Main User Detection and Recognition

Yicong Wu
Carnegie Mellon University
5000 Forbes Ave, Pittsburgh, PA 15213
yicongw@andrew.cmu.edu

Sibo Wang
Carnegie Mellon University
5000 Forbes Ave, Pittsburgh, PA 15213
sibowang@andrew.cmu.edu

Abstract

This project can detect the faces in the front-facing camera in real-time and mark them with yellow rectangle, and recognize the main user's face among them with a red rectangle. If the main user is the registered one, the color of rectangle will change to blue.

1. Introduction

This project mainly completes the face detection and recognition tasks. We use three methods to implement face detection task including OpenCV, Core Image and GPUImage. For the recognition part, we use a machine learning method with local binary patterns histogram technology to train the sample image and do prediction to each detected face. In the end, we do tests on the three versions of program in order to select the most robust and efficient version as the final results.

This project is a part of Global Learning XPRIZE competition aiming to develop an open-source app on mobile devices that enables children 7-10 to learn reading, writing, and numeracy – in their native language of Swahili, so detecting the main user in the camera, and recognizing if it is the registered user is necessary. This can be trivially implemented in OpenCV, but the speed is really low and the CPU is fully occupied. So how to achieve it in real-time efficiently is meaningful and challenging.

2. Background

We use OpenCV, CoreImage and GPUImage to solve the detection task.

OpenCV (Open Source Computer Vision) is a library of programming functions mainly aimed at real-time computer vision. There are many convenient video processing classes. CvVideoCamera is easy to use for returning video stream from the camera, and programmer can implement image processing tasks on each frame. CascadeClassifier can do detection for many facial features.

CoreImage is a pixel-accurate, non-destructive image processing technology in Mac OS and IOS. It assembles the code for this instruction pipeline with a just-in-time compiler, which is executed by either the CPU or graphics card's GPU, whichever can perform the calculation faster. In CoreImage, we can use CIDetector to detect faces. The result of the detector is CIFaceFeature which describes a face detected in a still or video image.

GPUImage is a BSD-licensed iOS library that lets you apply GPU-accelerated filters and other effects to images, live camera video, and movies. For massively parallel operations like processing images or live video frames, GPUs have some significant performance advantages over CPUs. Recently, an extension to GPUImage was proposed to allow for the utilization of iOS face detection within iOS. So we finally choose this efficient frame to do face detection.

3. Approach

3.1. Detection

Object Detection in OpenCV using Haar feature-based cascade classifiers which is method proposed by Paul Viola and Michael Jones in their paper, "Rapid Object Detection using a Boosted Cascade of Simple Features" in 2001[1].

Initially, the algorithm needs a lot of positive images and negative images to train the classifier. Haar features are used to extract features from it. Each feature is a single value obtained by subtracting sum of pixels under white rectangle from sum of pixels under black rectangle.

In order to accelerate this process, integral image is introduced here. Then features with minimum error rate are selected, which means they are the features that best classifies the face and non-face images.

The final classifier is a weighted sum of these weak classifiers, and all features are separated into different stages. If a window fails the first stage, discard it. We don't consider remaining features on it. If it passes, apply the second stage of features and continue the process. The window which passes all stages is a face region.

(a) edge features (b)line features (c)four-rectangle features

Figure 1: Example of features.

For the detection in CoreImage provided by Apple, we could not know the algorithm in detail, but compared to CascadeClassifier in OpenCV, it is really faster. So for the GPUImage, it also uses CIDetector to detect faces.

3.2. Recognition

LBPH recognizer is an algorithmic level solution. It can remain both the local textures and spatial information. The process of feature extraction is simple and efficient. Besides, this feature extraction method is robust with intensity and scale changes.

Local Binary Patterns Histograms[2]:

The basic idea of LBP is to use the pixel as the center in order to compare it with its neighborhood. If the intensity of the central pixel is larger than that of the pixel next to it, we label the neighbor 1. If not, we label it 0. Thus, we can record the 8 neighbor pixels as a set of binary code, for example: 11011101 so called LBP code. We can get 128 different kinds of codes. This mode of management is called local binary mode.

A formal LBP operation can be defined as:

$$LBP(x_c, y_c) = \sum_{p=0}^{P-1} 2^p s(i_p - i_c)$$

(x_c, y_c) is central pixel, i_c is its intensity. S(t) is a function that returns 1 if t is negative.

We can get the textures in images by this method. In fact, this is a latest and advanced method for researchers to get the texture information. However, the LBP codes in fixed region become defective because of scaling. So, one way for extension is to use a changeable circle to mark the neighborhood that will be encoded.

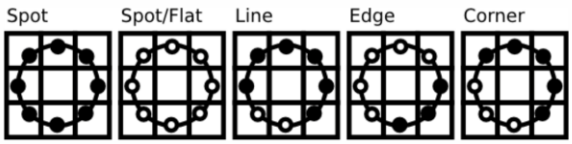


Figure 2: Circular LBP.

For a specific pixel (x_c, y_c), its neighbor pixel (x_p, y_p) can be decided by the following equations:

$$x_p = x_c + R\cos(\frac{2\pi p}{P})$$

$$y_p = y_c - R\sin(\frac{2\pi p}{P})$$

R is the radius of the circle, and P is the number of sample pixels.

This is an extension for the original LBP operator, so this is called extended LBP or circular LBP. If the pixel on the circle is not on the image, an interpolation value will be used. LBP operator is robust for intensity change on the whole image.

Spatial pyramid matching[3]:

The next problem is how to remain the spatial information of the image. Spatial pyramid matching model is a suitable solution to this problem. We simply divide the whole image into small blocks. For each block, we make a histogram of the LBP code. Then, we combine these histograms into a long histogram vector. Finally, we get the features of image with spatial arrangement.

LBPH recognizer:

The LBPH recognizer contains two parts: training and predicting.

Training process:

i. Get the face image and its label;
ii. Generate LBP image;
iii. Generate LBP histogram vector;
iv. Store the LBP histogram vector;

Predicting process:

i. Get the target face;
ii. Generate LBP image;
iii. Generate LBP histogram vector;
iv. Calculate the difference between sample and target;
v. Find the most similar sample;
vi. Mark the target face with the same label;

4. Results

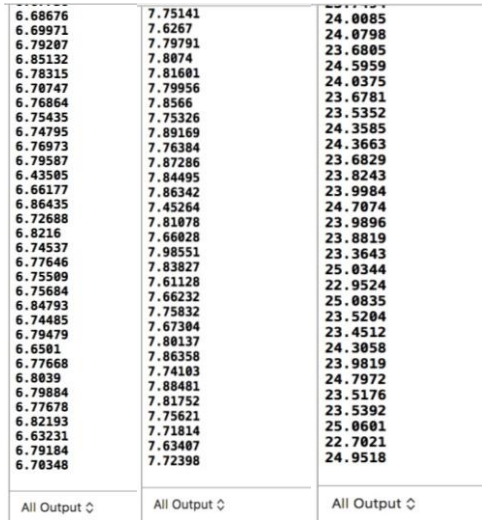
We test our application with respect to its accuracy and efficiency. All the tests are carried out in iPad air 2. Its CPU is A8X(64 bits).

For accuracy part, as we just use one frontal face as the training set, we find that recognize the registered user in most of the situations. But it cannot recognize the profile of the same person unless we add the sample profile into the training set.

Figure 3: training set and result.

For efficiency, comparisons among the three versions are listed below. We separately run these three programs and output their current numbers of frames per second in order to know how fast each program is. We set the original camera the same 30 frames per second. We want to know how much time each program takes to process each frame and display the result.

Here is the result:



(a) OpenCV (b)Core Image (c)GPUImage

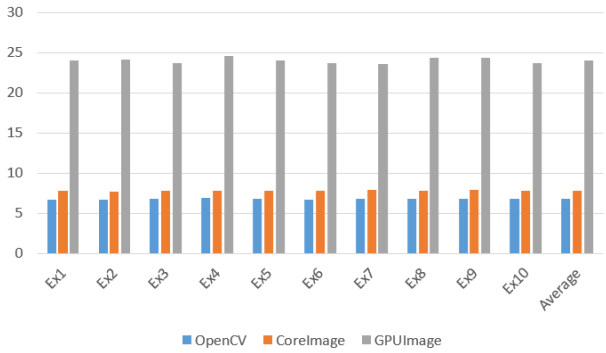


Figure 4: results of frame per second test.

	OpenCV	CoreImage	GPUImage
Ex1	6.68676	7.75141	24.0085
Ex2	6.69971	7.6267	24.0798
Ex3	6.79207	7.79791	23.6805
Ex4	6.85132	7.8074	24.5959
Ex5	6.78315	7.81601	24.0375
Ex6	6.70747	7.79956	23.6781
Ex7	6.76864	7.8566	23.5352
Ex8	6.75435	7.75326	24.3585
Ex9	6.74795	7.89169	24.3663
Ex10	6.76973	7.76383	23.6829
Average	6.756115	7.786437	24.00232

Figure 5: statistical results of frame per second test.

In conclusion, we find out that the GPU based program is fastest one among the three versions.

Analysis:

We also use ‘Instrument’ as a tool to see the differences among the three versions.

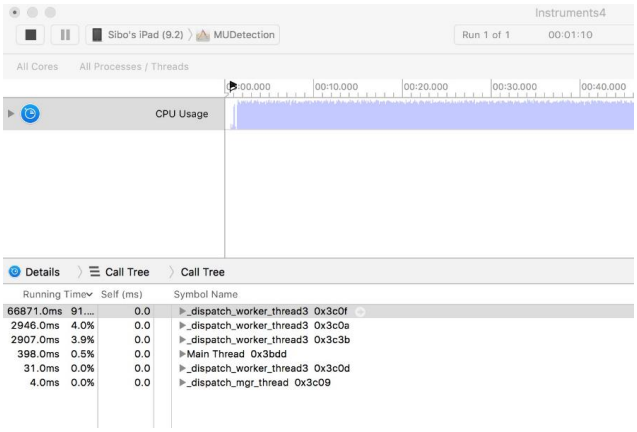


Figure 6: instrument statistics of OpenCV.

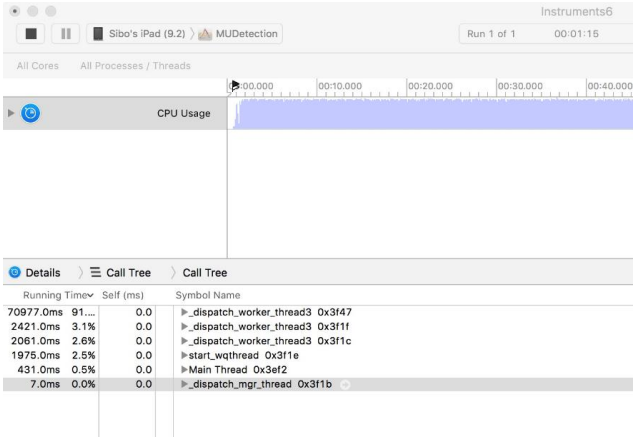


Figure 7: instrument statistics of CoreImage.

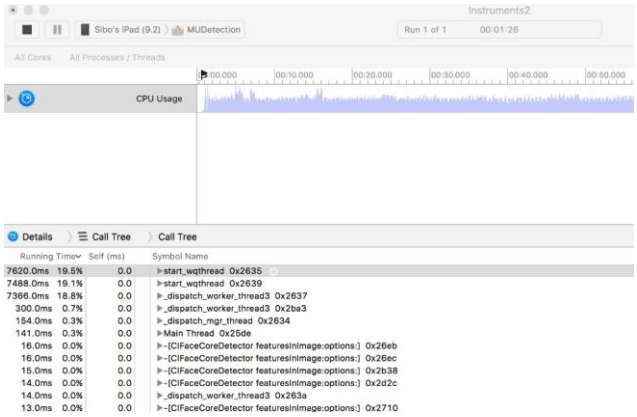


Figure 8: instrument statistics of GPUImage.

We can see that CPU takes most of the works in OpenCV and Core Image version. And only one thread is working during the whole process. However, in GPUImage version, the whole task has been allocated to several threads. As parallel processing is the most prominent advantage of GPU, it takes less time for GPU to finish the face detection task and draw rectangles on each frame.

5. Conclusion

In this paper, we have implement face detection and face recognition in iOS system by GPUImage. It is a much faster method than that in OpenCV. The GPUImage version is approximately three times faster than the other two versions and CPU has lower usage. Although there are only a few resources in GPU framework, we are happy to see that GPUs have the excellent advantage that GPUs can do parallel processing much more efficient. Moreover, it is easy for developers to design their own filters and just put filters in a daisy chain. Next, we hope to find how to implement face recognition task by GPUImage so that this can further speed up the whole program.

6. List of Work

Equal work was performed by both project members.

a. Do research on recognizing method
b. Face detection in OpenCV
c. Face detection in CoreImage
d. Raw data extraction of GPUImage
e. Face recognition
f. Efficiency test
g. Instruments based test
h. Proposal write up
i. Checkpoint write up
j. Final write up

7. Github Page

https://github.com/yicongwu/MUDetection.git

We implemented our project totally in OpenCV till the checkpoint. After that, we revised our codes using CoreImage and GPUImage. Now three versions of codes are updated on the github.

References

[1] Viola P, Jones M. Rapid object detection using a boosted cascade of simple features[C]//Computer Vision and Pattern Recognition, 2001. CVPR 2001. Proceedings of the 2001 IEEE Computer Society Conference on. IEEE, 2001, 1: I-511-I-518 vol. 1.

[2] Simon J.D. Prince. Computer vision: models, learning and inference. July 7,2012.

[3] Lazebnik, Svetlana, Cordelia Schmid, and Jean Ponce. "Beyond bags of features: Spatial pyramid matching for recognizing natural scene categories." Computer Vision and Pattern Recognition, 2006 IEEE Computer Society Conference on. Vol. 2. IEEE, 2006.