26th International Conference on Field Programmable Logic and Applications, FPL 2016, Lausanne, Switzerland, August 29 - September 2, 2016

Optimizing Hardware Design for Human Action Recognition

Xiaoyin Ma, Jose Rodriguez Borbon, Walid Najjar, Amit K. Roy-Chowdhury
University of California, Riverside
Riverside, CA 92521, USA
xma@ece.ucr.edu, jrodr050@ucr.edu, najjar@cs.ucr.edu, amitrc@ece.ucr.edu

***Abstract*—Human action recognition (HAR) is an important topic in computer vision having a wide range of applications: health care, assisted living, surveillance, security, gaming, etc. Despite significant amount of work having been conducted in this area in recent years, the execution speed still limits real-time applications. Moreover, it is highly desirable to have the compute-intensive feature extraction stage done right at the output of the camera to extract and transfer only action feature in multi-camera network setting and hence reduce network bandwidth requirement. In this work, we first evaluate the possibility to perform feature extraction under reduced precision fixed-point arithmetic to ease hardware resource requirements. We compared the Histogram of Oriented Gradient in 3D (HOG3D) feature extraction with state-of-the-art Convolutional Neural Networks (CNNs) methods and shown the later to be 75X slower than the former. Our experiment shows that by re-training the classifier with reduced data precision, the classification performs as well as the original double-precision floating-point. Based on this result, we implement an FPGA-based HAR feature extraction for near camera processing using fixed-point data representation and arithmetic. This implementation, using a single Xilinx Virtex 6 FPGA, achieves about 70x speedup over multicore CPU. Furthermore, a GPU implementation of HAR is introduced with 80x speedup over CPU (on an Nvidia Tesla K20). Last but not least, a power comparison is presented for the three platforms.**

I. Introduction

The rapid growth of camera and storage capabilities, over the past decade, and the related drop in their prices, has resulted in an exponential growth in the size of video repositories, such as YouTube. In 2015, 400 hours of videos were uploaded to YouTube every minute [1]. At the same time, massive amount of images/videos are generated from monitoring cameras for care to the elderly, assistance to the sick, satellite-based monitoring for earth science research, telescopes for space exploration, and security. Human annotation and manual manipulation of such videos is infeasible. Computer vision technology plays an essential role in automating the indexing, sorting, tagging, searching and analyzing huge amount of video data. Activity recognition in general, and Human Action Recognition (HAR) in particular, are some of the most challenging topics in computer vision today. Despite significant progress in accuracy and speed over the past few years, the execution speed, less than one frame/sec in most of the existing methods, still limits real-time applications [2] in a wide range of applications: health care, assisted living, surveillance, security, gaming, etc.

Moreover, many, if not most, monitoring situations increasingly rely on a network of cameras. These networks are mostly wireless for cost and portability reasons. Hence, the bandwidth is at a premium. It is therefore highly desirable to have the compute-intensive feature extraction stage of HAR done near the camera and to extract and transfer only action features and hence reduce network bandwidth requirements. The hardware acceleration of HAR for those applications is particularly important due to the stringent power and speed requirement. We choose HOG3D [3] as our HAR feature extraction method for its high accuracy and relatively low computational complexity. Compared to the state-of-the-art Convolutional Neural Networks (CNNs) methods, HOG3D is about 75x faster (see Section VI). Therefore, the HOG3D method is more applicable for the real-time applications discussed above.

In the last few years we have witnessed the end of Denard Scaling that had held since 1974: It is no longer possible to increase the clock speed of digital devices simply by shrinking the feature size. While Moores Law still holds, meaning that more cores can be built on the same die, a multi- or many-core execution suffers from high memory off-loading overhead for streaming data. Streaming video data needs to be loaded into memory before it can be processed. This implies that the processing speed will be limited by the memory bandwidth. FPGAs do not rely on memory off-loading; rather the video data can be streamed directly onto the chip where it is processed. Furthermore, in recent years we have witnessed a tremendous increase in the size, speed and bandwidth capabilities of modern FPGA devices making them excellent candidates to implement, in hardware, large complex, massively parallel and bandwidth intensive applications such as HAR.

In this work, we explore the use of reduced, but variable, bit-width for data representations in feature extraction, machine learning and classification (recognition) in our FPGA implementation for HAR acceleration. To the best of our knowledge, this is the first FPGA implementation that is targeted on full human body action recognition with state-of-the-art recognition rate. The contributions of this paper are:

1) A comprehensive evaluation of the HAR recognition accuracy under reduced data precision. Our experimental results show that retraining the classifier using reduced data width can compensate for the precision loss in feature extraction and achieve the same recognition rate

as using floating-point data. This result significantly relieves hardware resource requirement for video classification and enables faster and more energy efficient vision processing. Our final implementation starts with 8-bit pixels but preserves the precision of the data using variable bit-width in the intermediate and final results (Section IV).

2) A complete FPGA implementation of feature extraction for real-time human action recognition using near-camera processing. This implementation reads raw video pixels as input and produces the final bag-of-words features. The output bag-of-words is only 1,000 16-bit integers. Thus it is especially suitable for embedded platforms that process videos/images close to cameras to reduce network bandwidth requirement.
3) In addition to the FPGA implementation, we have implemented a single-precision floating-point HAR application in GPU. To the best of our knowledge, it is the first GPU implementation of HAR algorithm using handcrafted features. Our application is able to process 16 videos in parallel achieving a throughput of 1,616 frames per second.
4) A throughput comparison of multicore CPUs, GPU and FPGA platforms shows our FPGA and GPU implementations achieve 80x and 70x speed-up over the multicore CPU. What's more, a power consumption comparison shows that FPGA uses 3x less power than GPU.

The remainder of this paper is organized as follows: Section II covers related work in state-of-the-art HAR algorithms as well as previous implementations on FPGAs. Section III introduces the Histogram of Oriented Gradients in 3D (HOG3D) and bag-of-words features for human action recognition. Fixed-point evaluation of human action recognition is discussed in Section IV. In sections V we show the details of our FPGA implementation. The result and comparison is presented in Section VI. The conclusion is in Section VII.

II. Background

Most of the HAR algorithms developed over the past few years [4] can be broadly described in terms of the following framework. A temporal sliding window is applied on a video stream to find actions. For frames within a window, spatial-temporal features are extracted from 3D regions by either interest point detector or dense-sampling. The extracted features are clustered (e.g. K-means) to build visual vocabularies in training. These features are then binned into histograms based on their center to form high-level fixed-size bag-of-words (BOW) feature vector [5], [6], [7], [8], [9]. A classifier is trained using BOW features to detect the targeted action. In many recent methods, the relationships between the actions are also exploited in a structural support vector machine (SVM) [10] or graph-modeling framework [11]. Note that in our work, we only consider the case that each video clip contains one action and there are known numbers (and known labels) of actions to be recognized.

Despite many different detectors being developed for interest point detection [12], [13], [14], [15], [16], [17], it has been shown that sampling the window at regular positions in space and time (dense-sampling) achieves the best performance in most benchmarks [2]. Also many algorithms have been proposed for the abstraction of pixel information to capture human movements, including patches of normalized derivatives in space and time [18], image gradients [13], optical flow [19], [13], Speeded-up Robust Features (SURF) extended to 3D [17], combination of histograms of oriented gradients and histograms of oriented optical flow [20], and 3D extended HOG (HOG3D) [3], [21]. Among the various spatial-temporal features, HOG3D with dense-sampling has been shown to achieve good performance for HAR [2] while being less stringent in hardware requirement. Thus, we choose this method in our real-time embedded action recognition system.

Previous FPGA implementations have mainly focused on the hand gesture recognition, a predecessor of HAR [22], [23]. The design of a vision processing chip that can be used for gesture recognition is described in [24]. In [25] a 600-fps real time action recognition system was proposed for four types of hand gestures. Meng and Freeman implemented a re-configurable system for action recognition and have tested the algorithm using a full human body action benchmark [26]. However, the 63% mean-average recognition rate on the KTH dataset [18] is well below state-of-the-art results. To the best of our knowledge, none of the previous work have performed action recognition under reduced data precision. In this paper, we propose an FPGA implementation that is comparable to state-of-the-art recognition rates in complex HAR benchmarks while maintaining efficient FPGA resource usage by applying 8-bit fixed-point arithmetic.

The study of fixed-point implementation in image and video processing were primarily focused on the data range analysis and precision/errors associated with reduced bit-width [27], [28], [29]. It was also shown that machine learning model has certain tolerance on the reduced data precision and has lead to various FPGA implementation using fixed-point data [30], [31], [32]. In our work, we take one step further by building a learning predictive model using the reduced precision features and show that the new model can compensate for the precision loss in feature extraction having a comparable recognition rate with double-precision floating-point.

III. Histograms of Oriented Gradients in 3D

In this section, we describe the HOG3D algorithm and BOW features used for the action recognition.

HOG3D Features HOG3D Features [3] are extracted from within a 3D box centered at a key point to encode both spatial and temporal information. In this algorithm each 3D box is divided into several non-overlapping cells from which the HOG3D features are calculated. The final feature for each box is the concatenation of all cell features in that box. Each cell is further divided into several sub-cells to compute spatial-temporal histogram. The overview of the feature extraction

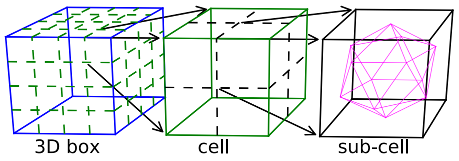


Fig. 1: Illustration of HOG3D box, cells and sub-cells.

process is shown in Figure 1. For each sub-cell, the histogram is obtained by projecting the three mean-gradients (dx, dy and dt) to the icosahedron surfaces, as shown in Equation 1 and 2, where

$$\vec{g} = \begin{pmatrix} \bar{d}_x \\ \bar{d}_y \\ \bar{d}_t \end{pmatrix} \tag{1}$$

$$\vec{h} = P\vec{g} \tag{2}$$

P is the projection matrix (10×3 for binning to half orientation and 20×3 for full orientation). In half orientation (used in this work), only the absolute value of the projected gradient is kept. The projected gradients are then subtracted by a threshold computed in Equation 3 and all negative values are set to zero. Then, the histogram vector is normalized in Equation 4. Every group of gradient vectors (dx, dy, dt) generates 10 sub-cell histogram values.

$$threshold = \frac{1.618034}{\sqrt{\sum g_i^2}} \tag{3}$$

$$\vec{h}_{norm} = \frac{\sqrt{\sum g_i^2}}{\sum h_j}\vec{h} \tag{4}$$

Histogram vectors in a single cell are accumulated from sub-cell histogram using a vector add operation and then normalized again by L2 normalization shown in Equation 5.

$$\vec{H}_{norm} = \frac{1}{\sqrt{\sum H_i^2}}\vec{H} \tag{5}$$

Gradient Computation and Integral Video The histograms are computed by using average gradients in three directions (x, y, t) . The gradients are computed using a simple mask $[-1, 1]$ as in Equation 6:

$$\begin{cases} dx = p(x+1, y, t) - p(x, y, t) \\ dy = p(x, y+1, t) - p(x, y, t) \\ dt = p(x, y, t+1) - p(x, y, t) \end{cases} \tag{6}$$

Note that edge pixels are replicated (gradients are set to 0 at edges). Integral video is used to rapidly compute average gradients within sub-cells. The integral video is an extension of the popular integral image method proposed by Viola and Jones [33]. Integral video has been shown to be an efficient method to extract spatio-temporal features in previous works [34], [17]. In integral video, the integration value for a gradient at location (x', y', t') is the sum of all gradient values at current and previous locations as shown in Equation 7.

$$I_d(x', y', t') = \sum_{t=0}^{t'}\sum_{y=0}^{y'}\sum_{x=0}^{x'} d(x, y, t) \tag{7}$$

In HOG3D, the integral videos for each of the three gradients are computed using Equation 7. With a given 3D sub-cell at location (x, y, t, w, h, l), the average gradient is computed using Equation 8. Note that x, y and t are the smallest column, row and time index in a sub-cell respectively and w, h and l are the width, height and length of the sub-cell.

$$\begin{aligned} \bar{d} = [I_d(x, y, t+l) &+ I_d(x+w, y+h, t+l)- \\ I_d(x+w, y, t+l) &- I_d(x, y+h, t+l)]- \\ [I_d(x, y, t) &+ I_d(x+w, y+h, t)- \\ I_d(x+w, y, t) &- I_d(x, y+h, t)] \end{aligned} \tag{8}$$

Dense Sampling and Multi-Scale Processing Dense sampling algorithm extracts key points at regular locations by moving a 3D box across the video at a constant stride. In our experiment, the stride is 50% of the box size that is any two adjacent boxes have 50% overlap. To further increase the feature diversity (and increase recognition accuracy), features are extracted at multiple scales. Instead of re-sizing the original frames/images, the 3D box is enlarged by approximately $\sqrt{2}$ times (each side) until the box is larger than the frame size. Thus, a single integral video computation can be used for all spatio-temporal scales. Since dealing with multiple scaled images using shared hardware is difficult [35], [32], this design also simplifies the hardware design that will be discussed later. In our experiments, the box is only re-sized in spatial dimension at seven different scales with box size of $24, 32, 48, 64, 96, 136, 192$ pixels. The box size is fixed at 16 frames in temporal dimension as multiple temporal scales has shown little impact to the final classification result in [2].

Bag-of-Words Features (BOW) BOW feature is a higher level video representation built upon pixel-level features (such as HOG3D). This method was inspired by document classification, where a histogram of "words" (also called vocabularies) are generated to model the document [36]. In this method, the model is built using their occurrence in the document regardless of the order. For computer vision applications, a visual vocabulary is computed by a clustering algorithm (e.g. K-means or KD-Tree) using the extracted features. Each HOG3D feature vector in a video clip is binned into its closest vocabulary to form a BOW feature. The BOW method has been widely used in many of the latest HAR algorithms [37], [20], [17], [3], [38], [39], [40]. In our evaluation, we choose k-means as our clustering algorithm and use 1,000 vocabularies (i.e 1,000 cluster centers in K-means) for all recognition tasks.

Training and Classification To recognize multiple actions in a video clip, the BOW features are passed to a multi-class SVM classifier for classification. The SVM classifier creates a large margin around the decision boundary (hyperplane) to achieve maximum classification performance [41]. Specifically in a linear SVM classifier, the final confidence score is the dot-product of the trained classification vector (normal vector to the hyperplane) $\vec{W}$ and the BOW feature vector $\vec{V}$ plus a constant intercept term s_0, as shown in Equation 9.

$$s = \vec{W} \cdot \vec{V} + s_0 \tag{9}$$

The decision boundary can be non-linear if the "kernel trick" is applied to the SVM leading to improved recognition rate [42], [43] but also increased computational complexity. In our evaluation, we have used both linear and χ^2 kernels to test the recognition rate as in Equation 9 and 10 respectively.

$$s = \sum \frac{2w_i \cdot v_i}{w_i + vi} + s_0 \tag{10}$$

Originally, SVM classifiers were designed for binary classification. To extend it for multi-class cases, we have adopted the generally accepted "one versus one" method. In this method, one classifier is built for each pair of actions, and the final classification decision is the highest count action after evaluating all classifiers. For example, if there are six actions to recognize, 15 classifiers (C_2^6) are modeled. The outcome of such classifier is a histogram of action counts, and the highest count (maximum count is *five* if all classifier predictions are correct) will be the decision.

Due to limited number of data in all benchmarks for training and testing, we have used cross-validation to evaluate the performance of our detector. Cross-validation is a technique commonly used in machine learning to estimate the accuracy of the predictive model. In cross-validation, the entire dataset is divided into two groups, training data and test data. A predictive model is built using the entire training data and then applied against the test data. The average recognition rate is computed by comparing the predicted labels with the ground truth. Leave-one-out cross-validation is a special case of cross-validation that uses one data as test group and the rest as training. The process is repeated until all data are used as test set. In our evaluation, we have used both regular corss-validation and leave-one-out cross-validation per dataset specification.

IV. Fixed-Point HAR

In this section, we review the HAR benchmarks used for evaluation of the fixed-point recognition. The effect of reduced bit-width on the HAR applications in k-mean clustering, SVM training and classification are also studied.

HAR Evaluation Benchmarks We have used three different benchmarks with increasing difficulty to evaluate our HAR implementation: KTH [18], UCF11 [44], [45] and UCF50 [39]. KTH benchmark is a collection of 599 videos with six different human actions. The recognition on this dataset is relatively trivial. The UCF11 consists of 1,600 video clips with 11 different actions. Every action is divided into several different groups based on similarity of videos in terms of actors, background, and/or view point. This dataset is very challenging as there is a large number of variations in camera motion, background, subjects, and object scales. UCF50 is an extension of the UCF11 by adding 39 more actions with a total of 6,680 video clips. This dataset is the most difficult among all three benchmarks due to the large number of actions to be recognized.

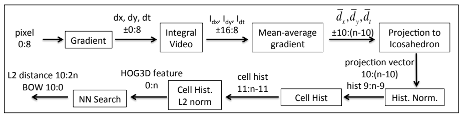
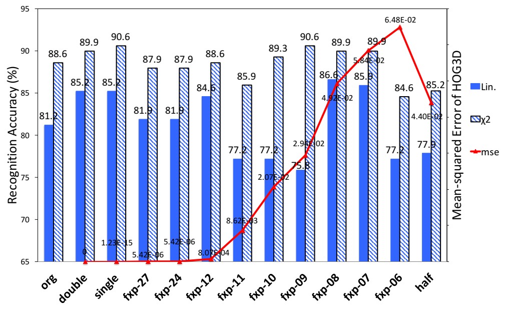


Fig. 2: HOG3D data-flow diagram and key parameter data sizes (integer:fractional) used in our implementation.

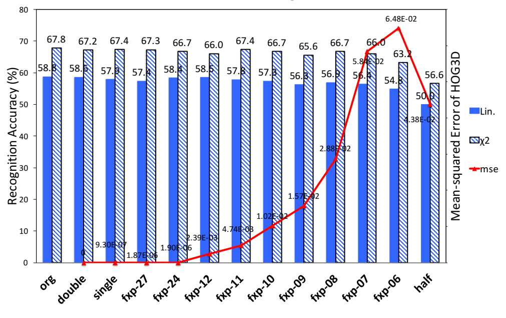
Fixed-Point Experiments We have implemented both floating-point and fixed-point HOG3D feature extraction in C++. The floating-point code is based on the original code [3], [21] with our own version of dense-sampling implementation. In dense sampling, a 3D box is moved at a regular positions within the video with 50% overlapping stride. Moreover, the spatial size is increased approximately $\sqrt{2}$ times until the box is larger than the original frame size. The two spatial directions always have the same size and we have fixed the box' temporal length to be 16 frames. In our dense-sampling algorithm, the smallest spatial size is 24 pixels. As shown in Figure 1, we choose four cells per box per direction and two sub-cells per cell per direction, thus the box size should always be a multiple of *eight* pixels. Accordingly, the dimension of the HOG3D feature is 640 with half-orientation ($4 \times 4 \times 4 \times 10$). We have also fixed the number of frames in each video clip as 396 frames for KTH, 96 frames for UCF11 and 80 frames for UCF50.

We have implemented the floating-point code in double-, single- and half-precision floating point to test the outcome at different data precision. The fixed-point feature extraction is similar to the floating-point version but with all computations performed in fixed-point. In our fixed-point evaluation program, the bit-width can be passed as an input argument at run-time for processing in different precision. The data size was obtained by sampling the three benchmarks at every step of the computation. The data flow of the activity recognition procedure and the bit-width at different steps are shown in Figure 2, where n is the bit-width (excluding sign bit, if applicable). Negative fractional values for small n in Figure 2 is automatically set to 0 in our implementation. In our experiment, we have used a large range of bit-width from 27-bit down to 6-bit. The upper bound is chosen to make sure no intermediate fixed-point data exceeds 64-bit during any stage of the computation. Due to vast amount of features extracted from each dataset, we have randomly selected features for K-means clustering. For KTH and UCF11, 400 and 200 features are selected from a video clip respectively. For UCF50, we have randomly sampled 10,000 features per action category for clustering. Fixed-point and half-precision HOG3D features are converted to single-precision floating point for k-means clustering. All clustering are set to terminate after reaching one million iterations. For fixed-point data, the centers are converted back to their respective bit-width.

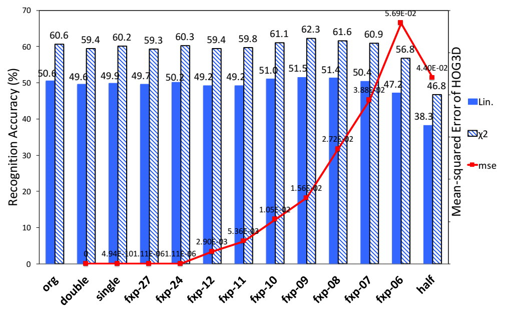
BOW features are built using brute-force nearest neighbor



(a) KTH dataset recognition results

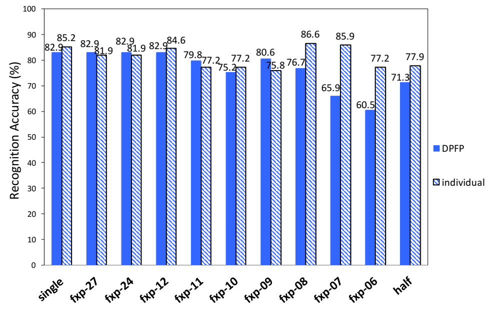


(b) UCF11 dataset recognition results

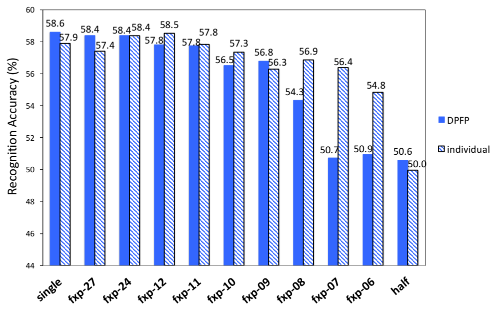


(c) UCF50 dataset recognition results

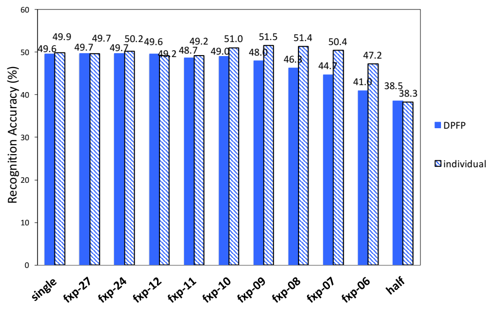
Fig. 3: Recognition accuracy, linear and χ^2, and MSE of the feature extraction for the three benchmark versus bit-with.



(a) KTH dataset recognition results using different centers.



(b) UCF11 dataset recognition results using different centers.



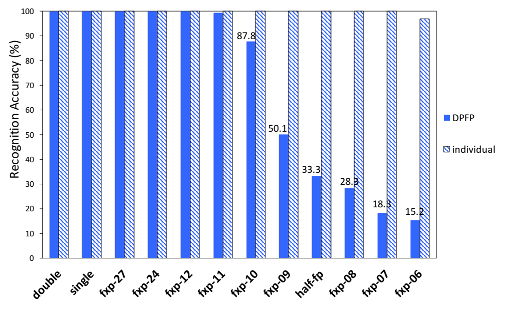
(c) UCF50 dataset recognition results using different centers.

Fig. 4: Recognition accuracy, BDFP centers and individually trained centers versus bit-width.

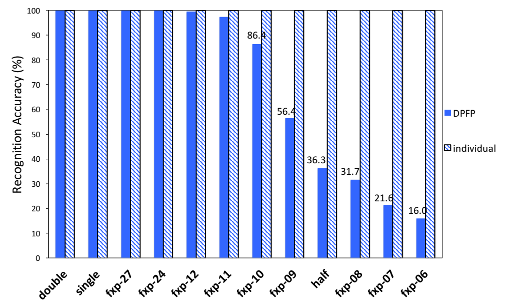
search method. The nearest cluster center for a feature vector is determined by comparing the L2 distance with all the 1,000 centers. We have implemented the nearest neighbor search in double- and single-, precision floating-point as well as fixed-point. Half-precision features are converted to single-precision for nearest neighbor search.

To evaluate the accuracy of the activity recognition, we have followed the original experimental settings from the authors of the benchmarks. We use a modified version of LIBSVM (added χ^2 kernel) for classifier training [46]. This library uses double-precision floating-point for all training. For KTH dataset, we use nine subjects (2,3,5,6,7,8,9,10, and 22) as test group and the rest as training group. For both UCF11 and UCF50 datasets, leave-one-group-out cross-validation is used (since videos at the same group are similar, they are used in training/testing together). Note that because the UCF50 is a superset of UCF11, we only evaluate the 39 actions that are not included in UCF11 dataset.

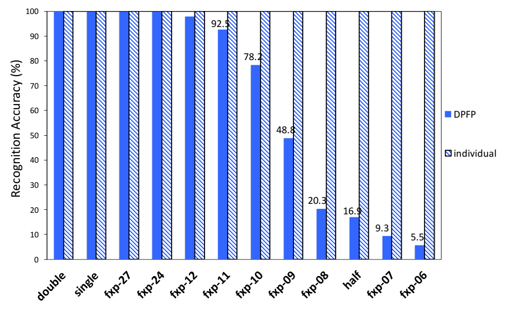
Recognition Results We evaluate the recognition accuracy by using all processing steps in fixed-point (fxp-) and compared the results with float-point data, as shown in Figure 3. We report both linear and χ^2 kernel SVM classification result. The "org" evaluation uses the original authors' code and configurations in feature extraction that uses floating-point indexed and sized 3D boxes to sample the video. We also calculate the mean-squared error (MSE) of the HOG3D features by using the double-precision floating-point (DPFP) feature as ground truth. As shown in Figure 3-(a), the accuracy at different bit-width fluctuates a lot. However, for UCF11 and UCF50, the lower bit-width is very stable as shown in Figures 3 (b) and (c). The fluctuation in KTH dataset is likely due to the limited number of training/testing samples. For UCF50 dataset, the fxp-8 slightly outperforms the DPFP recognition even though the MSE in feature extraction increased by several orders of magnitude. The half-precision recognition performs worst in all cases. The MSE at fxp-8 ranges from 2.7% to 4.9% and



(a) KTH dataset recognition results using different SVM models.



(b) UCF11 dataset recognition results using different SVM models.



(c) UCF50 dataset recognition results using different SVM models.

Fig. 5: Recognition accuracy, DPFP SVM model, individually trained SVM model.

increases by about 13 orders of magnitude from the MSE of single-precision float, while the recognition accuracy remains relatively the same. Additionally, the MSE for half-precision float is between fxp-8 and fxp-7. However, half-precision recognition accuracy is well below the lower bit-width fixed-point recognition for all test cases. This is due to the error propagation in the integral video stage. On one hand, fixed-point feature extraction does not suffer precision loss during integral video as the data range is carefully selected to avoid overflow. On the other hand, the integral video using half-precision may suffer large amounts of information loss due to the limited range and precision of half-float. This information loss is further amplified at later stages.

To study how the recognition performs well under low bit-width, we performed further experiments to investigate the effect of Kmeans-clustering. In addition to performing recognition by finding clustering centers in each bit-width ("individual" in Figure 4), we also performed nearest neighbor search by using clustering centers trained from DPFP features ("DPFP" in Figure 4). All other evaluation parameters remains the same as in previous evaluations. Only χ^2 kernel SVM result in cross-validation is used for comparison. As shown in Figure 4, recognition accuracy drops significantly as the bit-width decreases when using clustering centers from DPFP features. Therefore, rebuilding clustering centers for individual bit-width has an important contribution to overall recognition accuracy at low bit-width.

What's more, the effect of SVM training is evaluated. Similar to the K-means study, we build an SVM prediction module (with χ^2 kernel) using DPFP features in HOG3D and K-means. Then the same model ("DPFP" in Figure 5) is applied to the fixed-point features extracted from HOG3D plus BOW features using DPFP clusters. Additionally, individual SVM model is trained using the fixed-point plus BOW features for comparison. No cross-validation is used for SVM classifier comparison and the features are applied to both training and testing. Figure 5 show the comparison result. The dramatic accuracy difference between individually trained SVM models and DPFP SVM model shows that when retraining SVM using reduced precision features, the internal prediction model has been changed. This model change is significant enough, under low bit-width (fxp-10 and below), to cause the recognition failure even with the same training data.

Based on above analysis, the good recognition accuracy under low bit-width is attributed to three main factors: (1) Small information loss at early stage of feature extraction. (2) Performing K-means clustering for the reduced bit-width to build centers better suited for that bit-width. (3) Re-train the SVM classifier at the end to generate classification models specific for the data. This findings are not limited to FPGA-based applications but are relevant all hardware resource constrained embedded or real-time processing learning systems to achieve faster and more power-efficient computation.

V. FPGA Implementation

In this section we provide detailed FPGA implementation. We have implemented the feature extraction on Convey HC-2ex machine [47]. The system is composed of two Intel Xeon E5-2643 CPUs and four Xilinx Virtex-6 LX760 FPGAs. Each FPGA has 16 64-bit memory channels at 150 MHz controlled by eight memory controllers. We use 8-bit fixed-point to perform HOG3D feature extraction and nearest neighbor search. The implementation is a complete end-to-end solution that reads raw pixels in gray-scale and generates BOW features as output. Each video has 97 frames with 320×240-pixels per frame. The BOW feature is a histogram consisting 1,000 bins. For our single FPGA implementation, two channels are used for input and eight for output. The entire implementation is fully pipelined so that all modules can immediately start processing the next group of data after the first is streamed in.

Integral Video and Gradient Vector As the first step of HOG3D feature extraction, gradients are calculated based on Equation 6. One memory channel is used for pixel input. Because temporal gradients dt needs two frames to compute the final gradient, to avoid using on-chip memory to store the entire previous frame, two frames are accessed at the same time. The input controller generates addresses for "current" frame and "next" frame alternatively. "Next" frame is only used in computation of dt while "current" frame is used in all gradients. Memory resource is minimized as only one line buffer is used in computing dy. Three gradients are then sent to integral video module for further processing.

Integral video computation can potentially be expensive on FPGA as frame sizes increase (to store previous frame values for integration). Observe that Equation 8 can be re-written as $\bar{d} = J(t+l) - J(t)$, where $J(t) = I_d(x,y,t) + I_d(x+w, y+h, t) - I_d(x, y+h, t) - I_d(x+w, y, t)$. Also $J(t+l) - J(t)$ is the sum of all pixels between t and t+l (excluding t) and in the area of rectangle $(x, y, w, h\ )$. In our configuration, L is always two and t are constant indices (0, 2, 4, 6, ...). Hence, the gradient vector $\bar{d}$ can be computed using the sum of two consecutive frames INT_d ($\bar{d} = INT_d(x,y) + INT_d(x+w, y+h) - INT_d(x+w, y) - INT_d(x, y+h)$ where $INT_d(x', y') = \sum_{t=t_1}^{t_2} \sum_{y=0}^{y'} \sum_{x=0}^{x'} d(x,y,t)$).

Moreover, only a subset of pixels are used in gradient vector computation. For example, for the first scale (scale 0), pixels in every third row/column are accessed and for the third scale (scale 2) pixels in every sixth row/column are used. To maximize data sharing and reduce on-chip buffering, integral video is computed for a group of scales that uses the same pixels (e.g. multiple of 3). All groups share a single line-buffer that stores integration values at previous row. Each group uses one FIFO to store previous frame's integration values at locations they use the pixels. As a result, the video integration module does not need to buffer every pixel in the previous frame. In our configuration, only 12761 out of 76800 (320×240) pixels in a frame are saved in the previous frame FIFO.

In the FPGA design, integral image is first computed for each frame of the pixel gradients. Pixels are sent to frame FIFO after line integration if they belong to that scale group (every 3 row/col for scale 0, 2, 4, 6, every 4 row/col for scale 1, 3 and every 17 row/col for scale 5). Odd indexed frames are sent to the FIFO for integration and then to compute gradient vector.

The pixel gradient, video integration and gradient vector computation module is implemented in C++ and synthesized by Xilinx Vivado HLS. The input address generator unit is directly implemented in Verilog VHDL.

HOG3D Feature Extraction Gradient projection module is implemented in Verilog HDL. The procedure of generating sub-cell HOG3D features are shown in Figure 6. In our design, the $L2-norm$ of the gradient vectors ($\sqrt{\sum g_i^2}$ is computed in parallel with the projection. The norm coefficient is to compute $\frac{\sqrt{\sum g_i^2}}{\sum h_j}$ in Equation 4. Note that each HOG3D feature consists of 10 elements. Because the values of $L2-norm$ for gradient vectors have a very large range, values are converted to half-precision floating-point before performing division and then converted back to fixed-point. Seven gradient projection units

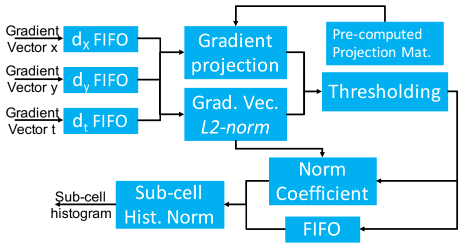


Fig. 6: Block diagram of gradient projection module on FPGA.

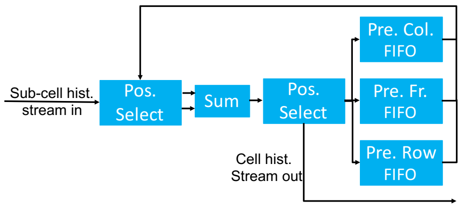


Fig. 7: Block diagram of cell histogram generation module.

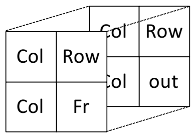


Fig. 8: Diagram of destination FIFOs for each location in a cell. Out means the cell histogram is streamed out.

are instantiated (one for a scale).

Sub-cell HOG3D features are then accumulated by vector add. Each cell consists of $2 \times 2 \times 2$ sub-cells, as shown in Figure 1. There are no overlapping between adjacent cells in any dimension. Three FIFOs are used to store histograms in previous column, previous row, and previous frame as shown in Figure 7. The sub-cell histogram is arranged in columns, rows and frames like pixels. Each column contains one sub-cell features (10 elements) and each row contains all sub-cells extracted in a row of sliding windows from integral video. The accumulation module determines the position of current sub-cell histogram inside a sliding box (see cell diagram in Figure 7). Based on the histogram location, it reads one of the three FIFOs and adds current feature to the values in the FIFO. Then, it sends the accumulated to appropriate destination FIFO as shown in Figure 8. After accumulating histograms in all eight locations, cell histograms are streamed out. This module is implemented in C++ and synthesized by Xilinx Vivado HLS.

Cell histograms are normalized according to Equation 5 and the normalization unit is implemented in Verilog HDL. The entire computation is performed in fixed-point as discussed in Section IV.

Cell histograms are sent to main memory on FPGA. Different scales of histograms are stored linearly one after another. The output controller generates sequential addresses plus a constant scale offset for memory writes. Output memory controller is implemented in Verilog HDL. Seven memory channels are used to send HOG3D features into main memory.

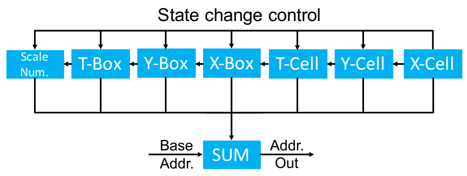


Fig. 9: Nested state machines control the address offsets generation to construct HOG3D features.

Note that the entire HOG3D feature extraction is computation-bounded. The input and output memory channels are sufficient to deliver/send data.

Nearest Neighbor Search The input controller of the nearest neighbor search is responsible to generate feature addresses that construct actual HOG3D features. The HOG3D features are obtained by concatenating all cell histograms in a 3D box as shown in Figure 1. The address generation module is analogous to [32]. Different scales are processed sequentially. Seven nested state-machines are used to generate the address offsets for current scale, t-box, y-box, x-box, t-cell, y-cell, and x-cell, as shown in Figure 9. “Scale Num.” is the outmost state machine while “X-cell” is the the innermost state machine. “X-cell” state machine generates five offsets for 40 consecutive histograms (4×10 cell histogram, 8-bit each as shown in Figure 2). When each inner state machine reached the end, it notifies all outer state machines so that they can check if a state change is needed. After all state machines reached the end, addresses for all scales are generated. It takes 80 clock cycles to generate all addresses for one HOG3D feature (640 elements), and there are 10241 features per 97-frame video.

The nearest neighbor search module finds the smallest distance between each feature and all 1000 centers. The HOG3D feature has 640 dimensions (10 features per sub-cell). To find the nearest neighbor, the distance between each feature and 1000 centers are computed and then compared. To maximize the throughput, 640 multipliers are instantiated to compute one distance per clock cycle. HOG3D features are loaded from memory and all center data is stored on chip in ROMs. These 640 multiplications are divided into 80 multiplication cores as each memory access will return 8 features as shown in Figure 10. The centers are also divided into 80 ROMs (1000×64-bit size) based on their position in a feature. Input feature is passed to a three-stage data splitter that sends the feature into appropriate multiplier core to better meet timing. Each multiplication core computes the sum of eight euclidean distances. Another three-stage summation is used to produce the final $L2$ distance between the feature with one of the centers.

The BOW feature is a histogram of 1000 bins. When a minimum distance center of a feature is found, the bin count corresponding to the center will be incremented by one. We build a streaming histogram accumulation unit to process 1000 histogram bins. The diagram of this unit is shown in Figure 11. Histogram bins are organized in a chain structure. The index (bin number) is streamed in from the first bin. Each bin

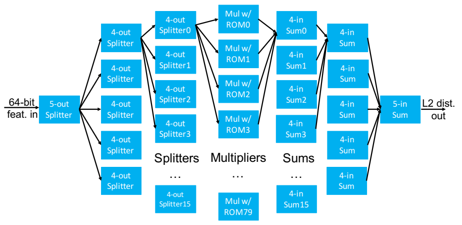


Fig. 10: Illustration of parallel nearest neighbor Search architecture.

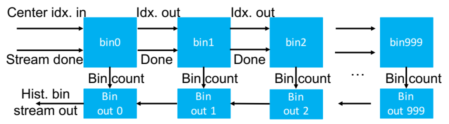


Fig. 11: Illustration of streaming histogram accumulation unit.

accumulator will check if the index is the same as its defined bin value. If the input equals current bin number, its counter is incremented. If it does not equal current bin, the index will be sent to the output which is the input of next bin. Immediately after all 10241 indices are streamed in, a done signal is streamed from bin0 to bin999 causing accumulated bin values to be streamed out. Once received the done signal, each bin is ready to start accumulating next histogram. Similar to HOG3D feature extraction, nearest neighbor search is computation-bounded.

VI. Results and Discussion

In this section we evaluate the the speedup of our feature extraction FPGA and GPU implementation over a highly optimized CPU code. All tests are based on UCF50 dataset with 97 frames per video.

CPU Results Our CPU implementation uses single-precision floating-point for HOG3D feature extraction and nearest neighbor search. Our CPU platform is an Ubuntu machine with two Intel Xeon-E5520 quad-core CPUs and 24-GB RAM. The CPU program is implemented in C++ with Intel TBB library for multi-threading capability. SSE is also enabled in nearest neighbor search. The processing time measured for CPU implementation is the actual feature extraction time. Loading the video from hard drive and writing features to file are not included. Our experiment shows that it takes about 4.80 seconds for a multi-core machine to process one video (97 frames).

GPU Results In this section[1] we briefly describe our single-precision floating point GPU implementation of the HAR algorithm using HOG3D feature extraction running on an Nvidia Tesla K20c GPU[48]. The code is compiled with the NVIDIA CUDA toolkit 6.0 with the Basic Linear Algebra Subroutines (cuBLAS) V2.0. The CPU is used for input-output and control only. The time measurements reported do not

[1]The entire details of our GPU implementation are beyond the scope of this paper due to space constraints. It should be noted however that it is the first such implementation of HOG3D on GPUs.

TABLE I: FPGA implementation resource utilization.

Attributes	Virtex-6 LX760 (Convey)	Kintex-7 XCU060
Registers	312085 (32%)	214820 (32%)
LUTRam	39987 (30%)	15068 (10%)
LUTs	197025 (41%)	123708 (37%)
36KBram	247 (34%)	265 (25%)
DSPs	168 (19%)	320 (12%)

TABLE II: Human action recognition feature extraction throughput comparison.

Platform	Throughput (fps)	Speedup
CPU	20.2	1
GPU	1,616.0	80
one FPGA-fxp8	1420.8	70
four FPGA-fxp8	5682.8	280

include the time to transfer the videos to the GPU nor the time to allocate GPU global memory. Our GPU implementation can process 1,616 frames per second (fps) i.e. 16 videos in 0.96 seconds.

FPGA Results We have synthesized the entire implementation (including the Convey memory interface) using Xilinx ISE 14.7. All Vivado HLS generated modules are connected to our Verilog code using FIFOs. For arithmetic operations, we choose to place portions of the nearest-neighbor search distance computation (multiplication) into DSPs (26% of the multiplications in nearest-neighbor search) and all other operations in pure logic. Table I summarizes the synthesis result. In addition to the Virtex-6 FPGA on the Convey HC-2ex machine, we also implemented our algorithm on a Xilinx Kintex-7 Ultrascale FPGA (XCU060) for comparison. The clock on the Kintex-7 FPGA is also set to 150 MHz.

Both HOG3D feature extraction and nearest neighbor search are computation-bounded. Then, the processing speed is determined by the actual number of clock cycles to process data on FPGA. HOG3D feature extraction takes $96 \times 320 \times 240$ clock cycles to process (frame 97 is only used for dt computation, and is processed in parallel with frame 96) which is equivalent to 0.049 seconds at 150MHz frequency. Additionally it takes about 0.068 seconds to process a video in nearest neighbor search (1000×10241 clock cycles). Consequently, the overall speed of our FPGA implementation is about 1420.8 frames per second. The summary of all platforms in shown in Table II.

Note that an FPGA implementation of the this application using floating-point data would not only be too large for both FPGAs, its parallelism would also be constrained by the memory bandwidth.

Power Consumption Comparison We have compared the power efficiency between CPU, GPU and FPGA. For CPU, we used the thermal design power (TDP) of the processor (two processors) and for GPU, we have used the NVIDIA System Management Interface (nvidia-sim) to determine power consumption. For FPGA power consumption, we estimated the power using Xilinx Power Estimator with 50% toggle rate. The power consumption result is shown in the Table III.

TABLE III: HOG3D implementation power consumption comparison.

Platform	Power (W)	Joules/frame
CPU	160	7.921
GPU	85	0.053
FPGA	28.1	0.020

Speed, Power, and Accuracy Trade-off Our work focuses on the action recognition in camera networks where battery powered cameras are distributed across multiple locations. Information captured by cameras are sent via Wi-Fi. A centralized processing of such information is limited by the available bandwidth, security concerns and the difficulty in processing massively large amounts of data. Thus we propose to use FPGAs to process raw pixels behind the camera and only send extracted action features back to center servers for classification. In this application, both processing speed and power consumption are critical. Therefore, FPGA is more suitable for its relatively fast processing speed and lowest power consumption compared to both GPU and CPU implementations.

Recently, CNNs have been shown to provide exceptional accuracy on large-scale recognition problems [49], [50], [51], [52]. However, applying a deep neural network for real-time embedded applications remains challenging [53], [54].

The computational complexity of CNNs is significantly higher than that of hand-crafted feature extraction (such as HOG3D). We have compared the throughput of the open-source CNN-based HAR algorithm [55] with our implementation of HOG3D on GPU: HOG3D feature extraction is 75X faster on the same benchmarks. Accordingly, using a hand crafted feature (e.g. HOG3D) for real-time and embedded system implementation is at present the better option.

VII. Conclusion

In this paper, we explore the feature extraction, classifier training and prediction under reduced fixed-point bit-width for human action recognition. We compare the accuracy of HAR using three popular action recognition benchmarks under different precision. The result shows that by re-training the classifier with low precision data (8-bit fixed-point), the recognition accuracy is the same as double-precision floating-point. This result significantly lessens the hardware requirement for embedded action recognition systems. By applying this result to our FPGA implementation, we are able to achieve 70x speedup over multicore CPU code. In addition, this speed is about 0.875x of our GPU implementation (approximately 80X speedup over CPU) running on a high-end GPU. Furthermore, the power consumption comparison shows that FPGA implementation is more suitable for embedded applications as it uses 3x less power.

References

[1] http://www.reelseo.com/hours-minute-uploaded-youtube/, accessed: 2016-03-01.

[2] H. Wang, M. M. Ullah, A. Kläser, I. Laptev, and C. Schmid, “Evaluation of local spatio-temporal features for action recognition,” in *British Machine Vision Conference*, 2009, pp. 124–1.

[3] A. Kläser, M. Marszałek, and C. Schmid, “A spatio-temporal descriptor based on 3D-gradients,” in *British Machine Vision Conference*, 2008, pp. 275–1.

[4] R. Poppe, “A survey on vision-based human action recognition,” *Image and Vision Computing*, vol. 28, no. 6, pp. 976–990, 2010.

[5] J. Sivic and A. Zisserman, “Video google: a text retrieval approach to object matching in videos,” in *Proceedings of the Ninth International Conference on Computer Vision*. IEEE, 2003, pp. 1470–1477.

[6] O. G. Cula and K. J. Dana, “Compact representation of bidirectional texture functions,” in *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, vol. 1, 2001, pp. 1041–1047.

[7] C. Wallraven, B. Caputo, and A. Graf, “Recognition with local features: the kernel recipe,” in *Proceedings of the Ninth IEEE International Conference on Computer Vision*. IEEE, 2003, pp. 257–264.

[8] J. Willamowski, D. Arregui, G. Csurka, C. R. Dance, and F. Lixin, “Categorizing nine visual classes using local appearance descriptors,” *Illumination*, vol. 17, p. 21, 2004.

[9] S. Lazebnik, C. Schmid, and J. Ponce, “Beyond bags of features: Spatial pyramid matching for recognizing natural scene categories,” in *IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, vol. 2. IEEE, 2006, pp. 2169–2178.

[10] X. Wu, D. Xu, L. Duan, J. Luo, and Y. Jia, “Action recognition using multilevel features and latent structural SVM,” *IEEE transactions on Circuits and Systems for Video Technology*, vol. 23, no. 8, pp. 1422–1431, 2013.

[11] Y. Zhu, N. Nanyak, and A. K. Roy-Chowdhury, “Context-Aware activity modeling using hierarchical conditional random fields,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 37, no. 7, pp. 1360–1372, 2015.

[12] I. Laptev, “On Space-Time interest points,” *International Journal of Computer Vision*, vol. 64, no. 2-3, pp. 107–123, 2005.

[13] P. Dollar, V. Rabaud, G. Cottrell, and S. Belongie, “Behavior recognition via sparse spatio-temporal features,” in *International Workshop on Visual Surveillance and Performance Evaluation of Tracking and Surveillance*. IEEE, 2005, pp. 65–72.

[14] A. Oikonomopoulos, I. Patras, and M. Pantic, “Spatiotemporal salient points for visual recognition of human actions,” *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, vol. 36, no. 3, pp. 710–719, 2005.

[15] H. Jhuang, T. Serre, L. Wolf, and T. Poggio, “A biologically inspired system for action recognition,” in *IEEE 11th International Conference on Computer Vision*. IEEE, 2007, pp. 1–8.

[16] C. Thurau and V. Hlavac, “Pose primitive based human action recognition in videos or still images,” in *IEEE Conference on Computer Vision and Pattern Recognition*. IEEE, 2008, pp. 1–8.

[17] G. Willems, T. Tuytelaars, and L. Van Gool, “An efficient dense and Scale-Invariant Spatio-Temporal interest point detector,” in *European Conference on Computer Vision*, 2008, pp. 650–663.

[18] C. Schuldt, I. Laptev, and B. Caputo, “Recognizing human actions: a local SVM approach,” in *Proceedings of the 17th International Conference on Pattern Recognition*, vol. 3. IEEE, 2004, pp. 32–36.

[19] B. D. Lucas and T. Kanade, “An iterative image registration technique with an application to stereo vision,” in *International Joint Conference on Artificial Intelligence*, vol. 81, no. 1, 1981, pp. 674–679.

[20] I. Laptev, M. Marszalek, C. Schmid, and B. Rozenfeld, “Learning realistic human actions from movies,” in *IEEE Conference on Computer Vision and Pattern Recognition*. IEEE, 2008, pp. 1–8.

[21] A. Kläser, “Learning human actions in video,” Ph.D. dissertation, Université de Grenoble, 2010.

[22] Y. Shi and T. Tsui, “An FPGA-Based smart camera for gesture recognition in HCI applications,” in *Asian Conference on Computer Vision*, 2007, pp. 718–727.

[23] C. Li and W. Chen, “A novel FPGA-based hand gesture recognition system,” *Journal of Convergence Information Technology*, vol. 7, no. 9, pp. 221–229, 2012.

[24] A. A. Maashri, M. Debole, M. Cotter, N. Chandramoorthy, Y. Xiao, V. Narayanan, and C. Chakrabarti, “Accelerating neuromorphic vision algorithms for recognition,” in *Proceedings of the 49th Annual Design Automation Conference*. ACM, 2012, pp. 579–584.

[25] Z. Hou, H. Zhu, N. Zheng, and T. Shibata, “A single-chip 600-fps real-time action recognition system employing a hardware friendly algorithm,” in *International Symposium on Circuits and Systems*. IEEE, 2014, pp. 762–765.

[26] H. Meng, M. Freeman, N. Pears, and C. Bailey, “Real-time human action recognition on an embedded, reconfigurable video processing architecture,” *Journal of Real-Time Image Processing*, vol. 3, no. 3, pp. 163–176, 2008.

[27] W. Osborne, R. Cheung, J. Coutinho, W. Luk, and O. Mencer, “Automatic Accuracy-Guaranteed Bit-Width optimization for fixed and Floating-Point systems,” in *International Conference on Field Programmable Logic and Applications*, 2007, pp. 617–620.

[28] D. Lee, A. Gaffar, R. Cheung, O. Mencer, W. Luk, and G. Constantinides, “Accuracy-Guaranteed Bit-Width optimization,” *Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 25, no. 10, pp. 1990–2000, 2006.

[29] A. Benedetti and P. Perona, “Bit-width optimization for configurable DSP’s by multi-interval analysis,” in *Conference Record of the Thirty-Fourth Asilomar on Signals, Systems and Computers*, vol. 1. IEEE, 2000, pp. 355–359.

[30] T. Wilson, M. Glatz, and M. Hodlmoser, “Pedestrian detection implemented on a fixed-point parallel architecture,” in *13th International Symposium on Consumer Electronics*. IEEE, 2009, pp. 47–51.

[31] K. Mizuno, Y. Terachi, K. Takagi, S. Izumi, H. Kawaguchi, and M. Yoshimoto, “Architectural study of HOG feature extraction processor for Real-Time object detection,” in *Workshop on Signal Processing Systems*. IEEE, 2012, pp. 197–202.

[32] X. Ma, W. A. Najjar, and A. K. Roy-Chowdhury, “Evaluation and acceleration of High-Throughput Fixed-Point object detection on FPGAs,” *Transactions on Circuits and Systems for Video Technology*, vol. 25, no. 6, pp. 1051–1062, 2015.

[33] P. Viola and M. Jones, “Rapid object detection using a boosted cascade of simple features,” in *Proceedings of the Conference on Computer Vision and Pattern Recognition*, vol. 1. IEEE, 2001, pp. 511–518.

[34] Y. Ke, R. Sukthankar, and M. Hebert, “Efficient visual event detection using volumetric features,” in *Tenth International Conference on Computer Vision*, vol. 1. IEEE, 2005, pp. 166–173.

[35] Q. ”Zhu, N. Garg, Y. Tsai, and K. Pulli, “An energy efficient time-sharing pyramid pipeline for multi-resolution computer vision,” in *IEEE 21st International Conference on Very Large Scale Integration*. IEEE, 2013, pp. 278–281.

[36] S. ”Deerwester, S. T. Dumais, G. W. Furnas, T. K. Landauer, and R. Harshman, “Indexing by latent semantic analysis,” *Journal of the American Society for Information Science*, vol. 41, no. 6, pp. 391–407, 1990.

[37] I. Laptev and T. Lindeberg, “Local descriptors for spatio-temporal recognition,” in *Spatial Coherence for Visual Motion Analysis*. Springer, 2006, pp. 91–103.

[38] H. Wang, A. Kläser, C. Schmid, and C. L. Liu, “Action recognition by dense trajectories,” in *Conference on Computer Vision and Pattern Recognition*. IEEE, 2011, pp. 3169–3176.

[39] K. K. Reddy and M. Shah, “Recognizing 50 human action categories of web videos,” *Machine Vision and Applications*, vol. 24, no. 5, pp. 971–981, 2013.

[40] H. Wang and C. Schmid, “Action recognition with improved trajectories,” in *Proceedings of the International Conference on Computer Vision*. IEEE, 2013, pp. 3551–3558.

[41] V. Vapnik, *The nature of statistical learning theory*. Springer Science & Business Media, 2000.

[42] S. Y. Kung, *Kernel Methods and Machine Learning*. Cambridge Uni. Press, 2014.

[43] B. Schölkopf, C. J. C. Burges, and A. J. Smola, *Advances in kernel methods: support vector learning*. MIT press, 1999.

[44] J. Liu, J. Luo, and M. Shah, “Recognizing realistic actions from videos “in the wild”,” in *Conference on Computer Vision and Pattern Recognition*. IEEE, 2009, pp. 1996–2003.

[45] J. Liu, Y. Yang, and M. Shah, “Learning semantic visual vocabularies using diffusion distance,” in *Conference on Computer Vision and Pattern Recognition*. IEEE, 2009, pp. 461–468.

[46] C. Chang and C. Lin, “LIBSVM: A library for support vector machines,” *Transactions on Intelligent Systems and Technology*, vol. 2, no. 3, pp. 1–27, 2011.

[47] “Convey Computers,” www.conveycomputers.com, accessed: 2016-01-17.

[48] "NVIDIA's next generation CUDA compute architecture: Kepler GK110," www.nvidia.com, accessed: 2016-01-17.

[49] A. Karpathy, G. Toderici, S. Shetty, T. Leung, R. Sukthankar, and L. Fei-Fei, "Large-Scale video classification with convolutional neural networks," in *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition*. IEEE, 2014, pp. 1725–1732.

[50] K. Simonyan and A. Zisserman, "Two-Stream convolutional networks for action recognition in videos," in *Advances in Neural Information Processing Systems*, Z. Ghahramani, M. Welling, C. Cortes, N. D. Lawrence, and K. Q. Weinberger, Eds., 2014, pp. 568–576.

[51] M. Hasan and A. K. Roy-Chowdhury, "A continuous learning framework for activity recognition using deep hybrid feature models," *IEEE Transactions on Multimedia*, vol. 17, no. 11, pp. 1909–1922, 2015.

[52] J. Yue-Hei Ng, M. Hausknecht, S. Vijayanarasimhan, O. Vinyals, R. Monga, and G. Toderici, "Beyond short snippets: Deep networks for video classification," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2015, pp. 4694–4702.

[53] Z. Du, R. Fasthuber, T. Chen, P. Ienne, L. Li, T. Luo, X. Feng, Y. Chen, and O. Temam, "ShiDianNao: Shifting vision processing closer to the sensor," *ACM SIGARCH Computer Architecture News*, vol. 43, no. 3, pp. 92–104, 2015.

[54] C. Zhang, P. Li, G. Sun, Y. Guan, B. Xiao, and J. Cong, "Optimizing FPGA-based accelerator design for deep convolutional neural networks," in *Proceedings of the ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*. ACM, 2015, pp. 161–170.

[55] J. Donahue, L. Anne Hendricks, S. Guadarrama, M. Rohrbach, S. Venugopalan, K. Saenko, and T. Darrell, "Long-term recurrent convolutional networks for visual recognition and description," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2015, pp. 2625–2634.