Pedestrian Detection Aided by Temporal Prior

Zhaowei Cai, Matthew Jacobsen and Nuno Vasconcelos
University of California, San Diego
9500 Gilman Drive
La Jolla, California, USA
zwcai@ucsd.edu, mdjacobs@cs.ucsd.edu, nuno@ucsd.edu

Abstract— **In this paper, we developed an algorithm to construct temporal priors to improve the stability of pedestrian detections. The proposed temporal prior has no additional computation cost and could be applied to any image based pedestrian detector. We also present a FPGA implementation that can process VGA video at a frame rate of 30~40 frames per second in real time. The design is implemented on a Xilinx Zynq FPGA. It performs all steps of the algorithm in the FPGA fabric including: color space conversion, frame rescaling, HOG feature extraction, and candidate evaluation in a sliding window. The design uses parallel pipelines to evaluate multiple scales per frame.**

Pedestrain detection; temporal prior; hardware design; FPGA

I. INTRODUCTION

Pedestrian detection is a very important research area, especially for driver assistance systems and autonomous vehicles. However, it is still very challenging since it has high requirements for speed and accuracy. Currently, the typical pedestrian detection process is based on individual images, even though the applications are usually on videos. There is a couple of consequent problems: 1) the detections are not stable, e.g. the bounding boxes are flickering; 2) the same object will be missed at current frame even though it is detected in previous frames. These problems happen because of no temporal constraint. In this paper, we introduce temporal priors to make the detections over continuous frames stable. The experiments have also shown substantial improvements by temporal priors. Another advantage of the proposed temporal prior is it has no additional computation cost and can be applied to any other pedestrian detectors. We have also implemented the pedestrian detector using a FPGA. Our implementation maintains the same level of detection accuracy as the C++ software version. It achieves a 2× speed up over the software version to provide detections between 30~40 frames per second (FPS) on VGA video.

II. ALGORITHMS

The pedestrian detector of [1] is used in this paper, which uses cascaded framework, runs in real time and is based on individual images.

A. Temporal Priors

A detection at time t is defined by a bounding box $(\mathrm{x}_t, \mathrm{y}_t, \mathrm{h}_t, \mathrm{w}_t)$. The estimated detection at time $t+1$ is

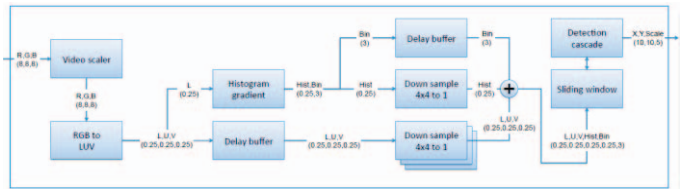

Fig. 1. The detection pipeline of hardware design.

$$\mathrm{x}_{t+1} = \mathrm{x}_t + \Delta \mathrm{x}_t \tag{1}$$

where $\Delta \mathrm{x}_t$ is the displacement at time t. The displacement depends on the speed of the object, which is $\Delta \mathrm{x}_t = \mathrm{x}_t - \mathrm{x}_{t-1}$.

The priors of this object present around x_{t+1} is subjected to a Gaussian distribution,

$$\mathrm{p}(\mathrm{x}' \mid \mathrm{x}_{t+1}) \sim N(\mathrm{x}_{t+1}, \sigma_{\mathrm{x}}^2).$$

The same model holds for the other coordinates. The final score at location $(\mathrm{x}, \mathrm{y}, \mathrm{h}, \mathrm{w})$ is

$$\mathrm{s}(\mathrm{x}, \mathrm{y}, \mathrm{w}, \mathrm{h}) = \mathrm{p}(\mathrm{x}, \mathrm{y}, \mathrm{w}, \mathrm{h}) * \mathrm{K} + \mathrm{s}_d(\mathrm{x}, \mathrm{y}, \mathrm{w}, \mathrm{h})$$

where p is the prior, K is a constant and s_d is the detection score of the detector of [1]. With temporal prior, the locations where objects were present will have higher scores.

B. Nearest Neighbor Tracking

Since the object speed is need in (1), a tracking algorithm is required to obtain $\Delta \mathrm{x}_t$. For an object detected in previous frame, there may be multiple detections which are candidate matches to the object. The tracking algorithm will help to find the best matched detection. The traditional single or multiple objects tracking algorithms could be used here, but most of them require expensive computational costs, which is inconsistent to our goal, real-time pedestrian detection. To save computation, we resort to nearest neighbor tracking algorithm. Given a detected object bounding box at time t, the tracked bounding box is

$$\mathrm{bb}_{t+1} = \arg\max_i (\mathrm{IOU}(\mathrm{bb}_t, \mathrm{bb}^i))$$

where bb^i is one candidate bounding box, and IOU is the "intersection over union" between two bounding boxes. The speed is known once two bounding boxes are matched.

This work is supported by the Technology Development Program for Commercializing System Semiconductor funded by the Ministry Of Trade, Industry and Energy (MOTIE, Korea). (No 10041126, Title: International Collaborative R&BD Project for System Semiconductor)

TABLE I. DETECTION EVALUATION

Method	detector (no prior)	detector (with prior)
MR	30.92%	26.47%
FPS	6	6

III. HARDWARE DESIGN

The hardware design implements our algorithm in single pass stream processing fashion, which is based on [2]. Fig. 1 is a block diagram of the detection data flow.

A. Video Scaler

The video scaler uses bi-linear interpolation to scale input video frames by an arbitrary factor. Input pixels are line buffered into on-chip RAM blocks and rescaled as they are read out. The interpolation is accomplished by calculating a set of coefficients for each pixel and using the surrounding pixel values to create an output pixel. The scaling parameters are also used to identify the number of pixels per line and total lines to output.

B. RBG to LUV Converter

The conversion between RGB and LUV requires linear combinations of the RGB components, multiplication, division, and cubed root operations. The cubed root operation was represented using a pre-calculated lookup table (LUT).

C. Histogram Gradient

The HOG features for each pixel are a measure of the magnitude and direction of the dominate gradient in a pixel. A pixel's value is determined by the value of the four surrounding pixels in the cardinal directions. We mapped the magnitude value directly to one of the 6 directions using a high precision LUT. We then found the 6 thresholds for entries in the LUT and encoded them directly into the module. Only square root operations were needed at runtime after simplification.

D. Sliding Window

Our implementation uses a set of line buffers to hold n pixel lines, the frame width. The value of n is determined by the height of the window (in our case 16). New pixels fill the buffers from the bottom up, pushing the old pixels out the top. This technique is similar to that used by Cho in his Viola-Jones FPGA implementation [3]. The key difference is that pixels are accessed from the line buffers directly instead of from a synchronously maintained register array.

E. Detection Cascade

Once features are extracted, they must be compared to thresholds. A weak learner contributes a value to a running score. The detection cascade handles reading feature data from the sliding window and cascade parameters (thresholds, scores, etc.) from a pre-loaded on-chip RAM. Comparisons are made, feature scores are calculated, and the running score is updated. Afterwards, it is compared to the current stage threshold to see if an early rejection is possible. This process will continue until all the stages have passed or the window is early rejected. If passed, the window location and scale is outputted. If rejected, the process resets and the next window begins evaluation. The process is fully pipelined to extract a feature value each cycle.

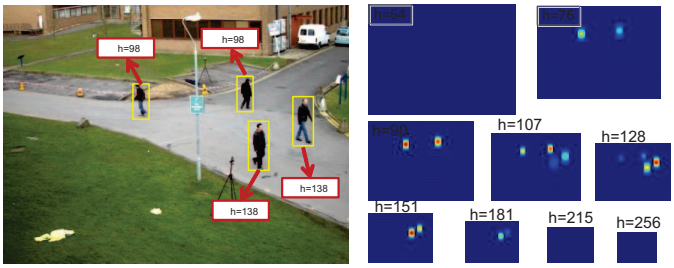

Fig. 2. The temporal prior maps over different detection scales.

IV. EXPERIMENTS

The proposed algorithm is evaluated on the Caltech pedestrian dataset [4]. The typical evaluation on Caltech pedestrian dataset is image based. To show the improvement of temporal prior, the new algorithm is tested on continuous video frames. The prior maps over different scales are illustrated in Fig. 2. It is reasonable to assign high priors around the locations where objects are present in the previous frame. The detection performance comparison with/without temporal prior is shown in TABLE I. It can be found that the temporal priors improve temporal stability of detections, enabling better and more stable detection results. The speed comparison also shows that the introduction of temporal prior has no additional computation costs. Although the detector of [1] is used in this paper, the proposed algorithm has no limitation to be applied to any other detectors. It is a useful general tool.

We also implemented our hardware design in Verilog using Xilinx Vivado 2014.3 targeting 200 MHz on a Zynq XC7Z045 FPGA. We used a ZC706 development board. Depending on the number of pedestrians in the scene, the detector was able to run at 30~40 FPS.

V. CONCLUSION

We proposed an efficient temporal prior to obtain stable pedestrian detections in real-time videos. We also presented a hardware based implementation on an FPGA which runs at a rate of 30~40 FPS on VGA video.

REFERENCES

[1] Z. Cai and N. Vasconcelos. "A real-time cascade pedestrian detection based on heterogeneous features," International SoC Design Conference, 2015.

[2] M. Jacobsen, Z. Cai and N. Vasconcelos. "FPGA implementation of HOG based pedestrian detector," International SoC Design Conference, 2015.

[3] J. Cho, S. Mirzaei, J. Oberg, and R. Kastner, "Fpga-based face detection system using haar classifiers," in FPGA. ACM, 2009.

[4] P. Dollar, C. Wojek, B. Schiele, and P. Perona. "Pedestrian detection: An evaluation of the state of the art". IEEE Transactions on Pattern Analysis and Machine Intelligence, 34(4):743–761, 2014.