FACE DETECTION

TUAN VU BUI (BUI0016)

May 21,2021

1 Introduction

Face detection – also called facial detection – is an artificial intelligence (AI) based computer technology used to find and identify human faces in digital images.
Face detection technology can be applied to various fields – including security, biometrics, law enforcement, entertainment and personal safety – to provide surveillance and tracking of people in real time.
Face detection has progressed from rudimentary computer vision techniques to advances in machine learning (ML) to increasingly sophisticated artificial neural networks (ANN) and related technologies; the result has been continuous performance improvements.
It now plays an important role as the first step in many key applications – including face tracking, face analysis and facial recognition. Face detection has a significant effect on how sequential operations will perform in the application. algorithms that discern which parts of an image or video are needed to generate a faceprint. Once identified, the new faceprint can be compared with stored faceprints to determine if there is a match.
Today, I will use HOG algorithm to solving the facial detection.

2 The algorithm

1. Step 1: Preprocessing.
   HOG feature descriptor used for pedestrian detection is calculated on a 64×128 patch of an image. Of course, an image may be of any size. Typically patches at multiple scales are analyzed at many image locations. The only constraint is that the patches being analyzed have a fixed aspect ratio.

2. Step 2: Calculate the Gradient Images.
   To calculate a HOG descriptor, we need to first calculate the horizontal and vertical gradients; after all, we want to calculate the histogram of gradients. This is easily achieved by filtering the image with the following kernels.

   $$\begin{array}{|c|c|c|}\hline -1 & 0 & 1 \\ \hline \end{array} \qquad \begin{array}{|c|}\hline -1 \\ \hline 0 \\ \hline 1 \\ \hline \end{array}$$

   Next, we can find the magnitude and direction of gradient using the following formula:

   $$g = \sqrt{g_x^2 + g_y^2}$$
   $$\theta = \arctan \frac{g_y}{g_x}$$

   Notice, the x-gradient fires on vertical lines and the y-gradient fires on horizontal lines. The magnitude of gradient fires where ever there is a sharp change in intensity. None of them fire when the region is smooth. I have deliberately left out the image showing the direction of gradient because direction shown as an image does not convey much.

The gradient image removed a lot of non-essential information (e.g. constant colored background), but highlighted outlines. In other words, you can look at the gradient image and still easily say there is a person in the picture.
At every pixel, the gradient has a magnitude and a direction. For color images, the gradients of the three channels are evaluated (as shown in the figure above). The magnitude of gradient at a pixel is the maximum of the magnitude of gradients of the three channels, and the angle is the angle corresponding to the maximum gradient.

3. Step 3: Calculate Histogram of Gradients in 8×8 cells
   In this step, the image is divided into 8×8 cells and a histogram of gradients is calculated for each 8×8 cells. But why 8×8 patch ? Why not 32×32 ? It is a design choice informed by the scale of features we are looking for. HOG was used for pedestrian detection initially. 8×8 cells in a photo of a pedestrian scaled to 64×128 are big enough to capture interesting features (e.g. the face, the top of the head etc.)

4. Step 4: 16×16 Block Normalization Ideally, we want our descriptor to be independent of lighting variations. In other words, we would like to "normalize" the histogram so they are not affected by lighting variations. A 16×16 block has 4 histograms which can be concatenated to form a 36 x 1 element vector and it can be normalized just the way a 3×1 vector is normalized. The window is then moved by 8 pixels (see animation) and a normalized 36×1 vector is calculated over this window and the process is repeated.

5. Step 5: Calculate the Histogram of Oriented Gradients feature vector

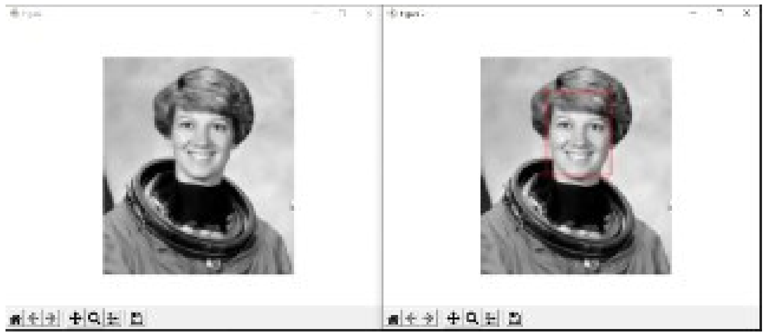
   - How many positions of the 16×16 blocks do we have ? There are 7 horizontal and 15 vertical positions making a total of 7 x 15 = 105 positions.
   - Each 16×16 block is represented by a 36×1 vector. So when we concatenate them all into one gaint vector we obtain a 36×105 = 3780 dimensional vector

3 Benchmarks

For the benchmarks, I am using a set of positive training samples by import $fetch_lfw_people$ from *sklearn.datasets* library. It contains a sample of 13,000 face images to use for training. Also, we need a set of similarly sized thumbnails which do not have a face in them.
One way to do this is to take any corpus of input images, and extract thumbnails from them at a variety of scales. Here we can use some of the images shipped with Scikit-Image, along with Scikit-Learn's PatchExtractor.
This training part takes so much time so I decide to use multiprocessors to reduce the execution time.
For the speed of algorithm, I am measuring two different times: CPU Time. Sum of time on all threads in ms. Real Time. A real time from the beginning of the algorithm until the algorithm is finished in ms

Type	One Proccessor	Three Proccessors	Five Proccessors
CPU Time[ms]	242.3558	90.4707	64.8533
Real Time[ms]	366.9420	215.3566	189.8882

After loading data and training, we will apply it to detect the face in the image. These images below will show:
The left-side image is from the data that we loaded and trained from the previous step.
The right-side image is the result after detecting face



4 Conclusion

Although the execution time is declined when I used multiprocessors for loading and training data, the increase in difficult of the task is significant. Either, the parallelization needs to be better optimised.
In my implementation, I did not use a dlib library, I tried to implement in another way because dlib library supports everything about facial detection topic for us and I do not wanna dependent on dlib. From that reason, If you want to improve the execution time even more also results are more accurate, dlib library is the best choice for solving face detection by HOG Algorithm.