Combination of SVM and FERN for GPU-assisted texture recognition on mobile devices

Vsevolod Yugov[1] and Itsuo Kumazawa[2]

[1] **Tokyo Institute Of Technology, Japan, Dept. of Information Processing**
satsuoni@hotmail.com

[2] **Dr. of Engineering, Imaging Science and Engineering Laboratory**
Tokyo Institute of Technology, Japan
kumazawa.i.aa@m.titech.ac.jp

Abstract

Feature point matching and texture recognition are two of the most important problems in the image processing. Recently, several new approaches to these problems using simple local features and semi-naive Bayesian classification scheme have been developed. In our paper, we show how to enhance these techniques further by combining them with Support Vector Machines using online learning techniques. The resulting algorithm is simple, robust and can be adapted to various tasks in image processing. Furthermore, we demonstrate the advantages of our method by using it to achieve real-time texture recognition on a mobile device by utilizing parallel processing capabilities afforded by the device GPU.

Keywords: *Image processing, Mobile device, GPU, Texture recognition, FERNS.*

1. Introduction

Image processing and computer vision on the mobile devices are rapidly developing topics due to the increased processing power available. In particular, many modern devices are equipped with programmable GPU (graphics processing unit), allowing for parallel computation over the whole image instead of sequential computation over each pixel. The drawback of using GPU lies in the fact that many state-of-the art image-processing algorithms are employing sophisticated features that are too complex to be processed with the limited resources available for a single thread during parallel execution. Due to this limitation, algorithms using extremely simple features, like Ferns ([8], [14]) or Local Binary Patterns ([7]) are preferable. Such algorithms have been successfully adapted to mobile tracking problem in [11], however, it has been noted that the accuracy is degrading rapidly as the number of features decreases. In our work, we concentrate on improving the accuracy of the method named Ferns, presented in [8] and [14], which uses non-hierarchical structures consisting of a small number of random binary tests to estimate probability of an image patch belonging in a certain class. Several of such structures are later combined in a Naive Bayesian way. We first consider a simple case of binary classification problem, which allows using Ferns algorithm for object and texture detection, similar to [7], which is our focus in this paper. Later we show how our algorithm can be expanded for multiple classes, allowing its usage in keypoint classification. We first replace the Naive Bayesian combination by a weighted Bayesian approach. In order to calculate weights we use online training algorithm for Support Vector Machines described in [5], for the merits of its simplicity and low computational cost. We also evaluate and compare efficiency and resulting accuracy of using probabilities, logarithmic likelihoods or binary thresholding for likelihood calculation. Our results indicate, that application of this method can achieve significant increase in accuracy compared to original Ferns, and that using binary thresholding allows reduction of memory requirements while retaining acceptable accuracy levels. We then show how our algorithm can be used for real-time texture recognition on a mobile device (in our case, iPhone 4S) using parallel processing afforded by the OpenGL ES 2.0 programming framework. Despite the limitations of this framework, increased accuracy and ease of implementation of our method allow us to achieve high degree of accuracy in the task of texture recognition. Reduced computational costs combined with parallel processing allow us to run our algorithm on a high-resolution video stream in real time.

1.1 Outline of paper

The rest of the paper is organized as follows: in section 2 we give a brief overview of related works, including the works our paper is based on. In section 3.1, we describe the Ferns algorithm in some detail. Then, in section 3.2 we introduce the main part of our algorithm, as well as some rationale behind it. In 3.5, we show how SVM training methods can be expanded for the case of several classes. We further compare accuracy and simplicity of our

Copyright (c) 2013 Advances in Computer Science: an International Journal. All Rights Reserved.

method to the original in section 3.6. Section 4 is devoted to our implementation of the proposed method on the iPhone 4S GPU, including overcoming such problems as limited memory and lack of bitwise operators. The results of this implementation are documented in section 4.3. Section 5 summarizes the presented work and gives an outline of the possible future developments.

2. Related works

The problem of recognizing a specific image patch or a kind of image texture invariant to pose or lightning conditions is the heart of many Computer Vision algorithms. Some of the algorithms used for this purpose, such as the popular SIFT algorithm ([6]), rely on the robustness of the features to certain kinds of transformations, specifically affine transformations. Others, such as LBP ([7]) and Ferns ([8]) may incorporate various poses in the statistical models used for classification. Both kinds of algorithms can be more or less efficiently used for the problem of pattern tracking on the mobile device, as shown in [11]. This approach is good when the pattern is known beforehand, has well-defined keypoints, such as angles, and the pattern does not significantly change during tracking. Due to its reliance on the CPU, this method does not scale well with the increase of the video resolution and the number of keypoints. However, it also shows that decreasing the size of Ferns leads to a significant decrease in accuracy of the method, rendering usage of the GPU parallel processing (with corresponding decrease of available number of Ferns due to memory constraints) unfeasible. Our paper aims to increase the accuracy of the Ferns by applying Support Vector Machines (SVM, [10]) training methods to replace the semi-naive Bayesian approach with a weighted semi-naive approach. The work on SVM boosting ([13], based on [3]) shows that online SVM training as described in [5] and [9] can be used to easily increase the performance of other weak classifiers, and the works like [2] and [1] confirm that weighting is an effective method for increasing accuracy of the Bayesian models. For our paper, instead of implementing a keypoint-based algorithm as in [8] and [6], we opt to use modified Ferns for a texture recognition problem, for which the use of LBP ([7]) is more common. For that purpose we combine the training of both methods, that is, we accumulate histograms of combined binary features over a selected texture area which is transformed several times by using appropriate affine transformations. In order to reduce memory requirements of the implementation, we use method similar to the one used for Real-Time SLAM ([12]), by replacing the probabilities of a certain observed features with class numbers of class with maximum likelihood, which in our case results in binary values (texture or background). We show that this does not negatively affect accuracy of the method.

3. Algorithm description

3.1 A brief outline of Ferns algorithm

In this section, we briefly outline the algorithm for image patch recognition described in [8], [14], which serves as a basis to our work. In these works, Ozuysal et. al show that image patches corresponding to a certain keypoint can be recognized on the basis of simple binary tests, when a set of possible appearances of the keypoint is treated as a class. Since the recovery of a full joint distribution of a large number of features (typically about 400) is not feasible, they propose separating a set of features of a large size N into M subsets of size S = N⁄M, choosing M in such a way that joint posterior distribution over S features can be recovered. Each subset is then assumed to be independent from all other subsets, which allows them to combine posterior probabilities by using naive Bayesian approach:

$$P(f_1, f_2, ..., f_N|C = c_i) = \prod_{k=1}^{M} P(F_k|C = c_i) \qquad (1)$$

where $P(f_1,f_2,...,f_N|C = c_i)$ is conditional probability over features f_i and $P(F_k|C = c_i)$ are probabilities of ferns F_k estimated from training values. Since ferns employ binary features, values of ferns are encoded as an integer in binary representation, $F_m = \sum_{i=0}^{S-1} 2^i f_{m*S+i}$. Each fern can then take values from 0 to $K = 2^S - 1$ The end result is semi-naive Bayesian approach, which models some but not all dependencies between features. The training phase of Ferns estimates the class conditional probabilities for each Fern F_m and each class c_i (represented by a set of affine transformations of the image patch around corresponding keypoint). For estimation of probabilities, [8] used uniform Dirichlet prior, resulting in a formula:

$$\hat{P}(F_m = k|C = c_i) = \frac{N_{k,c_i} + 1}{N_{c_i} + K + 1} \qquad (2)$$

, where $N_{k,ci}$ is the number of test samples in class c_i for which $F_m = k$, and N_{ci} is the total number of members of that class in the training set. This prevents zero-valued probability estimates. During classification, the binary features are extracted for each keypoint on the input image, and each keypoint is classified according to maximum likelihood or discarded if the maximum likelihood is too low:

$$\hat{c}_i = arg\,max_{c_i} P(f_1, f_2, ..., f_N|C = c_i) \qquad (3)$$

Since a large number of affine transformations of an image patch are used for probability estimation, the resulting distribution is independent of pose and lightning

Copyright (c) 2013 Advances in Computer Science: an International Journal. All Rights Reserved.

conditions, allowing a simple and efficient classification at run-time.

3.2 Algorithm description

In this section, we derive our algorithm for a simple case of binary classification. This algorithm can then be used for such tasks as pose-independent texture recognition (further explored in section 4) or background extraction. In our algorithm, we also use subsets of binary features for estimating joint conditional probabilities. The estimation process is in general similar to the one described in section 3.1, though it can be adapted depending on the applications. Some examples of the adaptation are described in section 4. The main difference lies in combination of the estimated joint probabilities. While Ferns use a sum of log-likelihoods:

$$log(P(f_1, f_2, ..., f_N|C = c_i)) = \sum_{k=1}^{M} log(P(F_k|C = c_i)) \qquad (4)$$

we explore the possibility of weighting the likelihoods. Specifically, the formula for final likelihood for class I is as follows:

$$L(f_1, ..., f_N, c_i) = \sum_{k=1}^{M} w_k l(F_k, c_i) \qquad (5)$$

where $L(f_1,...,f_N,c_i)$ is estimated likelihood and $l(F_k,c_i)$ can be one of the three functions:

Function 1. The joint probabilities themselves: $l(F_k, c_i) = P(F_k|C = c_i)$,

Function 2. The logarithms of joint probabilities: $l(F_k, c_i) = log(P(F_k|C = c_i))$,

Function 3. The binary-thresholded probabilities, 1 for the class with maximum joint probability over selected features, and -1 for all others:

$$l(F_k, c_i) = \begin{cases} 1 & \text{if } c_i = arg\max_{c_i} P(F_k, c_i) \\ -1 & \text{otherwise} \end{cases}$$

All three functions have their own advantages and disadvantages. Using Function 2 results in the model closest to naive Bayesian, with the weights more or less than one roughly representing positive and negative values of Spearmans rank correlation between a given fern and all others for a certain class. The training, however, has a slightly larger performance cost, which has to be taken into account if the likelihoods are updated during classification. Use of Function 1 has the least calculation cost, but has little theoretical basis. Use of Function 3 is ess performance-intensive option that sacrifices additional information present in Functions 1 and 2, but can provide memory savings up to a factor of 8. Our experiments in 3.6 show that in case the training is done offline, Functions 1 or 3 in general provide better accuracy over Function 2, with no additional cost at the classification stage. If we consider Function 3 for the binary case, we can see that it reduces original problem to a set of semi-independent binary classifiers, which have to be linearly combined into a stronger (also binary) classifier. This is a classical definition of the binary boosting problem. Similar to [13], we use the online support vector machine training methods [5], [9] to calculate the weighting coefficients n. In particular, for this problem we use NORMA ([5]) over Pegasos ([9]) method to further simplify implementation. NORMA is a stochastic gradient descent-based algorithm that can be used to solve large-scale SVM problems. Its weights are updated iteratively, and for the linear case, the update is as follows:

$$\begin{aligned} &\vec{w}_0 = \vec{0} \\ &\alpha_i = \begin{cases} \eta_i y_i & y_i(\vec{w}_i, \vec{x}_i) < 1 \\ 0 & y_i(\vec{w}, \vec{x}_i) \geq 1 \end{cases} \\ &\vec{w}_{i+1} = (1 - \eta_i \lambda)\vec{w}_i + \alpha_i \vec{x}_i \end{aligned} \qquad (6)$$

where $\vec{w}$ is the weight vector, λ is regularization parameter and η_i is a descent parameter decreasing according to some schedule. In our case, output vectors $y_i \in \{1;-1\}$ represent classes c_i for binary problem that alternate during training process, and input vectors consist of $\vec{x}_i = (l(F_1, c_i), l(F_2, c_i), ..., l(F_S, c_i))$, with different values of $l(F_k,c_i)$ as defined for Functions 1, 2 and 3.

3.3 Adding colors

The original Fern formulation in [8], as well as many algorithms for keypoint and texture recognition, operates on the greyscale images, completely ignoring color information provided by most image sensors of mobile phones these days. This is less important for the keypoint classification, since most of the keypoints are by default located in the region of varying intensity, near the edges or corners. If we consider texture or object recognition problem, color becomes much more important, since the texture to be recognized may contain large uniform areas, and only differ in color from the background. As a solution, we propose slight modification to the Fern binary checks. Instead of simply comparing intensities of two pixel with random offsets, each check is represented by the following formula:

$$f_i = I(r(\vec{I}(\vec{x} + \Delta\vec{x}_0)) \pm r(\vec{I}(\vec{x} + \Delta\vec{x}_1)) > 0) \qquad (7)$$

Copyright (c) 2013 Advances in Computer Science: an International Journal. All Rights Reserved.

```
Select $N*2$ random small offsets separated into M subsets of S elements $\Delta\vec{x}_{k,j,i}$, $M*S=N$, $i=0,1$, $k=1..M$, $j=1..S$
Select $N$ signs $s_i \in \{+1,-1\}$
Select $N$ random vectors with three elements $\vec{z}_{k,j}$
Set $\vec{w} = \vec{0}$
  for each class $c_i$ do
      Set $N_{c_i} = 0$ for all fern indices $F_k$
    for each image pixel $\vec{I}(\vec{x})$ in $c_i$ do
      for $k = 1$ to $M$ do $F_k = 0$
        for $j = 1$ to $S$ do
      $f_0 = (\vec{I}(\vec{x} + \Delta\vec{x}_{k,j,0}), \vec{z}_{k,j})$
      $f_1 = (\vec{I}(\vec{x} + \Delta\vec{x}_{k,j,1}), \vec{z}_{k,j})$
      $f = f_0 + s_{k,j} * f_1$
      $F_k = F_k * 2 + I(f > 0)$
        end for$N_{i,F_k} = N_{i,F_k} + 1$
      end for
      Update probabilities $\hat{P}(F_m = k|C = c_i)$ (eq. (3))
      Select and calculate $L$ as described in section 3.2
      Update $\vec{w}$ as described in eq. (6) or eq. (8), depending on the number of classes
    end for
  end for
```

(a)

```
Select $N*2$ random small offsets separated into M subsets of S elements $\Delta\vec{x}_{k,j,i}$, $M*S=N$, $i=0,1$, $k=1..M$, $j=1..S$
Select $N$ signs $s_i \in \{+1,-1\}$
Select $N$ random vectors with three elements $\vec{z}_{k,j}$
Set $\vec{w} = \vec{0}$
  for each class $c_i$ do
      Set $N_{c_i} = 0$ for all fern indices
    for each image patch $I_c$ in $c_i$ do
      for $k = 1$ to $M$ do $F_k = 0$
        for $j = 1$ to $S$ do
      $f_0 = (\vec{I}(\vec{x} + \Delta\vec{x}_{k,j,0}), \vec{z}_{k,j})$
      $f_1 = (\vec{I}(\vec{x} + \Delta\vec{x}_{k,j,1}), \vec{z}_{k,j})$
      $f = f_1 + s_{k,j} * f_2$
      $F_k = F_k * 2 + I(f > 0)$
        end for
      Store $F_k = F_k$
      $N_{i,F_k} = N_{i,F_k} + 1$
      end for
    end for
  end for
      Calculate probability estimates: $\hat{P}(F_m = k|C = c_i)$ (eq. (3))
      Select and calculate $L$ as described in section 3.2
  repeat
      Update $\vec{w}$ as described in eq. (6) or eq. (8), depending on the number of classes
  until convergence condition is satisfied
```

(b)

Figure 1: Variants of proposed training algorithm. $\vec{I}$ is a 3-component RGB image. (a) Interleaved training, weights are adjusted during Fern training. (b) Two separated training phases: weights are trained after Ferns training is completed.

where $\vec{I}(\vec{x} + \Delta\vec{x}_0)$and $\vec{I}(\vec{x} + \Delta\vec{x}_1)$ are 3-component color vectors for the feature offsets Δ_0 and Δ_1, correspondingly, $r(\vec{c}) = (\vec{z}, \vec{c})$, $\vec{z} = z_i$, $i = 1..3$ is a vector with randomly selected coefficients, and *I(cond)* is indicator function. Depending on the sign in the eq. (7), it can have what we call symmetrical (for +) and antisymmetrical (for -) forms. In order to preserve validity of the features, coefficients f for symmetrical features are selected so that $\sum_{i=1}^{3} z_i = 0$.

Several training experiments indicate that during training, Ferns containing mainly symmetrical or antisymmetrical features have larger resulting absolute values of SVM coefficients depending on whether the training area is flat or contains obvious intensity changes, correspondingly. This shows that the best ferns for a given pattern or keypoint can be selected by discarding ferns (and corresponding features) with the lowest w_k and adding newly selected random features.

3.4 Training

Training, then, can proceed in two ways.

1. The separate training. SVM weights are calculated after the joint probabilities have been estimated for all poses and texture positions. The advantages of this method include increased accuracy due to more precise probability estimates. The disadvantages are that either the features have to be extracted two times or the input vectors have to be saved, requiring higher memory consumption.
2. Interleaved training consists of adding each feature vector to both histograms for probability estimation and then SVM, according to eq. (6). This allows processing all input data in a single pass, and depending on implementation may allow for online adjustments, allowing model to change depending on the detected pattern. The resulting algorithms are shown in Figure 1.

3.5 Multiple classes

While the focus of our paper is on binary classification, most problems in image processing are not confined to only two classes. The keypoint recognition problem, for instance, can easily have several hundred detected keypoints, resulting in a large amount of classes. From eq. (6) it can be seen, that in linear case, NORMA training weights are altered in a way that changes value of the testing function in the direction of correct classification. This change only happens if the training vector is misclassified or lies within margin. Based on this, the update step for multiple classes can be formulated in a way

Copyright (c) 2013 Advances in Computer Science: an International Journal. All Rights Reserved.

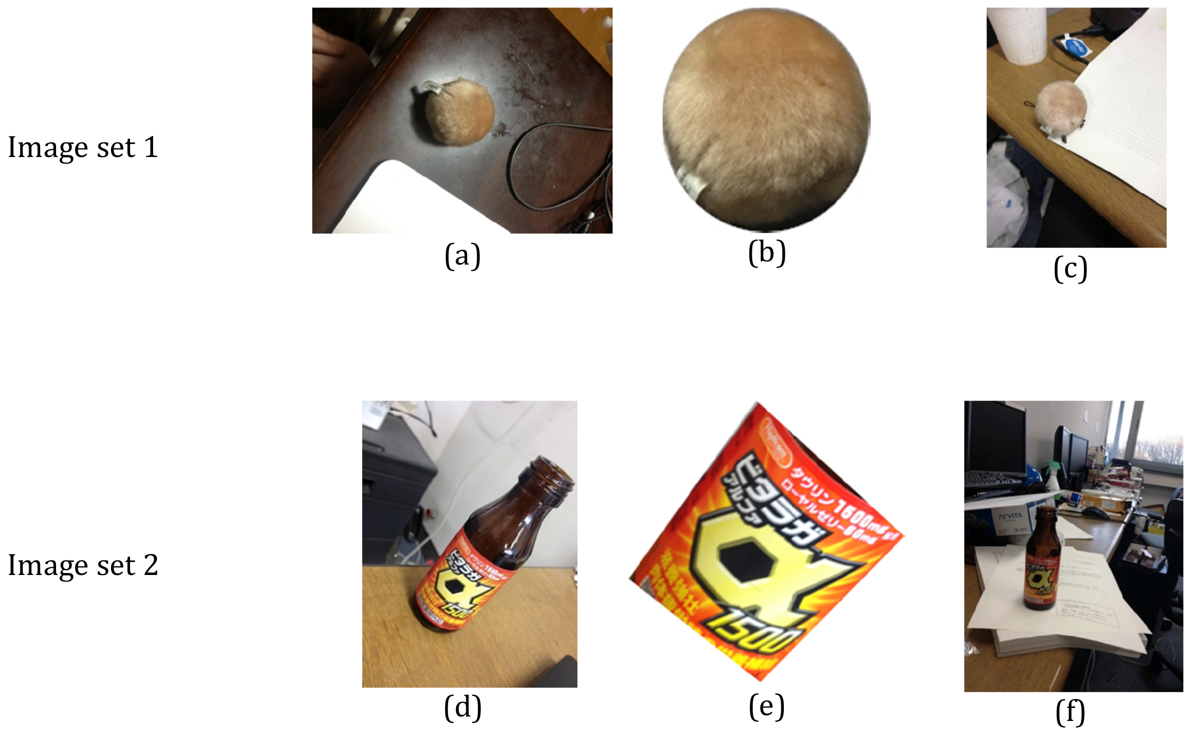


Figure 2: Examples of training images and clipped textures. (a) , (d) Images used for training. (b), (e) Texture area extracted for training (c), (f) Examples of the images used for evaluation.

that, in case of misclassification, estimated likelihood is reduced for the wrong class while it is increased for the correct class. The example of update step is then:

$$\hat{c}_k = arg\max_{c_t}(\vec{w}_i, \vec{l}_{c_t})$$
$$\hat{c}_s = arg\max_{c_t \neq \hat{c}_k}(\vec{w}_i, \vec{l}_{c_t})$$
$$\alpha_i = \begin{cases} 0 & c_k = c_i \ \&\& \ (\vec{w}_i, \vec{l}_{\hat{c}_k}) - (\vec{w}_i, \vec{l}_{\hat{c}_s}) > 1 \\ 1 & \text{otherwise} \end{cases} \qquad (8)$$
$$\vec{w}_{i+1} = (1 - \eta_i\lambda)\vec{w}_i + \alpha_i\eta_i(\vec{l}_{c_i} - \rho\vec{l}_{c_k})$$

where ρ is an additional parameter regulating ratio between the gradients for the correct and the incorrect classifications.

3.6 Experiments

In our experiments, we mainly concentrate on the task of binary texture recognition that we use for further implementation on the mobile device. For that, we select an image with a known textured area and train original Ferns as well as SVM-boosted Ferns by creating histograms of probability distributions for texture and background classes. This includes affine transformations of patches taken from both areas to make resulting marginal distribution pose-independent. We evaluate the accuracy by classifying several test images, for which the ground truth values were given by hand, and evaluating the ratio of misclassified pixels to the total number of pixels.

We perform tests of different methods on the two sets of images, 3 480x640 RGB images each, with 1 image used for training and two images with differing texture poses used for testing (examples given in Figure 2), comparing accuracy of original Ferns and SVM-boosted ferns for 3 values of l(F_k,c_i), for both interleaved and separate training phases. The results of completed training are presented in table 3. In the table log, prob and bin standing for likelihood functions 1, 2 and 3, respectively, applied to original Ferns (no weighting), wlog, etc. refer to weighted Ferns using separate training, and iwlog, etc. to weighted Ferns using interleaved training. It can be seen that all methods that use training provide increased accuracy over the original Ferns method, especially when the Ferns bit size and total amount of features are decreased. For particularly low values of fern size (below 6 bit), the binary method starts to outperform other methods of likelihood estimation. This is useful for applications where the amount of storage is limited, since it allows storing each feature in single bits rather than floating point values. It should be noted that this increase of accuracy comes

Copyright (c) 2013 Advances in Computer Science: an International Journal. All Rights Reserved.

Table 3: Average ratio of successfully recognized pixels to the total number of pixels in the test images. Sample test images for the image sets 1 and 2 are presented on 2c and 2f, respectively.

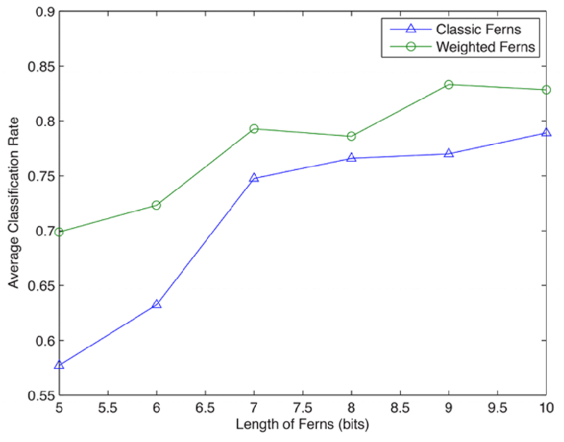
	Image set 1			Image set 2		
	6 bits, 30 Ferns	8 bits, 30 Ferns	8 bits, 50 Ferns	6 bits, 30 Ferns	8 bits, 30 Ferns	8 bits, 50 Ferns
log	0.76	0.78	0.80	0.58	0.57	0.58
prob	0.51	0.52	0.61	0.5	0.5	0.51
bin	0.62	0.69	0.73	0.61	0.61	0.63
wlog	0.83	0.84	0.88	0.78	0.81	0.90
wprob	0.81	0.81	0.82	0.9	0.9	0.92
wbin	0.93	0.92	0.92	0.88	0.89	0.92
iwlog	0.77	0.77	0.80	0.70	0.75	0.80
iwprob	0.60	0.63	0.75	0.81	0.82	0.85
iwbin	0.65	0.71	0.80	0.81	0.83	0.85

without any increase of computational cost during classification stage, since the likelihoods can be premultiplied by SVM coefficients in case of offline training, and multiplications replace logarithm calculation in case the online updates are used. To estimate the performance and accuracy of multiclass method, we perform the same tests as in the original Ferns article [8]. The results are shown in Figure 4. Our experiments have shown that while our algorithm outperforms the original on shorter Ferns and lower number of classes, benefits decrease as the amount of classes and available Ferns increases, indicating that for applications with larger available resources and stricter requirements to training times the original method could be preferable.

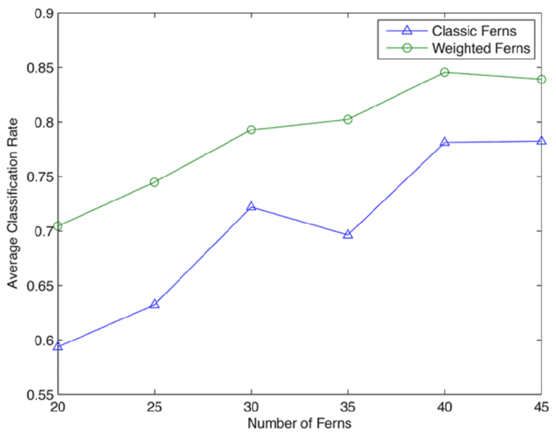
4. Implementing GPU-accelerated version of the algorithm on a mobile device

Recently, GPGPU (general purpose graphics processing unit) programming is becoming popular, since it allows designing low-cost high-speed parallel computational solution, by utilizing the fact that GPU are designed to perform a large number of identical tasks fast, such as polygon rendering. In the beginning, GPU computation has been confined to personal computers, since GPUs of mobile devices didn't allow custom programming. However, newer graphical libraries, such as now commonly used Open GL ES 2.0, allow the use of programmable shaders, small programs that run in parallel on GPU. In particular, fragment shaders that are run for each pixel of the output image are well suited for image processing tasks that use only local information, such as convolution with a small kernel. This has lead to an intensive research activity on using mobile GPU for high-speed image and video processing. For example [4] considered implementing SIFT algorithm on the mobile phones with Android operating system. They conclude that while using both CPU and GPU increases the performance on mobile devices, mobile platform remains very restrictive and requires a lot of effort from the programmer but does not achieve the same performance gains as observed for the PC. These restrictions, unfortunately, remain for current generation of the mobile phones. However, SIFT is not the best algorithm for parallel processing, though it certainly benefits from it. It requires repeated rescaling and convolving of the input image, and therefore uses a large amount of GPU iterations, increasing computational and memory costs. In this section, we show how to implement the texture recognition algorithm outlined in section 3.2, based on two-class SVM-boosted Ferns. Its computation is performed mostly by GPU, with

Copyright (c) 2013 Advances in Computer Science: an International Journal. All Rights Reserved.

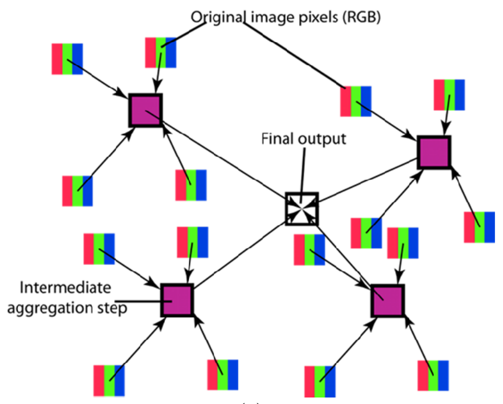


(a)

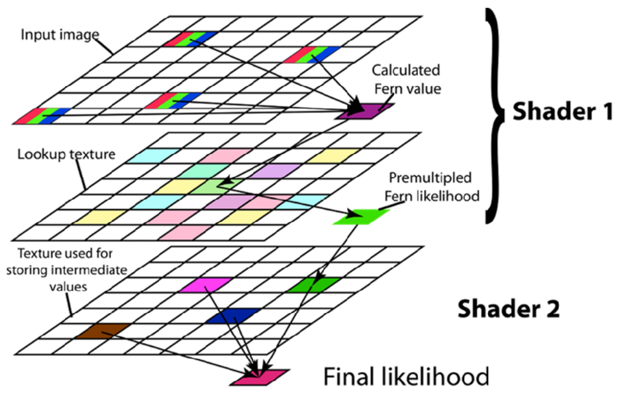


(b)

Figure 4. Results of testing multiclass approach described in section 3.5 with 50 classes. Ratio of correctly classified image patches to a total number of patches (accuracy ratio). (a) Dependence of accuracy ratio on a bit length of each Fern (25 Ferns total). (b) Dependence of accuracy ratio on a total number of Ferns (8 bits per Fern).



(a)



(b)

Figure 5. (a) Illustration of multilevel data aggregation. Four RGB pixels are processed by first layer, resulting in the intermediate value (purple). The next shader in chain then processes four intermediate values, aggregating data from a total of 16 pixels, and gets the result of the second level(shown as a white square). (b) Illustration of complete shader chain for likelihood estimation. Shader 1 calculates Ferns and looks up premultiplied binary values in an auxiliary lookup texture. Shader 2 aggregates outputs of Shader 1 into a single likelihood.

very little CPU participation. Since it allows estimating likelihood of a given pixel being part of texture are for every pixel of an image, the keypoint / region of interest detection step can be avoided.

4.1 Implementation details

At its core, the implemented algorithm is simple. Once the offline training is done, we have a set of probability distributions ($F_m = k|C = c_i$) and corresponding weights w_m for all Ferns, which can then be arranged into lookup tables and saved as reference textures in the video memory. To simplify uploading process, tables can be arranged and saved as PNG images beforehand (examples in Figure 6), or saved, for smaller number of Ferns, saved as text file.

Then, for classification, the fragment shader has to perform necessary binary tests to create Ferns for each pixel (using eq. (7)), form lookup indices and calculate the resulting likelihoods by summing over values fetched from lookup texture. Here, however, we run into several limitations of the OpenGL ES shader programming.

1. Relatively slow texture lookup. Looking up texel (texture element) values, especially when the coordinates are calculated in the fragment shader instead of being passed from vertex shader, is one of the most computationally expensive operations performed by GPU. Slow lookups limit the amount of binary tests that can be performed while maintaining real-time processing speed. It is therefore not possible to perform all binary feature evaluations and corresponding lookups necessary for Fern

Copyright (c) 2013 Advances in Computer Science: an International Journal. All Rights Reserved.

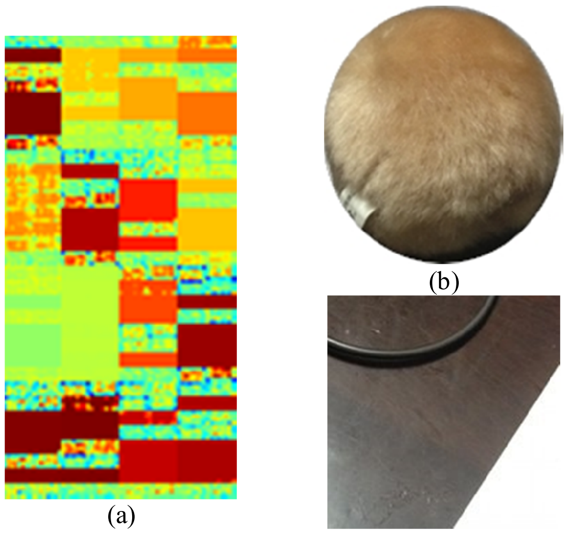


Figure 6. Weights and thresholded probabilities for 16 Ferns encoded in a PNG image ((a) for loading in mobile device. Each 16x32 rectangular region encodes a single 9-bit Fern (512 possible values). (b) Texture used for training (a). (c) Sample of background used for training (a)

evaluation in the single shader. The solution to this problem lies in separating evaluation into several stages, accumulating feature vectors and corresponding likelihood values over several iterations, as shown in Figure 5a. The drawback to this technique lies in the fact that it either introduces regularity in the feature offsets, limiting their randomness, or limits the total number of Ferns. This leads to additional dependence between separate Ferns, limiting contribution of each one in exchange for decreased computational costs. It also runs into a problem number of output constraints, outlined below.

2. Memory constraints. The amount of available video memory on the mobile devices is extremely limited, especially considering large size of Fern lookup tables. Since for the larger amount of Ferns they cannot be passed directly into the shader, in our implementation they are stored into an image and loaded into memory as a texture, introducing some ambiguity into the access routines, since transformation of the floating point coordinate values used in a shader to the integer texel coordinate is not exact. Still, as our experiments show below, this does not affect the accuracy of the results. Also, to further reduce the amount of the memory necessary, we store binary values resulting from thresholding outlined in section 3.2 instead of actual probabilities, which allows us to reduce storage requirements up to 8 times. Unfortunately, several experiments have shown that the simplest way of data packing, i.e. storing data as individual bits in the 32-bit texels, is not feasible due to the lack of bitwise operation or integer texture support in OpenGL ES 2.0 specification. This forced us to use somewhat less efficient method of storing individual bits premultiplied by SVM coefficients in color channels. This allows us to store data about 4 Fern values and corresponding coefficients in a single texture element, reducing the amount of texture fetches necessary and removing a multiplication operation from shader, reducing computational load.

3. Output constraints. The outputs of each fragment (pixel) shader in the OpenGL ES programming framework have to fit into a single pixel of the output texture, i.e. 4 bytes of data in floating point format, which is reduced to four 8-bit integers. Furthermore, the precision of floating point operations and variations in the driver implementation does not allow access to individual bits of the output.

4.2 Resulting algorithm

Our resulting algorithm uses chain of 2 shaders to transform original image into either likelihood estimation of each pixel belonging to an input texture or the thresholded value thereof. The complete chain is illustrated in Figure 5b. The chain uses two shaders. The first one calculates Fern values given pixel offsets used during training, and then recovers intermediate likelihood values from a lookup texture. In our case, each texel contains four values in four color channels available (RGBA), each value being either 0 (in case a Fern indicates background with greater likelihood) or 8-bit SVM coefficient for a corresponding Fern. The second shader sums outputs of the first one over the second set of predefined offsets, and outputs final likelihood of a given pixel belonging to a texture. An additional shader is then used to blend the likelihoods with original image for visualization. As can be seen, all of the image processing is completely performed on the GPU, freeing up CPU for additional tasks, such as possible online model training.

4.3 Implementation results

Our algorithm with the above modifications was implemented on the iPhone4S. A built-in video camera, running at 30 fps with the resolution of 640x480, was used as source of input images. The two sets of images shown in Figure 2 were used separately for training and recognition of the texture contained in each of them. The probability data from training was encoded in a set of PNG images each (example of an encoding image for 9-bit Ferns is presented in Figure 6, higher Fern length being used for improved clarity). For training, 64 9-bit ferns were used, and the joint distributions were then thresholded according to description in Section 3.2. Since no ground truth values were available, the video was evaluated visually, and the speed of the algorithm was measured by averaging the time passing between frames. Several screenshots captured during the operation are

Copyright (c) 2013 Advances in Computer Science: an International Journal. All Rights Reserved.

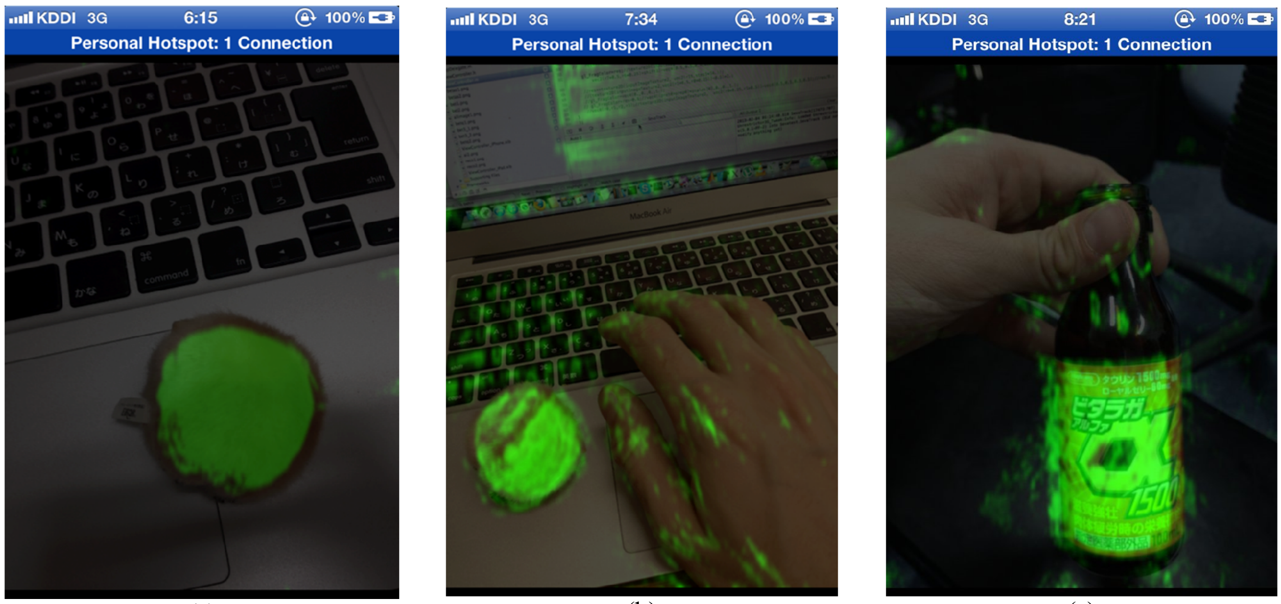

KDDI 3G 6:15 100%
Personal Hotspot: 1 Connection
KDDI 3G 7:34 100%
Personal Hotspot: 1 Connection
KDDI 3G 8:21 100%
Personal Hotspot: 1 Connection
(a)
(b)
(c)

Figure 7: Screenshots of texture recognition algorithm in operation. (a) , (b) are results of using weighted Ferns trained on texture from Figure 2b, (c) uses method trained on Figure 2e.

displayed in Figure 7. The average speed does not change with recognized texture, remaining stable at about 0.04 seconds per frame, that is, algorithm allows us to achieve 25fps for a relatively high-resolution video. As can be seen, our algorithm achieves high recognition accuracy for the trained texture despite change of pose, and achieves real-time speeds while processing all of the image pixels.

5. Conclusions and future work

We have introduced an algorithm to increase accuracy of methods based on semi-naive Bayesian approach, with the goal of using such methods for real-time image processing under computational limitation of CPU, Memory and GPU of a mobile device. Specifically, we modified Ferns algorithm to work with the support vector machine framework to combine estimated joint probabilities into class likelihood. The resulting algorithm keeps the simplicity and scalability of the Ferns algorithm and further achieves an increase in accuracy for applications with a lower number of features. The algorithm was also modified to allow texture recognition. This in turn allows us to implement proposed algorithm completely on a mobile device GPU, achieving high speed processing of 640x480 video feed, while maintaining an acceptable degree of accuracy.

References

[1] M.H. Chen. Importance Weighted Marginal Bayesian Posterior Density Estimation. Statistical analysis for stochastic modeling and simulation with applications to manufacturing. Purdue University, Department of Statistics, 1992.

[2] Eibe Frank, Mark Hall, and Bernhard Pfahringer. Locally weighted naive bayes. CoRR, abs/1212.2487, 2012.

[3] Yoav Freund and Robert E. Schapire. A decision-theoretic generalization of on-line learning and an application to boosting. J. Comput. Syst. Sci., 55(1):119–139, August 1997.

[4] G.-R. Kayombya. SIFT Feature Extraction on a Smartphone GPU using OpenGL ES 2.0. Master's thesis, MIT, Cambridge, MA, 2010.

[5] Jyrki Kivinen, Alexander J. Smola, and Robert C. Williamson. Online learning with kernels. IEEE Transactions on Signal Processing, 52(8):2165–2176, 2004.

[6] David G. Lowe. Distinctive image features from scale-invariant keypoints. International Journal of Computer Vision, 60:91–110, 2004.

[7] Timo Ojala, Matti Pietikainen, and Topi Maenpaa. Multiresolution gray-scale and rotation invariant texture classification with local binary patterns. IEEE Trans. Pattern Anal. Mach. Intell., 24(7):971–987, July 2002.

[8] Mustafa Ozuysal, Michael Calonder, Vincent Lepetit, and Pascal Fua. Fast keypoint recognition using random ferns. IEEE Trans. Pattern Anal. Mach. Intell., 32(3):448–461, March 2010.

[9] Shai Shalev-Shwartz, Yoram Singer, and Nathan Srebro. Pegasos: Primal estimated sub-gradient solver for svm. In Proceedings of the 24th international conference on Machine learning, ICML '07, pages 807–814, New York, NY, USA, 2007. ACM.

Copyright (c) 2013 Advances in Computer Science: an International Journal. All Rights Reserved.

[10] Vladimir N. Vapnik. The Nature of Statistical Learning Theory. Springer New York Inc., New York, NY, USA, 1995.
[11] Daniel Wagner, Gerhard Reitmayr, Alessandro Mulloni, Tom Drummond, and Dieter Schmalstieg. Real-time detection and tracking for augmented reality on mobile phones. IEEE Transactions on Visualization and Computer Graphics, 16(3):355–368, 2010.
[12] B. Williams, G. Klein, and I. Reid. Real-time SLAM relocalisation. In Proc. International Conference on Computer Vision, 2007.
[13] Vsevolod Yugov and Itsuo Kumazawa. Online boosting algorithm based on two-phase svm training. ISRN Signal Processing, 2012.
[14] Mustafa Ozuysal, Pascal Fua, and Vincent Lepetit. Fast keypoint recognition in ten lines of code. In In Proc. IEEE Conference on Computing Vision and Pattern Recognition, 2007.

Copyright (c) 2013 Advances in Computer Science: an International Journal. All Rights Reserved.