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

Vsevolod Yugov*, Itsuo Kumazawa†

March 19, 2013

Abstract

Feature point 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-nave Bayesian classification scheme have been developed. In our paper, we show how to further enhance these techniques 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 information processing. Furthermore, we demonstrate the advantages of our method by using it to achieve real-time texture recognition n a mobile device by utilizing parallel processing capabilities afforded by the device GPU.

1 Introduction

Image processing and computer vision on the mobile devices is a rapidly developing topic due to the increased processing power available. In particular, many modern devices are equipped with programmable GPU, allowing for parallel computation over the whole image. The drawback of using GPU lies in the fact that many state-of-the art algorithms used for image processing employ features that are too complex to be processed with the limited resources available. In such cases, algorithms using extremely simple features, like Ferns ([**?**], [**?**]) or Local Binary Patterns ([**?**]) are preferable. Such algorithms have been successfully adapted to mobile tracking problem in [**?**], however, it has been noted that the accuracy is degrading rapidly as the number of features is decreased. In our work, we concentrate on improving the accuracy of the method named FERNs, presented in [**?**] and [**?**], 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 Nave Bayesian way. We first consider a simple case of binary classification problem, which allows using Ferns algorithm for object and texture detection, similar to [**?**], 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 Nave Bayesian combination by a weighted Bayesian approach. In order to calculate weights we use online training algorithm for Support Vector Machines described in [**?**], due to its simplicity and low computational cost. We also 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. While this framework still leaves much to be desired, the ease of implementation of our method allows achieving high degree of performance and accuracy. There is no doubt that rapid advances in the mobile technology will soon allow even more complex image processing tasks to be done directly on the GPU without CPU intervention.

1.1 Outline of paper

The rest of the paper is organized as follows: in section **??** we give a brief overview of related works, including the works our paper is based on. In section **??**, we describe the Ferns algorithm in some detail. Then, in section **??** we introduce the main part of our algorithm, as well as some rationale behind it. In **??**, we show how SVM training methods can be expanded for the case of several classes. We further compare accuracy and simplicity of our method to the original in section **??**. Section **??** 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 **??**. Section **??** 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 ([**?**]), rely on the robustness of the features to certain kinds of transformations, specifically affine transformations. Others, such as LBP ([**?**]) and Ferns ([**?**]) may incorporate various poses in the sta-

*Tokyo Institute Of Technology, Dept. of Information Processing
†Dr. of Engineering, Imaging Science and Engineering Laboratory Tokyo Institute of Technology

tistical models used for classification. Both classes can be more or less efficiently used for the problem of pattern tracking on the mobile device, as shown in [**?**]. 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 fast decrease in accuracy of the method, rendering usage of the GPU (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, [**?**]) training methods to replace the semi-naive Bayesian approach with a weighted semi-nave approach. The work on SVM boosting ([**?**], based on [**?**]) shows that online SVM training as described in [**?**] and [**?**] can be used to easily increase the performance of other weak classifiers, and the works like [**?**] and [**?**] confirm that weighting represents a viable method for increasing accuracy of the Bayesian models. For our paper, instead of implementing a keypoint-based algorithm as in [**?**] and [**?**], we opt to use modified Ferns for a texture recognition problem, for which the use of LBP ([**?**]) 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 ([**?**]), by replacing the probabilities of a certain observed features with class numbers, which in our case result 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 [**?**], [**?**], which serves as a basis to our work. In these works, Ozuysal et. al show that images 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 nave 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, $F_m = \sum_{i=0}^{S-1} f_{m*S+i} * 2^i$. Each fern can then take values from 0 to $K = 2^S - 1$ The end result is semi-nave 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, [**?**] use uniform Dirichlet prior, resulting in a formula

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

, where N_{k,c_i} is the number of test samples in class c_i for which $F_m = k$, and N_{c_i} 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 likelihood 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 pose and lightning independent, 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 **??**) 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 **??**, though it can be adapted depending on the applications. Some examples of the adaptation are described in section **??**. 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 n can be one of the three functions:

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

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 those combinations have their own advantages and disadvantages. Using 2 results in the model closest to nave 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. However, there is an increased performance cost due to logarithm evaluation. 1 has decreased calculation cost, but little theoretical basis. 3 is the least performance-intensive option that sacrifices additional information present in options 1 and 2. If we consider option 3 in 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 [**?**], we use online support vector machine training methods [**?**], [**?**] to calculate the weighting coefficients n. In particular, for this problem we use NORMA ([**?**]) over Pegasos ([**?**]) method due to increased simplicity of 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, are:

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

, 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 $l(F_k, c_i)$ for 1, 2 and 3.

3.3 Adding colors

The original Fern formulation in [**?**], as well as many algorithms for keypoint and texture recognition operate on the greyscale images, completely ignoring color information provided by most images 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. Also, the data provided by hardware (camera on the mobile device) is naturally provided in the RGB format, so reducing it to greyscale not only reduces available information, but also incurs additional computational cost. As a solution, we propose slight modification to the Fern binary checks. Instead of simply comparing intensities, each check is represented by the following formula:

$$f_i = I(r(\vec{p}_0) \pm r(\vec{p}_1) > 0) \tag{7}$$

, where $\vec{p}_0$ and $\vec{p}_1$ are 3-component color vectors of the feature first and second pixel, $r(\vec{p}) = \sum_{p_i} z_i p_i$, z_i are randomly selected coefficients, and I is indicator function. Depending on the sign in the equation **??**, 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 zeomeanmath. Several training experiments indicate that during training, Ferns containing more 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. An example is illustrated on fig []. 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 possible 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 histogram for probability estimation and then SVM, according to eq. **??**. 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 demonstrated on figure **??**.

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. **??** 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 that, in case of misclassification, reduces estimated likelihood for wrong class while increasing it for correct class. The example of update step is then:

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

where ρ is an additional parameter regulating ratio between gradient towards correct and from incorrect classification.

Select $N * 2$ random small offsets $\vec{d}_{k,j,n}$
Select N signs $s_i \in \{+1, -1\}$
Select N random vectors with three elements $\vec{r}_{k,j}$
Set $\vec{w} = \vec{0}$
for each class c_i **do**
Set $N_{c_i} = 0$ for all ferns
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**
$f_1 = (\vec{I}_c(dk, j, 1), \vec{r}_{k,j})$
$f_2 = (\vec{I}_c(dk, j, 2), \vec{r}_{k,j})$
$f = f_1 + s_{k,j} * f_2$
$F_k = F_k * 2 + I(f > 0)$
end for$N_{i,F_k} = N_{i,F_k} + 1$
end for
Update probabilities $P(F_m = \hat{k}|C = c_i)$ (eq. (**??**))
Select and calculate L as described in section **??**
Update $\vec{w}$ as described in eq. (**??**) or eq. (**??**), depending on the number of classes
end for
end for

(a)

Select $N * 2$ random small offsets $\vec{d}_{k,j,n}$
Select N signs $s_i \in \{+1, -1\}$
Select N random vectors with three elements $\vec{r}_{k,j}$
Set $\vec{w} = \vec{0}$
for each class c_i **do**
Set $N_{c_i} = 0$ for all ferns
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**
$f_1 = (\vec{I}_c(dk, j, 1), \vec{r}_{k,j})$
$f_2 = (\vec{I}_c(dk, j, 2), \vec{r}_{k,j})$
$f = f_1 + s_{k,j} * f_2$
$F_k = F_k * 2 + I(f > 0)$
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: $P(F_m = \hat{k}|C = c_i)$ (eq. (**??**))
Select L
repeat
Update $\vec{w}$ as described in eq. (**??**) or eq. (**??**), depending on the number of classes
until convergence condition is satisfied

(b)

Figure 1: Proposed algorithm.

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 percentage of misclassified pixels. We perform tests on the same set of images (examples given on figure **??**), 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 averaged results of completed training are presented in table **??**. 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, the binary method starts to outperform other methods of likelihood estimation. This is useful for applications where the amount of strage is limited, since it allow storing data in single bits rather than floating point values. It should be noted that this increase of accuracy comes at little to no increase in computational cost during classification itself, since at most we only need to do additional M multiplications per classification, while removing the need to calculate logarithms. To estimate the performance and accuracy of multiclass method, we perform the same tests as in the original Ferns article [**?**]. The results are shown on fig []. Our experiments have shown that while our algorithm outperforms 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 higher available resources and stricter requirements to training times the original method could be preferrable.

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

Recently, GPGPU (general purpose GPU) programming is becoming more and more popular, since it allows designing low-cost high-speed parallel computational solution. While in the beginning, GPU computation has been confined to personal computers, with the widespread implementation of OpenGL ES 2.0 graphics library, which allows programmable shaders, on mobile devices, there have been significant research into using mobile GPU for high-speed image and video processing. For example [**?**] con-

(a) (b) (c)

(d) (e) (f)

Figure 2: **??** , **??** **??** The data separation corresponding to the set of classifiers in **??**. Background color shows class generated by classifier, form and color of data points show actual label y **??**

	Image 1			Image 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	6 bits, 30 Ferns	8 bits, 30 Ferns	8 bits, 50 Ferns
iwprob	0.60	0.63	0.75	6 bits, 30 Ferns	8 bits, 30 Ferns	8 bits, 50 Ferns
iwbin	0.65	0.71	0.80	6 bits, 30 Ferns	8 bits, 30 Ferns	8 bits, 50 Ferns

Figure 3

sider 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 on the PC. These restrictions, unfortunately, remain true 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 implementation of the texture recognition algorithm outlined in section **??**, based on two-class SVM-boosted Ferns. It is almost entirely based on the GPU processing, with very little CPU participation. Since it allows estimating likelihood of texture being present in every pixel of an image, the keypoint / ROI detection step of the most common algorithms can be completely omitted.

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 and corresponding weights for all Ferns, which can then be arranged into lookup tables and saved as textures in the video memory. Then, for classification, the fragment shader has to perform necessary binary tests for each pixel, form lookup indices and calculate the resulting likelihoods. Here, however, we run into several limitations of the OpenGL ES shader programming. 1. Relatively slow texture lookup. Looking up texel values, especially when the coordinates are calculated in the fragment shader instead of being passed from vertex shader. This limitation limits the amount of binary tests that can be performed. It is therefore not possible to perform all 200-300 feature evaluations and corresponding lookups necessary for Fern evaluation in the single shader. The solution to this problem lies in separating evaluation into several stages, accumulating feature vectors and corresponding likelihood

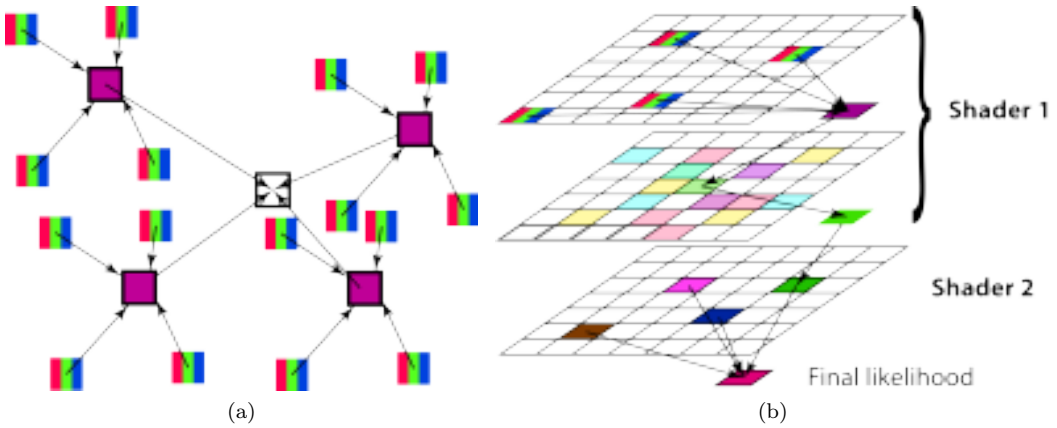

Figure 4: **??** , **??** **??** The data separation corresponding to the set of classifiers in **??**. Background color shows class generated by classifier, form and color of data points show actual label y **??**

values over several iterations, as shown on fig []. The drawback to this technique lies in the fact that it introduces regularity in the feature offsets, limiting their randomness. 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 3, 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 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 [0..1] floating point value to the texel coordinate is not exact. Still, as we show below, this does not impact resulting accuracy much. Also, to further decrease memory necessary, we store binary values resulting from thresholding outline in section [] instead of actual probabilities, which allows use 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 specification. This forced us to use somewhat more wasteful method of storing individual bits premultiplied by SVM coefficients. This, in turn, allows us to further reduce computational load of the single fragment shader. 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 3 shaders to transform original image into either likelihood estimation of each pixel belonging to an input texture or the thresholded value thereof. Schemes describing the action of each shader are illustrated on fig []. An additional shader is then used to blend the likelihoods with original image for visualization. The complete algorithm is illustrated on fig []. 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 frames. Two textures were trained separately and the probability data from training encoded in two png images each (fig []). For training, 64 6-bit ferns were used, and the joint distributions were then thresholded according to description in Section **??**. 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 displayed on fig []. The average speed does not change with texture, remaining stable at about 0.04 seconds per frame, that is, algorithm allows us to achieve 25fps on 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 introduce a method to increase accuracy of methods based on semi-nave Bayesian approach. Specifically, we modified Ferns algorithm to work with the support vector machine framework to combine estimated joint probabilities into class likelihood. The resulting algorithm shares the simplicity and scalability with the original, while achieving an increase in accuracy for applications with a lower number of features. The algorithm was also modified to allow texture and object recognition. This in turn allows us to implement proposed algorithm completely on a mobile device GPU, achieving high speed pro-

(a) (b) (c)

Figure 5: **??** , **??** Weights and thresholded probabilitites encoded in two images for loading in mobile device. **??** Texture used for training.

cessing of 640x480 video feed, while maintaining an acceptable degree of accuracy. In the future, we hope to refine developed method and test its application on such tasks as object segmentation or 3d tracking.

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 Pietikäinen, and Topi Mäenpää. 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.

[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 zuysal, 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.