Novel points:

1. Weighted semi-naïve Bayesian approach
2. Using online SVM to learn weights (for binary). Adjusting the same for multiclass distribution.
3. Comparing semi-naïve Bayesian to thresholded version.
4. Comparing precision of normal Ferns to our version
5. Using our method for texture recognition
6. Showing how our method works by implementing it on a mobile device in real time.

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-naïve 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.

Keywords: Image processing, computer vision, support vector machines.

Introduction.

Image processing and computer vision on the mobile devices is a rapidly developing topic due to 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 on 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 precision 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 Naïve Bayesian way.

We first consider a simple case of binary classification problem, which allows using Ferns algorithm for object and texture detection, similar to [lbp], 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 Naïve 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 to reduce 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 would soon allow even more complex image processing tasks to be done directly on the GPU without CPU intervention.

Outline of the 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 [], 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. We further compare accuracy and simplicity of our method to the original in []. In [], we show how SVM training methods can be expanded for the case of several classes. Section [] is devoted to our implementation of 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[] summarized the presented work and gives an outline of the possible future developments.

Related work.

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 statistical 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 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 [] training methods to replace semi-naive Bayesian approach with a weighted semi-naïve approach. The work on SVM boosting [] shows that online SVM training as described in [] and [] can be used to easily increase the performance of other weak classifiers, and the works [] 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 reason, 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.

Ferns

In this section, we briefly outline the algorithm for image patch recognition descried in [ferns], which serves as a basis to our work. 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 to combine posterior probabilities by using naïve Bayesian approach: {math}, where {math}. The end result is semi-naïve Bayesian approach, which models some but not all dependencies between features.

The training phase of Ferns estimates the class conditional probabilities for each Fern Fm and each class ci(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 {math}, where {math}. This prevents zero-valued probability estimates.

During classification, the binary features are extracted for each keypoint on the input image, and likelihood of each keypoint belonging to a certain class (i.e. being one of the keypoints of the original image) are estimated: {math}, where {math}

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.

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 {math}, we explore the possibility of weighting the likelihoods. Specifically, the formula for final likelihood for class $I$ is as follows:{math}, where $$ can stand for one of three things: 1. The joint probabilities $$ themselves, 2. The logarithms of joint probabilities $$ and 3. The binary-thresholded probabilities, 1 for the class with maximum joint probability over selected features, and -1 for all others.

All those combinations have their own advantages and disadvantages. Using $2$ results in the model closest to naïve Bayesian, with the weights more or less than one roughly representing positive and negative values of Spearman’s 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. 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 formulation is {math}, where {math}. In our case, input vectors consist of $$, with different values for $1$, $2$ and $3$, resulting, after substitution in {math}

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 high memory consumption. 2. Interleaved training consists of adding each feature vector to both histogram for probability estimation and then SVM, according to {math}. This allows to process 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 image {}.

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 {}, it can be seen, that in linear case, NORMA training weights can be separated into weights added when the input vector belonging to positive or negative class is misclassified, with the classification procedure then being {math}. From this formulation, the multiclass formula can be easily recovered: {math}. During the training, then, each class accumulates weights independently: {math}

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 $$, 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 original FERNs method, especially when the Ferns size and total amount of features are decreased. {For particularly low values of fern size, the binary method starts to outperform other methods.} 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 [].

{more conclusions}

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

Recently, GPGPU (general-processing 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 the mobile devices, there have been significant research into using mobile GPU for high-speed image and video processing. For example [] consider implementing SIFT algorithm on the mobile phones with Android operating system. They conclude that while using both CPU and GPU increases the performance on the 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.

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, 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 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. This does not allow us to aggregate data for different pixel offsets during a single pass, reducing the amount of features and Ferns we can process. The alternative is adding separate passes for each fern and storing results of fern probability estimation in the separate textures. This results in increased memory and calculation requirements, but as the memory capacity and processing speed of mobile devices increases, this might also become a viable software strategy.

{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: {math}, where {math} and $m$ are randomly selected coefficients. Depending on the sign of the equation {}, it can have what we call symmetrical (for +) and antisymmetrical (for -) forms. In order to preserve validity of the features, coefficients $$ for symmetrical features are selected so that {zeomeanmath}. Several training experiments indicate that during training, Ferns containing more symmetrical or antisymmetrical features have larger resulting absolute values of SVM coefficients depending on whether the training area is flat or contains obvious intensity changes, correspondingly. An example is illustrated on fig []. This shows that the best ferns for a given pattern or keypoint can be selected by discarding ferns (and corresponding features) with the lowest $$ and adding newly selected random features.

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 action of each shader are illustrated on fig []. An additional shader is then used to blend the likelihoods with original image for visualization. The resulting 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.

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.

Conclusion.

We introduce a method to increase accuracy of methods based on semi-naïve 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 increase in accuracy for 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 processing of 640x480 video feed, while maintaining 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.