OpenCV实战：人脸关键点检测（FaceMark）

原创：Amusi CVer 2018-04-08

Summary：利用OpenCV中的LBF算法进行人脸关键点检测（Facial Landmark Detection）
Author： Amusi
Date： 2018-03-20
Note： OpenCV3.4以及上支持Facemark

PS：**点击“阅读原文”，可以下载所有源码和模型**，记得给star哦！

教程目录

测试环境

- Windows10
- Visual Studio 2013
- OpenCV3.4.1

引言

人脸一般是有68个关键点，常用的人脸开源库有Dlib，还有很多深度学习的方法。

本教程仅利用OpenCV，不依赖任何其它第三方库来实现人脸关键点检测，这一特性是之前没有的。因为OpenCV自带的samples中只有常见的人脸检测、眼睛检测和眼镜检测等（方法是harr+cascade或lbp+cascade）。

本教程主要参考Facemark : Facial Landmark Detection using OpenCV[1]

截止到2018-03-20，OpenCV3.4可支持三种人脸关键点检测，但目前只能找到一种已训练好的模型，所以本教程只介绍一种实现人脸关键点检测的算法。而且此类算法还没有Python接口，所以这里只介绍C++的代码实现。

Facemark API

OpenCV官方的人脸关键点检测API称为Facemark。Facemark目前分别基于下述三篇论文，实现了三种人脸关键点检测的方法。

- FacemarkKazemi[2]: This implementation is based on a paper titled "One Millisecond Face Alignment with an Ensemble of Regression Trees" by V.Kazemi and J. Sullivan published in CVPR 2014[3]. An alternative implementation of this algorithm can be found in DLIB

- FacemarkAAM[4]: This implementation uses an Active Appearance Model (AAM) and is based on an the paper titled "Optimization problems for fast AAM fitting in-the-wild" by G. Tzimiropoulos and M. Pantic, published in ICCV 2013[5].

- FacemarkLBF[6]: This implementation is based a paper titled "Face alignment at 3000 fps via regressing local binary features" by S. Ren published in CVPR 2014[7].

在写这篇文章的时候，FacemarkKazemi类似乎不是从Facemark类派生的，而其他两个类都是。

Facemark训练好的模型

尽管Facemark API包含三种不同的实现，但只有FacemarkLBF（local binary features，LBF）才提供经过训练的模型。（之后在我们根据公共数据集训练我们自己的模型后，这篇文章将在未来更新）
你可以从中下载已训练好的模型：

- lbfmodel.yaml[8]

利用OpenCV代码进行实时人脸关键点检测

步骤

1. 加载人脸检测器（face detector）
所有的人脸关键点检测算法的输入都是一个截切的人脸图像。因为，我们的第一步就是在图像中检测所有的人脸，并将所有的人脸矩形框输入到人脸关键点检测器中。这里，我们可以使用OpenCV的Haar人脸检测器或者lbp人脸检测器来检测人脸。

2. 创建Facemark对象
创建Facemark类的对象。在OpenCV中，Facemark是使用智能指针（smart pointer，PTR），所以我们不需要考虑内存泄漏问题。

3. 加载landmark检测器
加载关键点检测器（lbfmodel.yaml）。此人脸检测器是在几千幅带有关键点标签的人脸图像上训练得到的。

带有注释/标签关键点的人脸图像公共数据集可以访问这个链接下载：https://ibug.doc.ic.ac.uk/resources/facial-point-annotations/

4.从网络摄像头中捕获帧

捕获视频帧并处理。我们既可以打开一个本地视频(.mp4)，也可以打开网络摄像机（如果电脑有的话）来进行人脸关键点检测。

5. 检测人脸

我们对视频的每一帧运行人脸检测器。人脸检测器的输出是一个包含一个或多个矩形（rectangles）的容器（vector），即视频帧中可能有一张或者多张人脸。

6. 运行人脸关键点检测器

我们根据人脸矩形框截取原图中的人脸ROI，再利用人脸关键点检测器（facial landmark detector）对人脸ROI进行检测。

对于每张脸我们获得，我们可以获得68个关键点，并将其存储在点的容器中。因为视频帧中可能有多张脸，所以我们应采用点的容器的容器。

7. 绘制人脸关键点

根据获得关键点，我们可以在视频帧上绘制出来并显示。

代码

本教程的代码一共有两个程序，分别为faceLandmarkDetection.cpp和drawLandmarks.hpp。

- faceLandmarkDetection.cpp实现视频帧捕获、人脸检测、人脸关键点检测；
- drawLandmarks.hpp实现人脸关键点绘制和多边形线绘制。

faceLandmarkDetection.cpp

```cpp
// Summary: 利用OpenCV的LBF算法进行人脸关键点检测
// Author: Amusi
// Date:  2018-03-20
// Reference:
//     [1]Tutorial: https://www.learnopencv.com/facemark-facial-landmark-detection-using-opencv/
//     [2]Code: https://github.com/spmallick/learnopencv/tree/master/FacialLandmarkDetection

// Note: OpenCV3.4以及上支持Facemark

#include <opencv2/opencv.hpp>
#include <opencv2/face.hpp>
#include "drawLandmarks.hpp"


using namespace std;
using namespace cv;
using namespace cv::face;

```

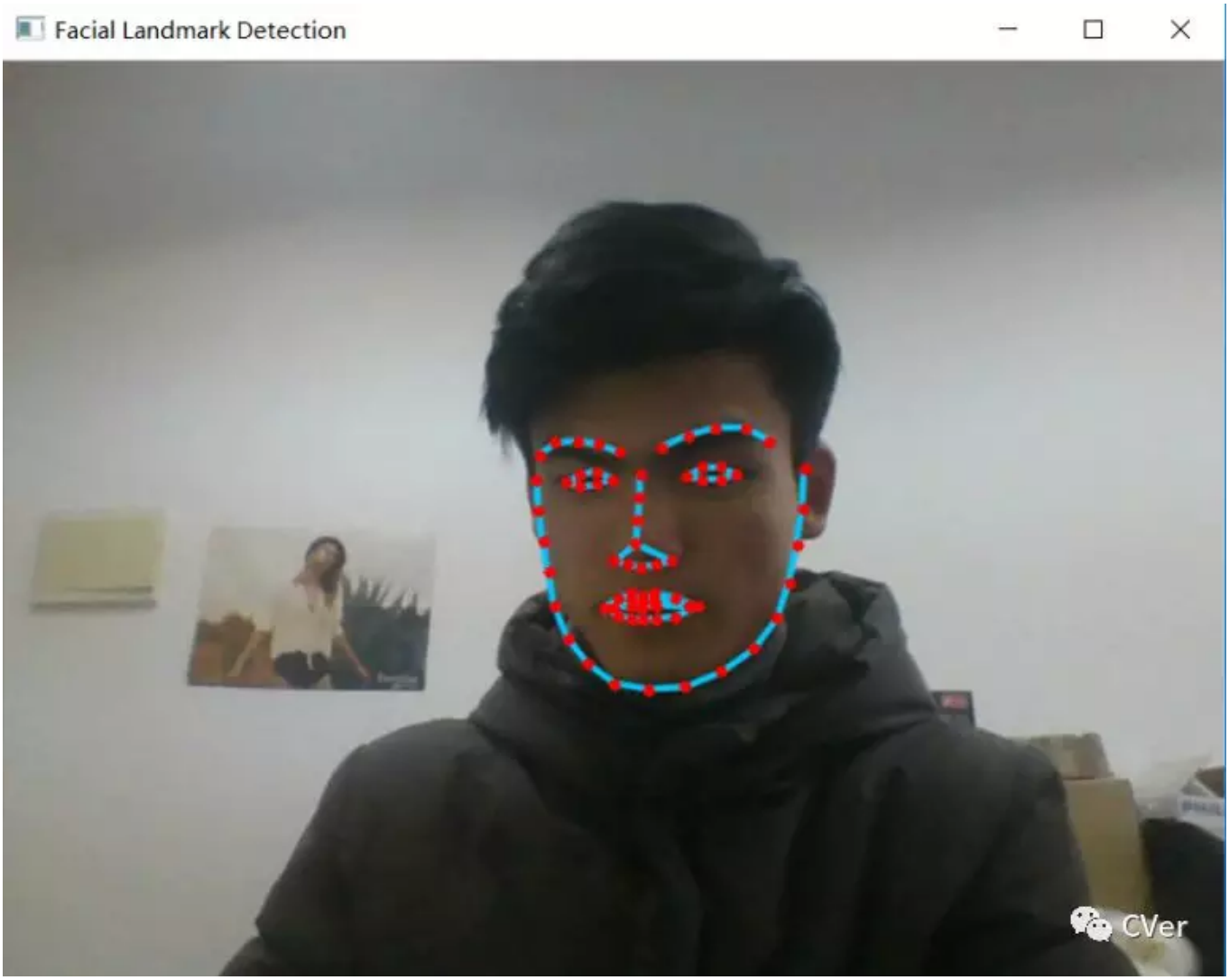
```cpp

int main(int argc,char** argv)
{
  // 加载人脸检测器（Face Detector）
  // [1]Haar Face Detector
  //CascadeClassifier faceDetector("haarcascade_frontalface_alt2.xml");
  // [2]LBP Face Detector
  CascadeClassifier faceDetector("lbpcascade_frontalface.xml");

  // 创建Facemark类的对象
  Ptr<Facemark> facemark = FacemarkLBF::create();

  // 加载人脸检测器模型
  facemark->loadModel("lbfmodel.yaml");

  // 设置网络摄像头用来捕获视频
  VideoCapture cam(0);

  // 存储视频帧和灰度图的变量
  Mat frame, gray;

  // 读取帧
  while(cam.read(frame))
  {

   // 存储人脸矩形框的容器
   vector<Rect> faces;
   // 将视频帧转换至灰度图，因为Face Detector的输入是灰度图
   cvtColor(frame, gray, COLOR_BGR2GRAY);

   // 人脸检测
   faceDetector.detectMultiScale(gray, faces);

   // 人脸关键点的容器
   vector< vector<Point2f> > landmarks;

   // 运行人脸关键点检测器（Landmark detector）
   bool success = facemark->fit(frame,faces,landmarks);

   if(success)
   {
    // 如果成功，在视频帧上绘制关键点
    for(int i = 0; i < landmarks.size(); i++)
    {
      // 自定义绘制人脸特征点函数，可绘制人脸特征点形状/轮廓
      drawLandmarks(frame, landmarks[i]);
      // OpenCV自带绘制人脸关键点函数：drawFacemarks
      drawFacemarks(frame, landmarks[i], Scalar(0, 0, 255));
    }

   }

   // 显示结果
   imshow("Facial Landmark Detection", frame);

   // 如果按下ESC键，则退出程序
   if (waitKey(1) == 27) break;

  }
  return 0;
}
```

drawLandmarks.hpp

```cpp
// Summary: 绘制人脸关键点和多边形线
// Author: Amusi
```

```
// Date:  2018-03-20

#ifndef _renderFace_H_
#define _renderFace_H_

#include <iostream>
#include <opencv2/opencv.hpp>

using namespace cv;
using namespace std;

#define COLOR Scalar(255, 200,0)

// drawPolyline通过连接开始和结束索引之间的连续点来绘制多边形线。
void drawPolyline
(
 Mat &im,
 const vector<Point2f> &landmarks,
 const int start,
 const int end,
 bool isClosed = false
)
{
  // 收集开始和结束索引之间的所有点
  vector <Point> points;
  for (int i = start; i <= end; i++)
  {
    points.push_back(cv::Point(landmarks[i].x, landmarks[i].y));
  }

  // 绘制多边形曲线
  polylines(im, points, isClosed, COLOR, 2, 16);

}

// 绘制人脸关键点
void drawLandmarks(Mat &im, vector<Point2f> &landmarks)
{
  // 在脸上绘制68点及轮廓（点的顺序是特定的，有属性的）
  if (landmarks.size() == 68)
  {
   drawPolyline(im, landmarks, 0, 16);       // Jaw line
   drawPolyline(im, landmarks, 17, 21);      // Left eyebrow
   drawPolyline(im, landmarks, 22, 26);      // Right eyebrow
   drawPolyline(im, landmarks, 27, 30);      // Nose bridge
   drawPolyline(im, landmarks, 30, 35, true);  // Lower nose
   drawPolyline(im, landmarks, 36, 41, true);  // Left eye
   drawPolyline(im, landmarks, 42, 47, true);  // Right Eye
   drawPolyline(im, landmarks, 48, 59, true);  // Outer lip
   drawPolyline(im, landmarks, 60, 67, true);  // Inner lip
  }
  else
  {
    // 如果人脸关键点数不是68，则我们不知道哪些点对应于哪些面部特征。所以，我们为每个Landamrk画一个圆圈。
    for(int i = 0; i < landmarks.size(); i++)
    {
      circle(im,landmarks[i],3, COLOR, FILLED);
    }
  }

}

#endif // _renderFace_H_
```

实验结果



Reference

[1]Tutorial : https://www.learnopencv.com/facemark-facial-landmark-detection-using-opencv/
[2]FacemarkKazemi : https://docs.opencv.org/trunk/dc/de0/classcv_1_1face_1_1FacemarkKazemi.html
[3]One Millisecond Face Alignment with an Ensemble of Regression Trees : http://www.csc.kth.se/~vahidk/face_ert.html
[4]FacemarkAAM : https://docs.opencv.org/trunk/d5/d7b/classcv_1_1face_1_1FacemarkAAM.html
[5]Optimization problems for fast AAM fitting in-the-wild : https://ibug.doc.ic.ac.uk/media/uploads/documents/tzimiro_pantic_iccv2013.pdf
[6]FacemarkLBF : https://docs.opencv.org/trunk/dc/d63/classcv_1_1face_1_1FacemarkLBF.html

[7]Face alignment at 3000 fps via regressing local binary features : http://www.jiansun.org/papers/CVPR14_FaceAlignment.pdf
[8]lbfmodel.yaml : https://github.com/kurnianggoro/GSOC2017/blob/master/data/lbfmodel.yaml

-----我是可爱的分割线-------

若喜欢Amusi写的文章，可以扫描下方二维码关注CVer公众号！

如何下载教程中的所有源码和模型？
请点击下面的“阅读原文”，
记得给star哦！

阅读原文