Department of Electrical and Computer Systems Engineering
Monash University

Multimedia Technologies, ECE4146

Lab 10: Pattern Matching Viola-Jones Algorithm

Chi Bach Pham

28961129

Introduction

This report discusses the investigation and experiments with the Viola-Jones face detection algorithm. The Viola-Jones algorithm is very robust process and is still in used widely after two decades from it releases.

The algorithm extracts rectangles of features from the images. Each rectangle features are divided into dark and light region where the pixels in each region is added up and then the sum of the dark region would subtract the sum of the light region. The result from the subtraction would give us an idea of how difference the area in these regions are to each other and decide that there is some feature in that rectangle features. For example, the eyes and the upper check usually have different brightness and would make a feature with large value. There are a lot of these feature in an image and the algorithm would use AdaBoost to determine which features is more significant. Then, we would have a cascade structure where it determine the images in stages, we group a first few significant feature in a group and if the image does not satisfy those features, it is then discard immediately, which make the process much faster as it quickly discard any irrelevant features.

Objective

The drawback of usual Viola-Jones is that it requires full view frontal faces, which mean the face would not be detect if it is tilt to the side or facing the camera at an angle. My suspected reason for this from what I learned about the algorithm is that it needs the feature of object to have similar orientation and relative location, and the frontal face constraints make the most sense as that would be the usual position for a person face to be detect.

Hence, what I set out to explore is if this is only a matter of design choice and if we train a classifier with tilted face data, it can then detect the tilted face, but maybe not able to detect full frontal face like it normally would. And the limitations is dependent entirely on the orientation of feature.

Procedure

I collect face data from http://vis-www.cs.umass.edu/lfw/ then I would run a MATLAB script to tilt the images.

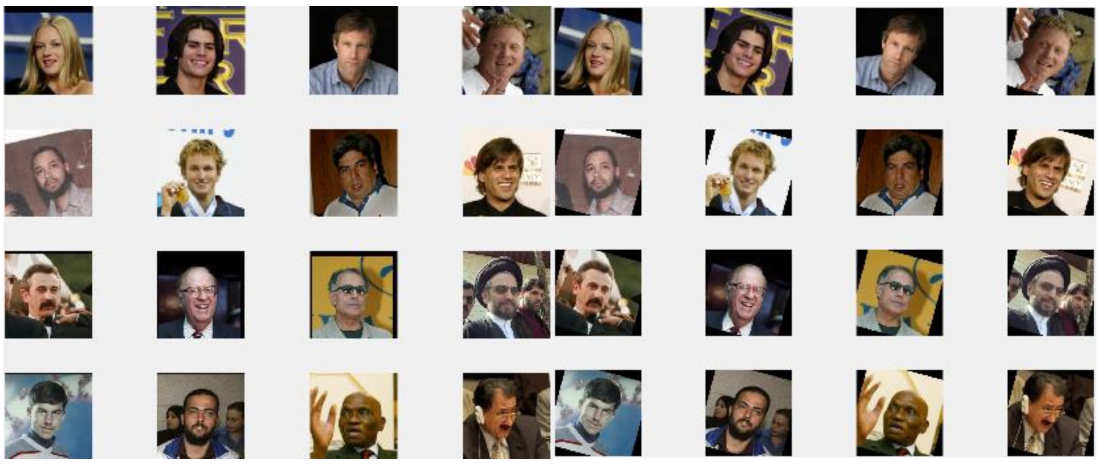

Figure 1: Original (left) and tilted images (right)

Then we need to collect data to form a negative, which is the images not containing faces, I downloaded that from https://github.com/JoakimSoderberg/haarcascade-negatives/tree/master/images . I labelled these 100 of these images by hand using MATLAB Image Labeller, extract the content from their bounding box. And together with the negatives I can run `trainCascadeObjectDetector` to train the detector, which produce a xml file that describe the detector.

Result

My detector is not as robust as the pre-trained one as it uses only a small dataset, but under good conditions like having a clean background and proper lighting on the faces it can detect my tilted face from the webcam. We can see below how it does not detect when I have my head in frontal versus when I tilted slightly to match the training data, the third images is from when I implemented motion tracking, the bounding box would rotate when I tilt my face up right again as it recognize the tilted faces as the correct orientation for this dataset.

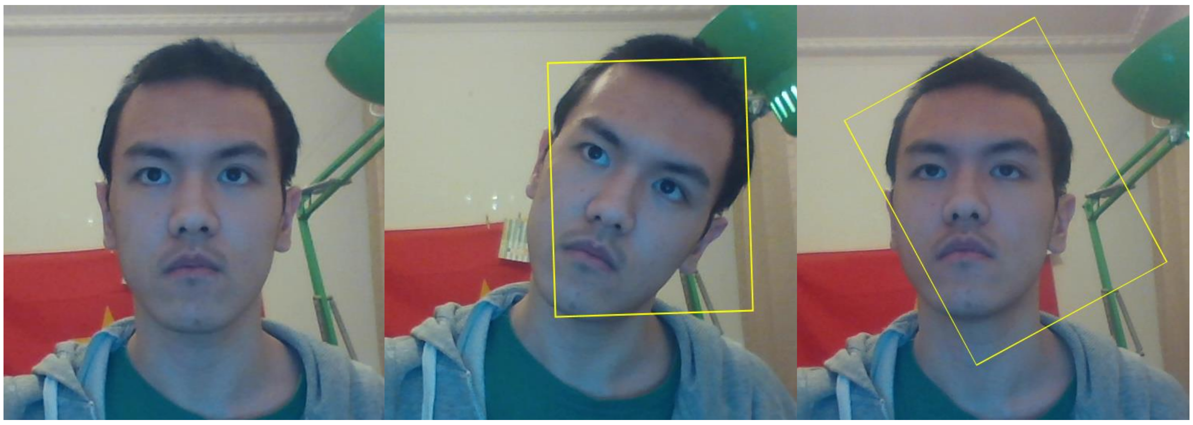

Figure 2: Face detection, Frontal (Left), Tilted (Centre), Frontal after tilting(right).

Conclusion

Even with a ridiculously small dataset, I can still get the detector to work, which prove the versatility of the algorithm. As well as I can confirm my theory that the detector can detect anything given a suitable dataset.

Appendix

MATLAB Code

```
close all,clear all,clc;
data = load('facedata.mat');
gTruth = data.gTruth;
data = objectDetectorTrainingData(gTruth);
positiveInstances = data(:,1:2);
negativeFolder = fullfile('NEGATIVE');
negativeImages = imageDatastore(negativeFolder);
%trainCascadeObjectDetector('faces.xml',positiveInstances,negativeFolder,'FalseAlarmRate',0.1,
'NumCascadeStages',5);

faceDetector = vision.CascadeObjectDetector('faces.xml');

% Create the point tracker object.
pointTracker = vision.PointTracker('MaxBidirectionalError', 2);

clear cam;
cam = webcam();
videoFrame = snapshot(cam);
frameSize = size(videoFrame);

% Create the video player object.

videoPlayer = vision.VideoPlayer('Position', [100 100 [frameSize(2), frameSize(1)]+30]);
runLoop = true;
numPts = 0;
frameCount = 0;

while runLoop

    % Get the next frame.
    videoFrame = snapshot(cam);
    videoFrameGray = rgb2gray(videoFrame);
    frameCount = frameCount + 1;

    if numPts < 10
        % Detection mode.
        bbox = faceDetector.step(videoFrameGray);

        if ~isempty(bbox)
            % Find corner points inside the detected region.
            points = detectMinEigenFeatures(videoFrameGray, 'ROI', bbox(1, :));

            % Re-initialize the point tracker.
            xyPoints = points.Location;
            numPts = size(xyPoints,1);
            release(pointTracker);
            initialize(pointTracker, xyPoints, videoFrameGray);

            % Save a copy of the points.
            oldPoints = xyPoints;

            % Convert the rectangle represented as [x, y, w, h] into an
            % M-by-2 matrix of [x,y] coordinates of the four corners. This
            % is needed to be able to transform the bounding box to display
            % the orientation of the face.
            bboxPoints = bbox2points(bbox(1, :));

            % Convert the box corners into the [x1 y1 x2 y2 x3 y3 x4 y4]
            % format required by insertShape.
            bboxPolygon = reshape(bboxPoints', 1, []);

            % Display a bounding box around the detected face.
            videoFrame = insertShape(videoFrame, 'Polygon', bboxPolygon, 'LineWidth', 3);

            % Display detected corners.
            %videoFrame = insertMarker(videoFrame, xyPoints, '+', 'Color', 'white');
        end

    else
        % Tracking mode.
        [xyPoints, isFound] = step(pointTracker, videoFrameGray);
        visiblePoints = xyPoints(isFound, :);
        oldInliers = oldPoints(isFound, :);

        numPts = size(visiblePoints, 1);

        if numPts >= 10
            % Estimate the geometric transformation between the old points
            % and the new points.
             [xform, inlierpoints1,inlierpoints2] = estimateGeometricTransform(oldInliers,
visiblePoints, 'similarity', 'MaxDistance', 10);
            oldInliers    = inlierpoints1;
            visiblePoints = inlierpoints2;

            % Apply the transformation to the bounding box.
            bboxPoints = transformPointsForward(xform, bboxPoints);

            % Convert the box corners into the [x1 y1 x2 y2 x3 y3 x4 y4]
            % format required by insertShape.
            bboxPolygon = reshape(bboxPoints', 1, []);

            % Display a bounding box around the face being tracked.
            videoFrame = insertShape(videoFrame, 'Polygon', bboxPolygon, 'LineWidth', 3);

            % Display tracked points.

            % Reset the points.
            oldPoints = visiblePoints;
            setPoints(pointTracker, oldPoints);
        end

    end

    % Display the annotated video frame using the video player object.
    step(videoPlayer, videoFrame);

    % Check whether the video player window has been closed.
    runLoop = isOpen(videoPlayer);
end

clear cam;
release(videoPlayer);
release(detector);
```