++){
        for(int col=0;col<score.w;col++){
            if(*p>threshold[0]){
                bbox.score = *p;
                bbox.x1 = round((stride*col+1)*inv_scale);
                bbox.y1 = round((stride*row+1)*inv_scale);
                bbox.x2 = round((stride*col+1+cellsize)*inv_scale);
                bbox.y2 = round((stride*row+1+cellsize)*inv_scale);
                bbox.area = (bbox.x2 - bbox.x1) * (bbox.y2 - bbox.y1);
                const int index = row * score.w + col;
                for(int channel=0;channel<4;channel++){
                    bbox.regreCoord[channel]=location.channel(channel)[index];
                }
                boundingBox_.push_back(bbox);
            }
            p++;
            //plocal++;
        }
    }
}

void MTCNN::nmsTwoBoxs(vector<Bbox>& boundingBox_, vector<Bbox>& previousBox_, const float
overlap_threshold, string modelname)
{
    if (boundingBox_.empty()) {
        return;
    }
    sort(boundingBox_.begin(), boundingBox_.end(), cmpScore);
    float IOU = 0;
    float maxX = 0;
    float maxY = 0;
    float minX = 0;
    float minY = 0;
    //std::cout << boundingBox_.size() << " ";
    for (std::vector<Bbox>::iterator ity = previousBox_.begin(); ity != previousBox_.end(); ity++) {
        for (std::vector<Bbox>::iterator itx = boundingBox_.begin(); itx != boundingBox_.end();) {
            int i = itx - boundingBox_.begin();
            int j = ity - previousBox_.begin();
```

```
                maxX = std::max(boundingBox_.at(i).x1, previousBox_.at(j).x1);
                maxY = std::max(boundingBox_.at(i).y1, previousBox_.at(j).y1);
                minX = std::min(boundingBox_.at(i).x2, previousBox_.at(j).x2);
                minY = std::min(boundingBox_.at(i).y2, previousBox_.at(j).y2);
                //maxX1 and maxY1 reuse
                maxX = ((minX - maxX + 1)>0) ? (minX - maxX + 1) : 0;
                maxY = ((minY - maxY + 1)>0) ? (minY - maxY + 1) : 0;
                //IOU reuse for the area of two bbox
                IOU = maxX * maxY;
                if (!modelname.compare("Union"))
                    IOU = IOU / (boundingBox_.at(i).area + previousBox_.at(j).area - IOU);
                else if (!modelname.compare("Min")) {
                    IOU = IOU / ((boundingBox_.at(i).area < previousBox_.at(j).area) ? boundingBox_.at(i).area :
previousBox_.at(j).area);
                }
                if (IOU > overlap_threshold&&boundingBox_.at(i).score>previousBox_.at(j).score) {
                //if (IOU > overlap_threshold) {
                    itx = boundingBox_.erase(itx);
                }
                else {
                    itx++;
                }
            }
        }
        //std::cout << boundingBox_.size() << std::endl;
}

void MTCNN::nms(std::vector<Bbox> &boundingBox_, const float overlap_threshold, string modelname){
        if(boundingBox_.empty()){
            std::cout << boundingBox_.size() << std::endl;
            return;
        }
        std::cout << boundingBox_.size() << std::endl;
        sort(boundingBox_.begin(), boundingBox_.end(), cmpScore);
        float IOU = 0;
        float maxX = 0;
        float maxY = 0;
        float minX = 0;
        float minY = 0;
        std::vector<int> vPick;
        int nPick = 0;
        std::multimap<float, int> vScores;
        const int num_boxes = boundingBox_.size();
        vPick.resize(num_boxes);
        for (int i = 0; i < num_boxes; ++i){
```

```
            vScores.insert(std::pair<float, int>(boundingBox_[i].score, i));
        }
        while(vScores.size() > 0){
            int last = vScores.rbegin()->second;
            vPick[nPick] = last;
            nPick += 1;
            for (std::multimap<float, int>::iterator it = vScores.begin(); it != vScores.end();){
                int it_idx = it->second;
                maxX = std::max(boundingBox_.at(it_idx).x1, boundingBox_.at(last).x1);
                maxY = std::max(boundingBox_.at(it_idx).y1, boundingBox_.at(last).y1);
                minX = std::min(boundingBox_.at(it_idx).x2, boundingBox_.at(last).x2);
                minY = std::min(boundingBox_.at(it_idx).y2, boundingBox_.at(last).y2);
                //maxX1 and maxY1 reuse
                maxX = ((minX-maxX+1)>0)? (minX-maxX+1) : 0;
                maxY = ((minY-maxY+1)>0)? (minY-maxY+1) : 0;
                //IOU reuse for the area of two bbox
                IOU = maxX * maxY;
                if(!modelname.compare("Union"))
                    IOU = IOU/(boundingBox_.at(it_idx).area + boundingBox_.at(last).area - IOU);
                else if(!modelname.compare("Min")){
                    IOU = IOU/((boundingBox_.at(it_idx).area < boundingBox_.at(last).area)?
boundingBox_.at(it_idx).area : boundingBox_.at(last).area);
                }
                if(IOU > overlap_threshold){
                    it = vScores.erase(it);
                }else{
                    it++;
                }
            }
        }

        vPick.resize(nPick);
        std::vector<Bbox> tmp_;
        tmp_.resize(nPick);
        for(int i = 0; i < nPick; i++){
            tmp_[i] = boundingBox_[vPick[i]];
        }
        boundingBox_ = tmp_;
}
void MTCNN::refine(vector<Bbox> &vecBbox, const int &height, const int &width, bool square){
        if(vecBbox.empty()){
            cout<<"Bbox is empty!!"<<endl;
            return;
        }
        float bbw=0, bbh=0, maxSide=0;
```

```
        float h = 0, w = 0;
        float x1=0, y1=0, x2=0, y2=0;
        for(vector<Bbox>::iterator it=vecBbox.begin(); it!=vecBbox.end();it++){
            bbw = (*it).x2 - (*it).x1 + 1;
            bbh = (*it).y2 - (*it).y1 + 1;
            x1 = (*it).x1 + (*it).regreCoord[0]*bbw;
            y1 = (*it).y1 + (*it).regreCoord[1]*bbh;
            x2 = (*it).x2 + (*it).regreCoord[2]*bbw;
            y2 = (*it).y2 + (*it).regreCoord[3]*bbh;



            if(square){
                w = x2 - x1 + 1;
                h = y2 - y1 + 1;
                maxSide = (h>w)?h:w;
                x1 = x1 + w*0.5 - maxSide*0.5;
                y1 = y1 + h*0.5 - maxSide*0.5;
                (*it).x2 = round(x1 + maxSide - 1);
                (*it).y2 = round(y1 + maxSide - 1);
                (*it).x1 = round(x1);
                (*it).y1 = round(y1);
            }

            //boundary check
            if((*it).x1<0)(*it).x1=0;
            if((*it).y1<0)(*it).y1=0;
            if((*it).x2>width)(*it).x2 = width - 1;
            if((*it).y2>height)(*it).y2 = height - 1;

            it->area = (it->x2 - it->x1)*(it->y2 - it->y1);
        }
    }

    void MTCNN::extractMaxFace(vector<Bbox>& boundingBox_)
    {
        if (boundingBox_.empty()) {
            return;
        }
        sort(boundingBox_.begin(), boundingBox_.end(), cmpArea);
        for (std::vector<Bbox>::iterator itx = boundingBox_.begin() + 1; itx != boundingBox_.end();) {
            itx = boundingBox_.erase(itx);
        }
    }
```

```
void MTCNN::PNet(float scale)
{
	//first stage
	int hs = (int)ceil(img_h*scale);
	int ws = (int)ceil(img_w*scale);
	ncnn::Mat in;
	resize_bilinear(img, in, ws, hs);
	ncnn::Extractor ex = Pnet.create_extractor();
	ex.set_light_mode(true);
	//sex.set_num_threads(4);
	ex.input("data", in);
	ncnn::Mat score_, location_;
	ex.extract("PNet25", score_);
	ex.extract("PNetnConv2dnconv4n2n26", location_);
	std::vector<Bbox> boundingBox_;

	generateBbox(score_, location_, boundingBox_, scale);
	nms(boundingBox_, nms_threshold[0]);

	firstBbox_.insert(firstBbox_.end(), boundingBox_.begin(), boundingBox_.end());
	boundingBox_.clear();
}

void MTCNN::PNet(){
	firstBbox_.clear();
	float minl = img_w < img_h? img_w: img_h;
	float m = (float)MIN_DET_SIZE/minsize;
	minl *= m;
	float factor = pre_facetor;
	vector<float> scales_;
	while(minl>MIN_DET_SIZE){
		scales_.push_back(m);
		minl *= factor;
		m = m*factor;
	}
	for (size_t i = 0; i < scales_.size(); i++) {
		int hs = (int)ceil(img_h*scales_[i]);
		int ws = (int)ceil(img_w*scales_[i]);
		ncnn::Mat in;
		resize_bilinear(img, in, ws, hs);
		ncnn::Extractor ex = Pnet.create_extractor();
		//ex.set_num_threads(2);
		ex.set_light_mode(true);
		ex.input("data", in);
		ncnn::Mat score_, location_;
```

```
            ex.extract("PNet25", score_);
            ex.extract("PNetnConv2dnconv4n2n26", location_);
            std::vector<Bbox> boundingBox_;
            generateBbox(score_, location_, boundingBox_, scales_[i]);
            nms(boundingBox_, nms_threshold[0]);
            firstBbox_.insert(firstBbox_.end(), boundingBox_.begin(), boundingBox_.end());
            boundingBox_.clear();
        }
    }
    void MTCNN::RNet(){
        secondBbox_.clear();
        int count = 0;
        for(vector<Bbox>::iterator it=firstBbox_.begin(); it!=firstBbox_.end();it++){
            ncnn::Mat tempIm;
            copy_cut_border(img, tempIm, (*it).y1, img_h-(*it).y2, (*it).x1, img_w-(*it).x2);
            ncnn::Mat in;
            resize_bilinear(tempIm, in, 24, 24);
            ncnn::Extractor ex = Rnet.create_extractor();
             //ex.set_num_threads(2);
            ex.set_light_mode(true);
            ex.input("data", in);
            ncnn::Mat score, bbox;
            ex.extract("RNet33", score);
            ex.extract("RNetnLinearnconv5n2n34", bbox);
             if ((float)score[0] > threshold[1]) {
                for (int channel = 0; channel<4; channel++) {
                    it->regreCoord[channel] = (float)bbox[channel];//*(bbox.data+channel*bbox.cstep);
                }
                it->area = (it->x2 - it->x1)*(it->y2 - it->y1);
                it->score = score.channel(1)[0];//*(score.data+score.cstep);
                secondBbox_.push_back(*it);
             }
        }
    }
    void MTCNN::ONet(){
        thirdBbox_.clear();
        for(vector<Bbox>::iterator it=secondBbox_.begin(); it!=secondBbox_.end();it++){
            ncnn::Mat tempIm;
            copy_cut_border(img, tempIm, (*it).y1, img_h-(*it).y2, (*it).x1, img_w-(*it).x2);
            ncnn::Mat in;
            resize_bilinear(tempIm, in, 48, 48);
            ncnn::Extractor ex = Onet.create_extractor();
             //ex.set_num_threads(2);
            ex.set_light_mode(true);
            ex.input("data", in);
```

```
            ncnn::Mat score, bbox, keyPoint;
            ex.extract("ONet40", score);
            ex.extract("ONetnLinearnconv6n2n41", bbox);
            ex.extract("ONetnLinearnconv6n3n42", keyPoint);
             if ((float)score[0] > threshold[2]) {
                    for (int channel = 0; channel < 4; channel++) {
                        it->regreCoord[channel] = (float)bbox[channel];
                    }
                    it->area = (it->x2 - it->x1) * (it->y2 - it->y1);
                    it->score = score.channel(1)[0];
                    for (int num = 0; num<5; num++) {
                        (it->ppoint)[num] = it->x1 + (it->x2 - it->x1) * keyPoint[num];
                        (it->ppoint)[num + 5] = it->y1 + (it->y2 - it->y1) * keyPoint[num + 5];
                    }

                    thirdBbox_.push_back(*it);
                }
        }
    }
void MTCNN::detect(ncnn::Mat& img_, std::vector<Bbox>& finalBbox_){
        img = img_;
        std::cout << img[0] << img[1] << img[2] << std::endl;
        img_w = img.w;
        img_h = img.h;
        img.substract_mean_normalize(mean_vals, norm_vals);
        std::cout << img[0] << std::endl;
        PNet();
        //the first stage's nms
        if(firstBbox_.size() < 1) return;
        nms(firstBbox_, nms_threshold[0]);
        refine(firstBbox_, img_h, img_w, true);
        //printf("firstBbox_.size()=%d\n", firstBbox_.size());

        //second stage
        RNet();
        //printf("secondBbox_.size()=%d\n", secondBbox_.size());
        if(secondBbox_.size() < 1) return;
        nms(secondBbox_, nms_threshold[1]);
        refine(secondBbox_, img_h, img_w, true);

        //third stage
        ONet();
        //printf("thirdBbox_.size()=%d\n", thirdBbox_.size());
        if(thirdBbox_.size() < 1) return;
        refine(thirdBbox_, img_h, img_w, true);
```

```
    nms(thirdBbox_, nms_threshold[2], "Min");
    finalBbox_ = thirdBbox_;
}


void MTCNN::detectMaxFace(ncnn::Mat& img_, std::vector<Bbox>& finalBbox) {
    firstPreviousBbox_.clear();
    secondPreviousBbox_.clear();
    thirdPrevioussBbox_.clear();
    firstBbox_.clear();
    secondBbox_.clear();
    thirdBbox_.clear();

    //norm
    img = img_;
    img_w = img.w;
    img_h = img.h;
    img.substract_mean_normalize(mean_vals, norm_vals);

    //pyramid size
    float minl = img_w < img_h ? img_w : img_h;
    float m = (float)MIN_DET_SIZE / minsize;
    minl *= m;
    float factor = pre_facetor;
    vector<float> scales_;
    while (minl>MIN_DET_SIZE) {
        scales_.push_back(m);
        minl *= factor;
        m = m*factor;
    }
    sort(scales_.begin(), scales_.end());
    //printf("scales_.size()=%d\n", scales_.size());

    //Change the sampling process.
    for (size_t i = 0; i < scales_.size(); i++)
    {
        //first stage
        PNet(scales_[i]);
        nms(firstBbox_, nms_threshold[0]);
        nmsTwoBoxs(firstBbox_, firstPreviousBbox_, nms_threshold[0]);
        if (firstBbox_.size() < 1) {
            firstBbox_.clear();
            continue;
        }
        firstPreviousBbox_.insert(firstPreviousBbox_.end(), firstBbox_.begin(), firstBbox_.end());
```

```
            refine(firstBbox_, img_h, img_w, true);
            //printf("firstBbox_.size()=%d\n", firstBbox_.size());

            //second stage
            RNet();
            nms(secondBbox_, nms_threshold[1]);
            nmsTwoBoxs(secondBbox_, secondPreviousBbox_, nms_threshold[0]);
            secondPreviousBbox_.insert(secondPreviousBbox_.end(), secondBbox_.begin(), secondBbox_.end());
            if (secondBbox_.size() < 1) {
                firstBbox_.clear();
                secondBbox_.clear();
                continue;
            }
            refine(secondBbox_, img_h, img_w, true);
            //printf("secondBbox_.size()=%d\n", secondBbox_.size());

            //third stage
            ONet();
            //printf("thirdBbox_.size()=%d\n", thirdBbox_.size());
            if (thirdBbox_.size() < 1) {
                firstBbox_.clear();
                secondBbox_.clear();
                thirdBbox_.clear();
                continue;
            }
            refine(thirdBbox_, img_h, img_w, true);
            nms(thirdBbox_, nms_threshold[2], "Min");

            if (thirdBbox_.size() > 0) {
                extractMaxFace(thirdBbox_);
                finalBbox = thirdBbox_;//if largest face size is similar,.
                break;
            }
        }

        //printf("firstPreviousBbox_.size()=%d\n", firstPreviousBbox_.size());
        //printf("secondPreviousBbox_.size()=%d\n", secondPreviousBbox_.size());
    }
```