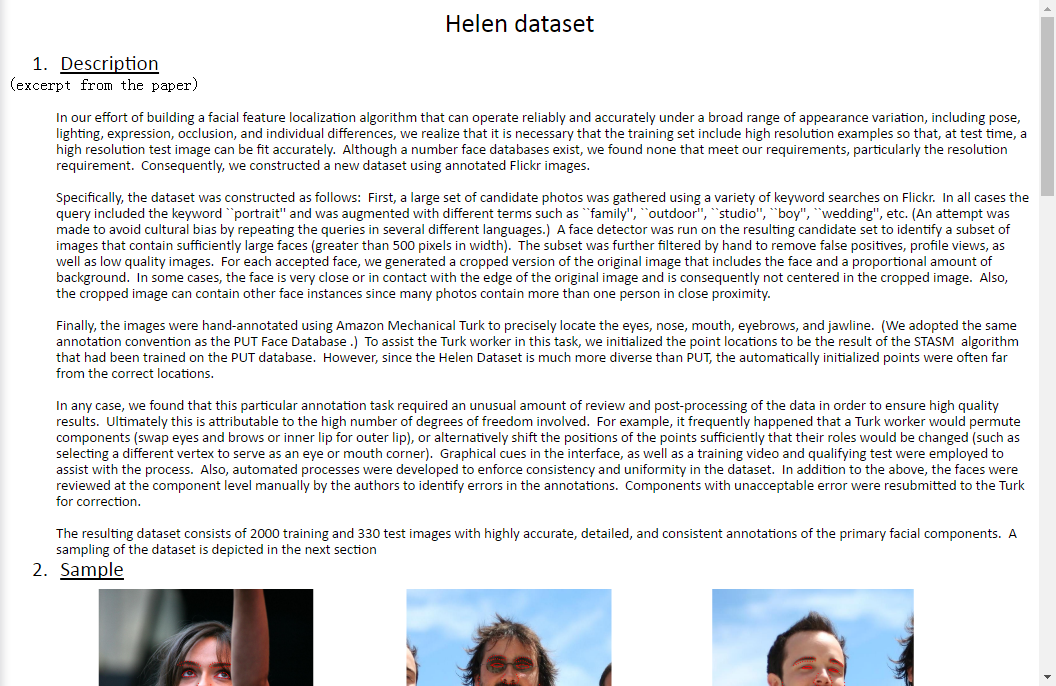
# 第8课作业

本次作业没有成功，花了较多时间训练人脸标志点模型，但测试效果一塌糊涂，多次训练后结果没有改善，因时间紧迫，待后续再来检查原因。因SeetaFace环境配置较为容易，本次作业使用SeetaFace完成。

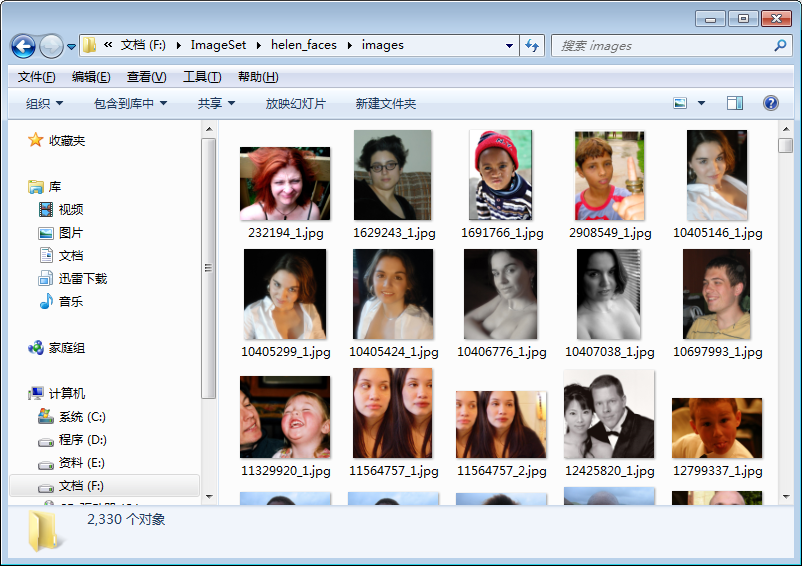
## landmark训练

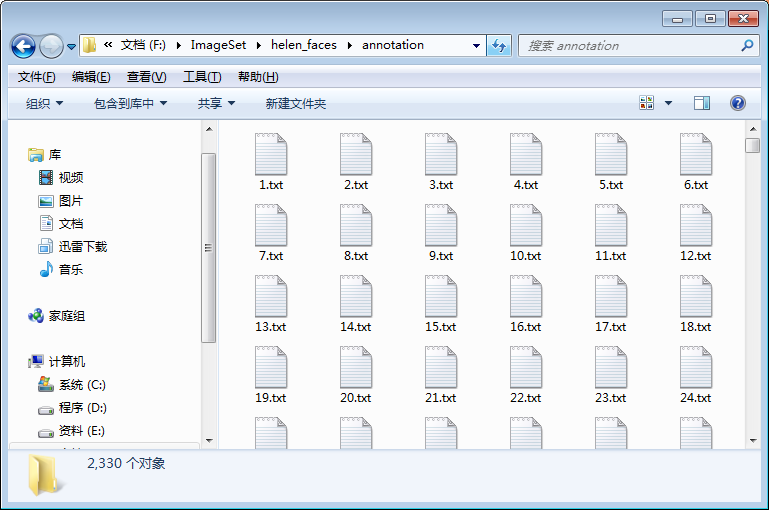
### 准备数据集

这里我使用的是helen数据集，网址：<http://www.ifp.illinois.edu/~vuongle2/helen/>



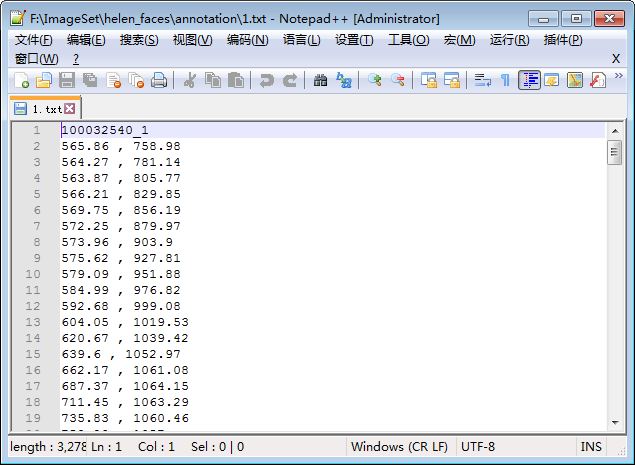
下载后将图像和标注文件解压：





打开一个标注文件可以发现，标注文件第一行为对应的图像文件名，以下每一行是一个地标点的坐标，这样的格式与OpenCV官方介绍的格式并不完全一致： <https://docs.opencv.org/3.4.0/d6/d49/md__build_master-contrib_docs-lin64_opencv_contrib_modules_face_tutorials_face_landmark_face_landmark_trainer.html>

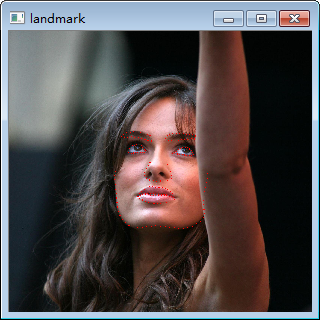
所以我编写了一个读取该格式文件的函数：



|  |
| --- |
| void loadTrainData(string imagefolder, string annotationfolder,  vector<vector<Point2f>>& trainlandmarks, vector<Mat>& images, int maxCount = 300)  {  images.clear();  trainlandmarks.clear();  vector<cv::String> annotation\_files;  string format = annotationfolder + "\\\*.txt";  glob(format, annotation\_files);  for (int i = 0; i < annotation\_files.size(); i++)  {  ifstream infile;  infile.open(annotation\_files[i], ios::in);  if (!infile)  {  cout << "Can not open " << annotation\_files[i] << endl;  continue;  }  string line, item;  //路径  getline(infile, line);  string path = imagefolder + "\\" + line + ".jpg";  Mat image = imread(path);  if (!image.data)  {  cout << "Can not find " << path << endl;  continue;  }  vector<float> val;  vector<Point2f> landmarks;  while (getline(infile, line))  {  std::istringstream ss(line);  getline(ss, item, ',');  float x = (float)atof(item.c\_str());  getline(ss, item, ',');  float y = (float)atof(item.c\_str());  landmarks.push\_back(Point2f(x, y));  }  trainlandmarks.push\_back(landmarks);  images.push\_back(image.clone());  if (images.size() > maxCount)  {  break;  }  }  } |

测试一张图像：

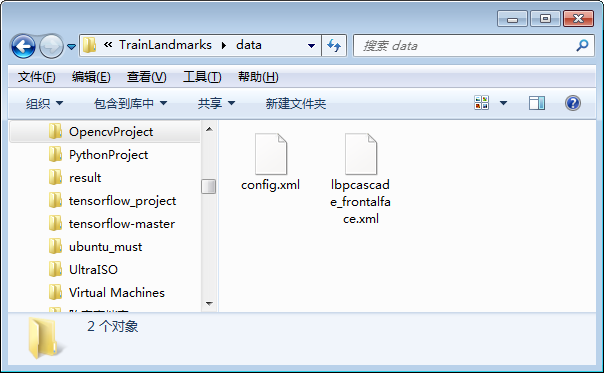
|  |
| --- |
| void main()  {  //载入训练数据  string imagefolder = "F:\\ImageSet\\helen\_faces\\images";  string annotationfolder = "F:\\ImageSet\\helen\_faces\\annotation";  vector<vector<Point2f>> trainlandmarks;  vector<Mat> images;  loadTrainData(imagefolder, annotationfolder, trainlandmarks, images, 1);  //显示一张图像  if (trainlandmarks.size() > 0)  {  Mat image = images[10].clone();  drawLandmark(image, trainlandmarks[10]);  imshow("landmark", image);  waitKey(0);  }  } |



可见，读取数据没有问题。

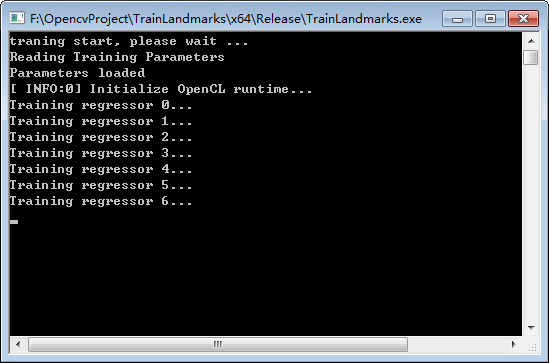
### 1.2 训练地标点检测模型

在工程目录下新建一个data文件夹，将config.xml和lbpcascade\_frontalface.xml放在该文件夹下，这两个文件都可以在OpenCV的示例程序中找到。



训练的代码如下：

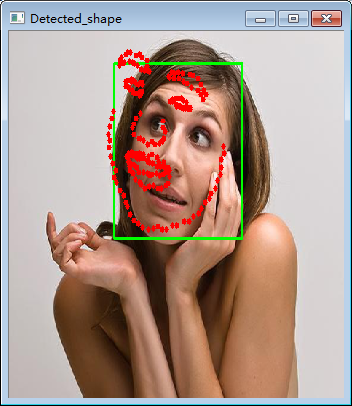
|  |
| --- |
| #include "stdafx.h"  #include <opencv2/opencv.hpp>  #include <fstream>  #include <iostream>  #include <string>  #include <opencv2\face.hpp>  using namespace std;  using namespace cv;  using namespace cv::face;  CascadeClassifier face\_cascade;  bool myDetector(InputArray image, OutputArray ROIs);  bool myDetector(InputArray image, OutputArray ROIs) {  Mat gray;  std::vector<Rect> faces;  if (image.channels() > 1) {  cvtColor(image.getMat(), gray, COLOR\_RGB2GRAY);  }  else {  gray = image.getMat().clone();  }  equalizeHist(gray, gray);  face\_cascade.detectMultiScale(gray, faces, 1.1, 3, 0, Size(30, 30));  Mat(faces).copyTo(ROIs);  return true;  }  void loadTrainData(string imagefolder, string annotationfolder,  vector<vector<Point2f>>& trainlandmarks, vector<Mat>& images, int maxCount = 300)  {  images.clear();  trainlandmarks.clear();  vector<cv::String> annotation\_files;  string format = annotationfolder + "\\\*.txt";  glob(format, annotation\_files);  for (int i = 0; i < annotation\_files.size(); i++)  {  ifstream infile;  infile.open(annotation\_files[i], ios::in);  if (!infile)  {  cout << "Can not open " << annotation\_files[i] << endl;  continue;  }  string line, item;  //路径  getline(infile, line);  string path = imagefolder + "\\" + line + ".jpg";  Mat image = imread(path);  if (!image.data)  {  cout << "Can not find " << path << endl;  continue;  }  vector<float> val;  vector<Point2f> landmarks;  while (getline(infile, line))  {  std::istringstream ss(line);  getline(ss, item, ',');  float x = (float)atof(item.c\_str());  getline(ss, item, ',');  float y = (float)atof(item.c\_str());  landmarks.push\_back(Point2f(x, y));  }  trainlandmarks.push\_back(landmarks);  images.push\_back(image.clone());  if (images.size() > maxCount)  {  break;  }  }  }  void drawLandmark(Mat& image, vector<Point2f> landmark)  {  for (int i = 0;i < landmark.size();i++)  {  circle(image, landmark[i], 3, Scalar(0, 0, 255), -1);  }  }  void main()  {  //初始化  Size scale(460, 460);  face\_cascade.load("data\\lbpcascade\_frontalface.xml");  FacemarkKazemi::Params params;  params.configfile = "data\\config.xml";  //载入训练数据  string imagefolder = "F:\\ImageSet\\helen\_faces\\images";  string annotationfolder = "F:\\ImageSet\\helen\_faces\\annotation";  vector<vector<Point2f>> trainlandmarks;  vector<Mat> images;  loadTrainData(imagefolder, annotationfolder, trainlandmarks, images, 2000);  //显示一张图像  if (trainlandmarks.size() > 0)  {  Mat image = images[0].clone();  drawLandmark(image, trainlandmarks[0]);  namedWindow("landmark", 0);  imshow("landmark", image);  waitKey(0);  }  //训练  cout << "traning start, please wait ..." << endl;  string modelfile\_name = "data\\landmark\_model.dat";  Ptr<FacemarkKazemi> facemark = FacemarkKazemi::create(params);  facemark->setFaceDetector((FN\_FaceDetector)myDetector, &face\_cascade);  facemark->training(images, trainlandmarks, params.configfile, scale, modelfile\_name);  cout << "training finish!" << endl;  } |



### 测试模型

测试代码：

|  |
| --- |
| void main()  {  face\_cascade.load("data\\lbpcascade\_frontalface.xml");  FacemarkKazemi::Params params;  Ptr<FacemarkKazemi> facemark = FacemarkKazemi::create(params);  facemark->setFaceDetector((FN\_FaceDetector)myDetector, &face\_cascade);  //facemark->loadModel("data\\face\_landmark\_model.dat");  facemark->loadModel("data\\landmark\_model\_2000.dat");  cout << "Loaded model" << endl;  //vector to store the faces detected in the image  vector<Rect> faces;  vector< vector<Point2f> > shapes;  String pattern = "F:\\ImageSet\\helen\_faces\\test\_images\\\*.jpg";  vector<String> filenames;  glob(pattern, filenames);  for (int i = 0;i < filenames.size();i++)  {  Mat img = imread(filenames[i]);  if (!img.data)continue;  faces.clear();  shapes.clear();  float r = 460.0 / img.cols;  resize(img, img, Size(460, img.rows \* r));  facemark->getFaces(img, faces);  if (faces.size() == 0) {  cout << "No faces found in this frame" << endl;  continue;  }  if (facemark->fit(img, faces, shapes))  {  for (unsigned long i = 0;i<faces.size();i++)  {  rectangle(img, faces[i], Scalar(0, 255, 0), 2);  drawLandmark(img, shapes[i]);  }  }  namedWindow("Detected\_shape", 0);  imshow("Detected\_shape", img);  waitKey(0);  }  } |



从结果看，本次训练效果基本可以用一塌糊涂来形容，多次训练后仍没解决问题，但鉴于时间有限，后续有时间再来检查出现的问题。

## SeetaFace的配置

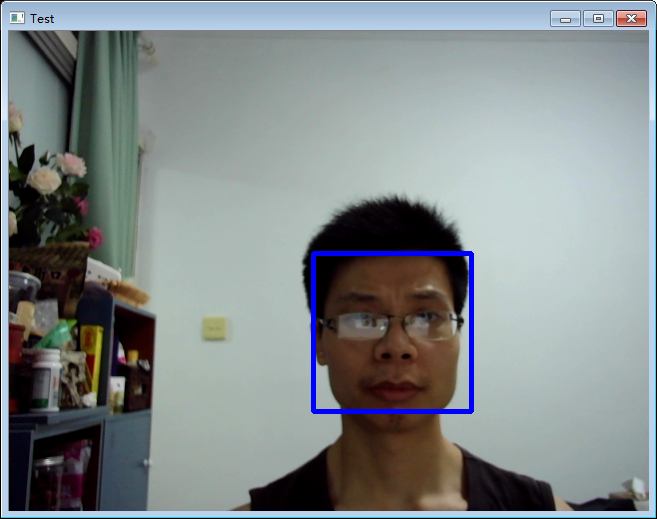
SeetaFace包括人脸检测模块、人脸对齐模块和人脸识别模块，项目地址：

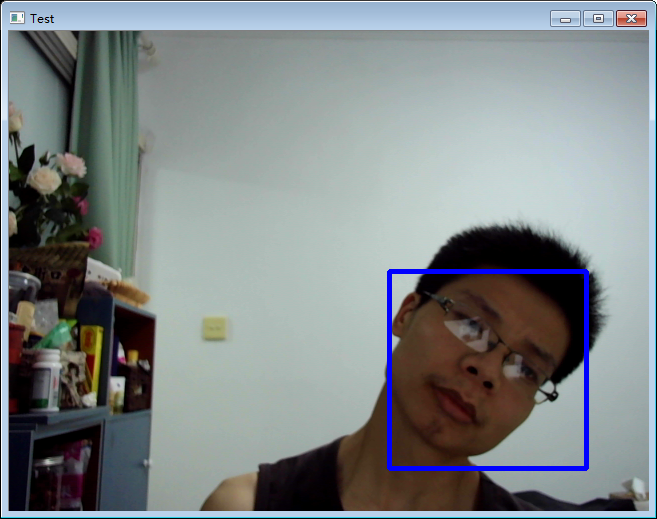
<https://github.com/seetaface/SeetaFaceEngine>，项目中包括详细的配置文档，参考该文档即可完成工程配置，所以配置过程在这里省略。

### 2.1人脸检测模块

配置好工程后，我仿照官方的例子，编写了以下代码进行测试：

|  |
| --- |
| #include <cstdint>  #include <fstream>  #include <iostream>  #include <string>  #include "opencv2/highgui/highgui.hpp"  #include "opencv2/imgproc/imgproc.hpp"  #include "face\_detection.h"  using namespace std;  using namespace cv;  int main(int argc, char\*\* argv) {  const char\* img\_path = "data/test\_01.jpg";  seeta::FaceDetection detector("data/seeta\_fd\_frontal\_v1.0.bin");  detector.SetMinFaceSize(40);  detector.SetScoreThresh(2.f);  detector.SetImagePyramidScaleFactor(0.8f);  detector.SetWindowStep(4, 4);  VideoCapture cap(0);  Mat img;  Mat img\_gray;  cap >> img;  while (img.data)  {  if (img.channels() != 1)  cv::cvtColor(img, img\_gray, cv::COLOR\_BGR2GRAY);  else  img\_gray = img;  seeta::ImageData img\_data;  img\_data.data = img\_gray.data;  img\_data.width = img\_gray.cols;  img\_data.height = img\_gray.rows;  img\_data.num\_channels = 1;  long t0 = cv::getTickCount();  std::vector<seeta::FaceInfo> faces = detector.Detect(img\_data);  long t1 = cv::getTickCount();  double secs = (t1 - t0) / cv::getTickFrequency();  cout << "Detections takes " << secs << " seconds " << endl;  cout << "Image size (wxh): " << img\_data.width << "x"  << img\_data.height << endl;  cv::Rect face\_rect;  int32\_t num\_face = static\_cast<int32\_t>(faces.size());  for (int32\_t i = 0; i < num\_face; i++) {  face\_rect.x = faces[i].bbox.x;  face\_rect.y = faces[i].bbox.y;  face\_rect.width = faces[i].bbox.width;  face\_rect.height = faces[i].bbox.height;  cv::rectangle(img, face\_rect, CV\_RGB(0, 0, 255), 4, 8, 0);  }  cv::namedWindow("Test", cv::WINDOW\_AUTOSIZE);  cv::imshow("Test", img);  cv::waitKey(10);  cap >> img;  }  } |





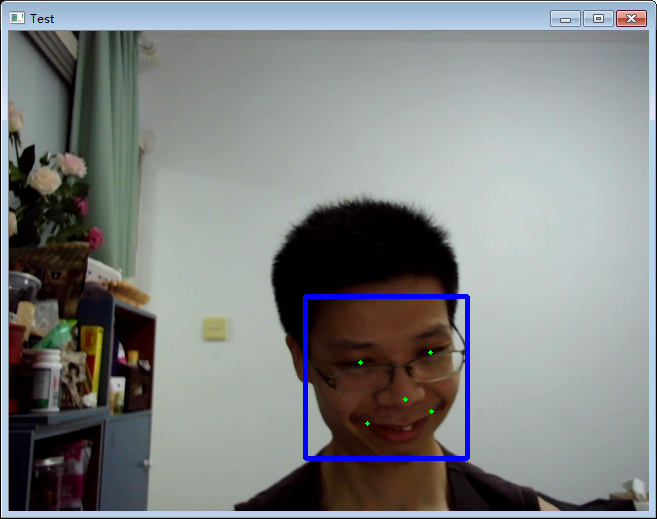
SeetaFace的人脸检测效果较好，对倾斜的人脸也能检测出来，速度也较快，在我的机器上平均160ms左右能够完成640\*480大小图像的处理。

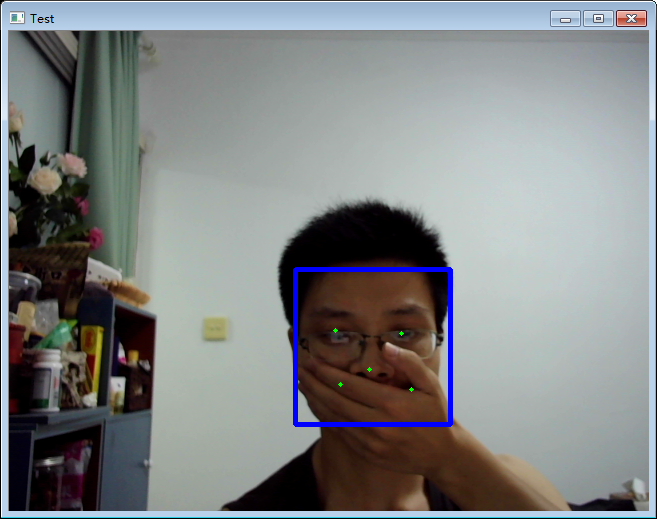
### 2.2人脸对齐模块

配置好工程后，我仿照官方的例子，编写了以下代码进行测试：

|  |
| --- |
| #include <cstdint>  #include <fstream>  #include <iostream>  #include <string>  #include "opencv2/highgui/highgui.hpp"  #include "opencv2/imgproc/imgproc.hpp"  #include "face\_detection.h"  #include "face\_alignment.h"  using namespace std;  using namespace cv;  int main(int argc, char\*\* argv) {  //人脸检测模型  seeta::FaceDetection detector("data/seeta\_fd\_frontal\_v1.0.bin");  detector.SetMinFaceSize(40);  detector.SetScoreThresh(2.f);  detector.SetImagePyramidScaleFactor(0.8f);  detector.SetWindowStep(4, 4);  //人脸对齐模型  seeta::FaceAlignment point\_detector("data/seeta\_fa\_v1.1.bin");  VideoCapture cap(0);  Mat img;  Mat img\_gray;  cap >> img;  while (img.data)  {  if (img.channels() != 1)  cv::cvtColor(img, img\_gray, cv::COLOR\_BGR2GRAY);  else  img\_gray = img;  seeta::ImageData img\_data;  img\_data.data = img\_gray.data;  img\_data.width = img\_gray.cols;  img\_data.height = img\_gray.rows;  img\_data.num\_channels = 1;  long t0 = cv::getTickCount();  //检测人脸  std::vector<seeta::FaceInfo> faces = detector.Detect(img\_data);  cv::Rect face\_rect;  int32\_t num\_face = static\_cast<int32\_t>(faces.size());  for (int32\_t i = 0; i < num\_face; i++) {  face\_rect.x = faces[i].bbox.x;  face\_rect.y = faces[i].bbox.y;  face\_rect.width = faces[i].bbox.width;  face\_rect.height = faces[i].bbox.height;  cv::rectangle(img, face\_rect, CV\_RGB(0, 0, 255), 4, 8, 0);  //检测人脸特征点  seeta::FacialLandmark points[5];  point\_detector.PointDetectLandmarks(img\_data, faces[0], points);  for (int i = 0; i<5; i++)  {  circle(img, Point(points[i].x, points[i].y), 2, CV\_RGB(0, 255, 0), -1);  }  }  long t1 = cv::getTickCount();  double secs = (t1 - t0) / cv::getTickFrequency();  cout << "Detections takes " << secs << " seconds " << endl;  cv::namedWindow("Test", cv::WINDOW\_AUTOSIZE);  cv::imshow("Test", img);  cv::waitKey(10);  cap >> img;  }  } |

从程序效果来看，SeetaFace能够准确检测出5个人脸特征点，并且当人脸局部被遮挡后，仍能够较为准确的给出人脸特征点的位置。





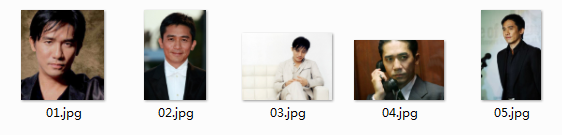
### 2.3人脸识别模块

为方便使用，我将人脸检测和特征点检测模块封装成两个函数模块，并参考例程编写了以下测试程序：

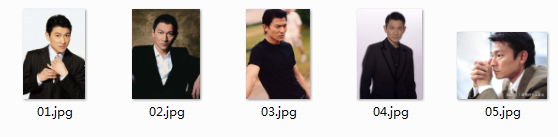
|  |
| --- |
| #include <cstdint>  #include <fstream>  #include <iostream>  #include <string>  #include "opencv2/highgui/highgui.hpp"  #include "opencv2/imgproc/imgproc.hpp"  #include "face\_detection.h"  #include "face\_alignment.h"  using namespace std;  using namespace cv;  int main(int argc, char\*\* argv) {  //人脸检测模型  seeta::FaceDetection detector("data/seeta\_fd\_frontal\_v1.0.bin");  detector.SetMinFaceSize(40);  detector.SetScoreThresh(2.f);  detector.SetImagePyramidScaleFactor(0.8f);  detector.SetWindowStep(4, 4);  //人脸对齐模型  seeta::FaceAlignment point\_detector("data/seeta\_fa\_v1.1.bin");  //人脸特征提取  seeta::FaceIdentification face\_recognizer("data/seeta\_fr\_v1.0.bin");  int feat\_size = face\_recognizer.feature\_size();  assert(feat\_size == 2048);  int n = 50, c = 2048, h = 1, w = 1;  Mat img;  Mat img\_gray;  string liudehua\_images[5] = {  "data/Liu/01.jpg",  "data/Liu/02.jpg",  "data/Liu/03.jpg",  "data/Liu/04.jpg",  "data/Liu/05.jpg"  };  string liangchaowei\_images[5] = {  "data/Liang/01.jpg",  "data/Liang/02.jpg",  "data/Liang/03.jpg",  "data/Liang/04.jpg",  "data/Liang/05.jpg"  };  float \*liudehua = new float[n \* c \* w \* h \* 5];  float \*liangchaowei = new float[n \* c \* w \* h \* 5];  memset(liudehua, 0, (n \* c \* w \* h \* 5) \* sizeof(float));  memset(liangchaowei, 0, (n \* c \* w \* h \* 5) \* sizeof(float));  for (int i = 0;i < 5;i++)  {  Mat img = imread(liudehua\_images[i]);  vector<FACE\_SHAPE> face\_shapes = detectFace(detector, point\_detector, img);  for (int t = 0;t < face\_shapes.size();t++)  {  FeatureExtract(face\_recognizer, face\_shapes[t], img, liudehua + i \* feat\_size);  }  img = imread(liangchaowei\_images[i]);  face\_shapes = detectFace(detector, point\_detector, img);  for (int t = 0;t < face\_shapes.size();t++)  {  FeatureExtract(face\_recognizer, face\_shapes[t], img, liangchaowei + i \* feat\_size);  }  }  //人脸检测及识别  Mat image = imread("data/test.jpg");  vector<FACE\_SHAPE> face\_shapes = detectFace(detector, point\_detector, image);  for (int i = 0;i < face\_shapes.size();i++)  {  float\* feat = new float[n \* c \* w \* h];  FeatureExtract(face\_recognizer, face\_shapes[i], image, feat);  //特征比对  float sim\_liu = 0, sim\_liang = 0;  for (int j = 0;j < 5;j++)  {  sim\_liu += face\_recognizer.CalcSimilarity(feat, liudehua + j \* feat\_size);  sim\_liang += face\_recognizer.CalcSimilarity(feat, liangchaowei + j \* feat\_size);  }  sim\_liu /= 5;  sim\_liang /= 5;  cout << sim\_liu << "," << sim\_liang << endl;  if (max(sim\_liu, sim\_liang) > 0.5)  {  if (sim\_liu > sim\_liang)  {  rectangle(image, face\_shapes[i].rect, Scalar(0, 255, 0), 2);  putText(image, "Liu De Hua", face\_shapes[i].rect.tl(), 1, 2, Scalar(0, 255, 0), 2);  }  else  {  rectangle(image, face\_shapes[i].rect, Scalar(255, 255, 0), 2);  putText(image, "Liang Chao Wei", face\_shapes[i].rect.tl(), 1, 2, Scalar(255, 255, 0), 2);  }  }  }  delete[] liudehua;  delete[] liangchaowei;  cv::imshow("Test", image);  cv::waitKey(0);  } |

我准备了一些人物照片：

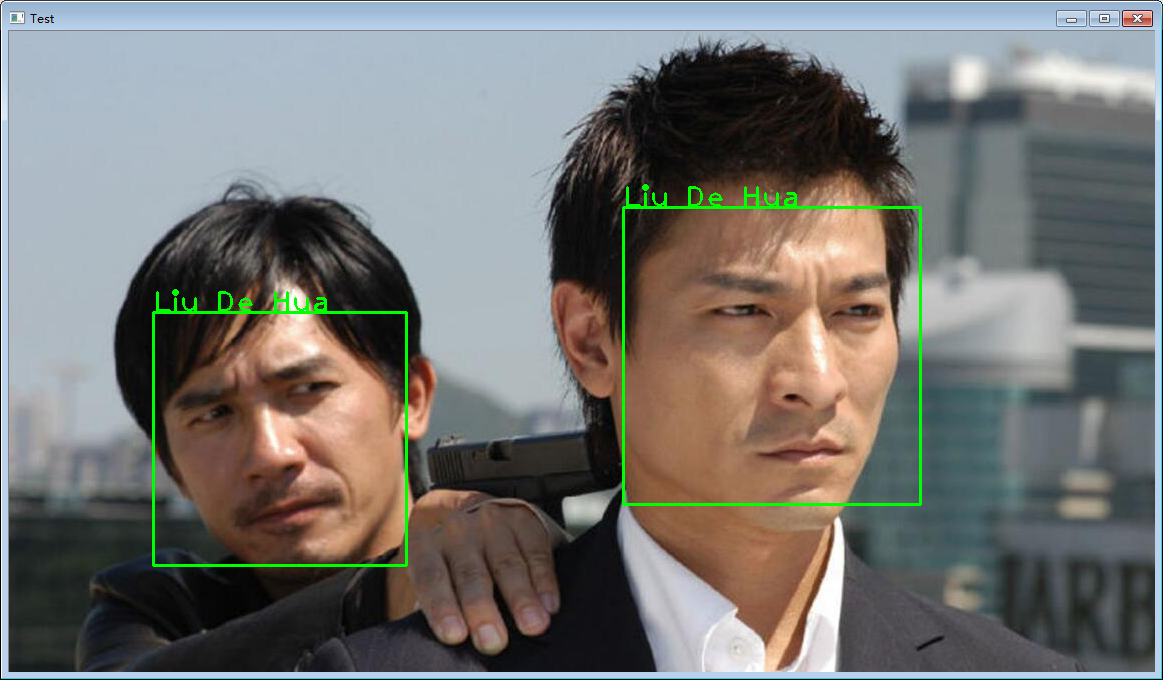
梁朝伟：



刘德华：



测试结果：



人脸识别的结果显然不正确，问题在哪里还没有调试明白，但时间已太晚，后续再有时间调试了。