Mach Learn (2011) 84:341–367
DOI 10.1007/s10994-011-5239-6

Onboard object recognition for planetary exploration

Michael C. Burl · Philipp G. Wetzler

Received: 2 July 2007 / Revised: 1 December 2010 / Accepted: 12 January 2011 /
Published online: 24 March 2011

Abstract Machine learning techniques have shown considerable promise for automating common visual inspection tasks such as the detection of human faces in cluttered scenes. Here, we examine whether similar techniques can be used (or adapted) for the problem of automatically locating geologic landforms in planetary images gathered by spacecraft. Beyond enabling more efficient and comprehensive ground analysis of down-linked data, we are aiming toward *perceptive spacecraft* that use onboard processing to autonomously analyze their collected imagery and take appropriate actions. In our current study, we have employed various supervised learning algorithms, including neural networks, ensemble methods, support vector machines (SVM), and continuously-scalable template models (CSTM) to derive detectors for craters from ground-truthed images. The resulting detectors are evaluated on a challenging set of Viking Orbiter images of Mars containing roughly one thousand craters. The SVM approach with normalized image patches provides detection and localization performance closest to that of human labelers and is shown to be substantially superior to boundary-based approaches such as the Hough transform. However, the run-time cost in applying the SVM solution in the standard way (spatial scanning in which the SVM is applied to each patch of the image on a window-by-window basis) is too high due both to the number of support vectors required and the number of test vectors generated by sliding a window across the data. We have developed an implementation using FFTs and the overlap-and-add technique, which can be used to efficiently apply SVMs to sensor data in resource-constrained environments such as on a spacecraft. The technique allows exact computation of the SVM decision function over an image using minimal RAM (typically less than 5% of the size of the image) and only $\mathcal{O}(n_s(\log_2 d + 11))$ real multiplications per pixel, where n_s

Editors: Amy McGovern and Kiri Wagstaff.

This work was performed in part at the Jet Propulsion Laboratory, California Institute of Technology, under contract with the National Aeronautics and Space Administration.

M.C. Burl (✉)
Jet Propulsion Laboratory, California Institute of Technology, Pasadena, USA
e-mail: Michael.C.Burl@jpl.nasa.gov

P.G. Wetzler
University of Colorado at Boulder, Boulder, USA

is the number of support vectors and d is the dimensionality of the vectors compared with $\mathcal{O}(n_s d)$ real multiplications per pixel for spatial scanning. Our approach is complementary to reduced set methods providing (in theory) a multiplicative gain in performance.

Keywords Support vector machines · Convolution · Run-time efficiency · Overlap-and-add · Crater detection

1 Introduction

Planetary spacecraft and rovers are typically equipped with one or more imaging devices ranging from visible wavelength cameras to hyperspectral imagers to synthetic aperture radar (SAR). Thus, images are a fundamental type of observation used to gain knowledge of distant planetary surfaces and processes. In the current paradigm of operation, images are collected, down-linked to Earth, and then examined manually by scientists. If desired, new command sequences can be uploaded to the spacecraft based on the content of the images.

Limitations with this paradigm have driven operations toward a new mode in which it is desirable for a spacecraft to autonomously perform some analysis of its collected data onboard and then take appropriate action. The Remote Agent Experiment (RAX) onboard DS1 (Bernard et al. 1999) demonstrated the ability of an onboard planner to control spacecraft operations and to respond autonomously to trigger events. Although this demonstration provided a key milestone in the effort to produce more capable spacecraft, the level of sensory interpretation (perception) for RAX was relatively simple, e.g., check the state of a switch ("I told the camera to turn on, is it on now?"). More recently the Autonomous Sciencecraft Experiment (ASE) on EO-1 (Chien et al. 2003) has demonstrated a higher level of onboard sensory perception (e.g., estimating the fraction of cloud obscuration, classifying hyperspectral pixels as snow-water-ice-land (Castano et al. 2005), etc.). The Deep Impact mission to Comet Tempel 1 used image analysis to provide guidance of the impactor toward its final target. DIMES software (Descent Imager Motion Estimation System) was successfully used with the MER landers to provide a vision-based estimate of horizontal velocities during the descent to Mars (Cheng et al. 2005). WATCH software, now running onboard the MER rovers, is autonomously looking for clouds and dust devils (Bornstein et al. 2007) based on a frame-differencing scheme. The Zoe rover has used onboard autonomy and data analysis to look for chlorophyll in the Atacama Desert of Chile (Wettergreen et al. 2005). The OASIS software (Onboard Autonomous Science Investigation System) is a research prototype that has been used successfully in a real-time rover testbed environment to autonomously detect and catalog rocks during traverses and select end-of-day targets for a narrow field-of-view instrument based on image analysis (Castano et al. 2006; Castano et al. 2007).

With the exception of the ASE classifiers, none of the space-deployed systems uses machine learning; instead, they rely on hand-coded solutions based on human insight to solve a specific problem. Even the ASE algorithms cannot recognize complex spatial patterns, as the algorithms are limited to pixelwise classification (possibly with some influence from neighboring pixels) based on a hyperspectral signature.

In an effort to generate greater scientific return from spacecraft and rovers, we are focused on improving the limited perception and interpretation capabilities that currently exist. In particular, we are interested in developing robust onboard detection and localization algorithms for some common types of geologic landforms such as craters, volcanoes, blocks, dunes, dark slope streaks, and plumes. Onboard detection of landforms

could be used for a variety of applications, including generation of summary products or statistics (a similar capability has been used in OASIS to produce summary rock catalogs Castano et al. 2007), feature-based motion estimation (Johnson and Matthies 1999; Leroy et al. 1999; Cheng et al. 2002, 2005), identification of targets for further analysis based on scientist-specified criteria or notions of interestingness, and geolocation using geologic landmarks (similar to Roweis 2007).

Although a hand-coded approach could be used to develop custom detectors for each landform of interest, it is perhaps more elegant to train a detector for the task of interest by presenting positive and negative examples to a learning algorithm. This approach was pioneered in the JARtool project (Burl et al. 1998), which used Principal Components Analysis and quadratic classifiers to learn detectors for small shield volcanoes in the Magellan SAR data of Venus. Sung and Poggio (1998) also proposed a learning approach for detecting volcanoes. Since these early efforts, more powerful techniques have emerged as evidenced by the considerable success of machine learning applied to human face detection (Rowley et al. 1998; Osuna et al. 1997; Viola and Jones 2001). We have attempted to harness similar methods from machine learning to develop robust landform detectors for planetary images.

In this paper, we focus primarily on the detection (and accurate localization and sizing) of impact craters. Although there are many interesting landforms, craters are the most abundant and are scientifically important because they provide significant clues about surface age and other processes (wind, lava, etc.) that have acted at various times and locations. In this regard the smaller craters are most useful for estimating the relative ages of different surface units. The Barlow *Catalog 1.0 of Martian Impact Craters* (Barlow 2003), which was created manually during the mid 1980's from Viking imagery, contains 40,000 craters but only goes down to size $\sim$ 5km. If the analysis were carried down to the smallest sizes supported by the Viking image resolution, one might find hundreds or thousands of craters in a single image. As an example, Fig. 1 shows a (1280×1280) Viking Orbiter image of a portion of the Martian surface. In this moderately-cratered region there are approximately 300 craters in the ground truth (with diameter ≥ 5 pixels).

We have employed various supervised learning algorithms, including feed-forward neural networks, ensemble methods, support vector machines (SVM) (Vapnik 1995), and continuously-scalable template models (CSTM) (Burl et al. 2001) to derive detectors for craters from ground-truthed images. The SVM approach with normalized image patches provides the best detection and localization performance. However, the run-time cost in applying the SVM solution in the standard way is too high due both to the number of support vectors required and the number of test vectors generated by sliding a window across the data. In this problem, as in many others, the number of support vectors amounts to nearly 10% of the training set. (Here, for the most accurate results the number of support vectors is on the order of thousands.) Also, to detect craters of various sizes, it is necessary to consider a multi-resolution pyramid representation (Adelson et al. 1984) of the image with the SVM applied at each pyramid level including "negative" levels to detect down to the smallest target size (5 pixels). Thus, the number of test patches that must be evaluated in such a scheme is close to ten million.

Methods such as reduced set vectors (Burges 1996; Tang and Mazzoni 2006) and the alternative formulation of ν-SVMs (Schölkopf et al. 2000) attack the problem by reducing the number of support vectors that need to be used. The approach in Romdhani et al. (2001) applies reduced sets in a sequential way to the problem of face detection in images. The key point is that many patches of the image clearly do not have a face so these can be eliminated from further consideration based on only a few reduced set vectors. Although large speed-ups are reported, it is not clear whether such large gains can be obtained in more target

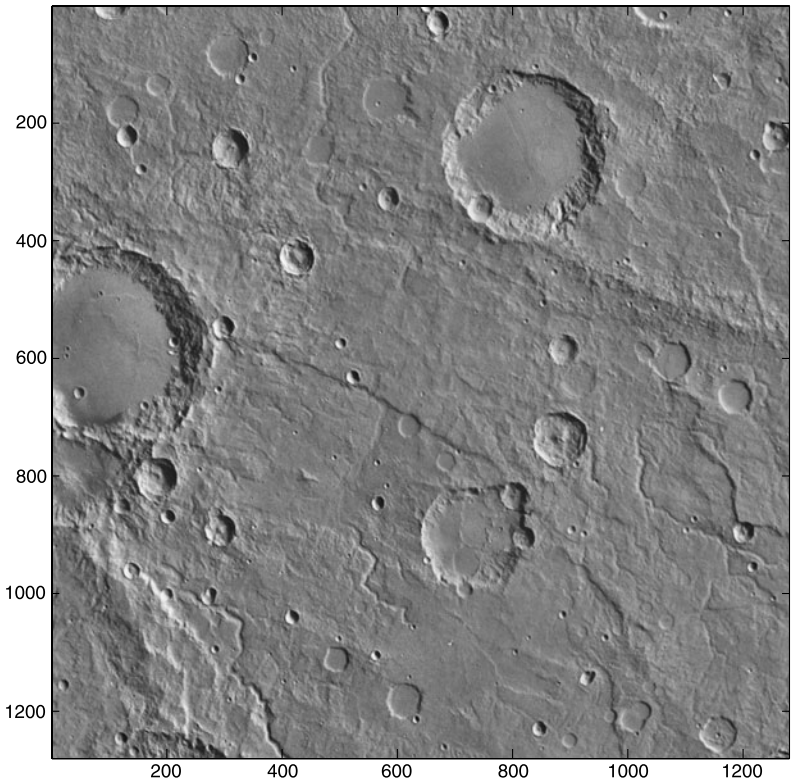

Fig. 1 Viking Orbiter image of a portion of the Martian surface. There are approximately 300 craters in the author-generated ground truth with diameter ≥5 pixels

rich images (e.g., in a planetary image with a high crater density) or with more complex backgrounds that are not easily rejected. In addition, such methods may not generalize to the case of using SVMs for regression. The query-tuned kernel machines in DeCoste (2003) are similar in spirit to the sequential reduced set approach in that they sequentially compute lower and upper bounds to enable early stopping in the calculation of the SVM classification function. Speed-ups from 2 to 64 fold are reported.

Our approach, which is complementary to traditional reduced set methods, attacks the problem by reusing computations on the test patches. In particular, our approach is based on blocked FFTs and the overlap-and-add technique from signal processing (Oppenheim and Schafer 1975). This technique allows exact computation of the SVM decision function over an image using minimal RAM (typically less than 5% of the size of the image) and only $\mathcal{O}(n_s(\log_2 d + 11))$ real multiplications per pixel, where n_s is the number of support vectors and d is the dimensionality of the vectors. The method is complementary to reduced set methods providing a multiplicative gain in performance. A related approach was developed contemporaneously by Kienzle et al. (2004). They also recognized that for many common kernels, the SVM calculation can be implemented with convolutions. However, their idea is to create a reduced set SVM in which the original two-dimensional support vectors are replaced with outer product approximations. Convolutions with these new support vectors are carried out in the spatial domain.

2 Related crater detection work

A fairly comprehensive bibliography of efforts to automate crater detection was compiled by Salamuniccar and Loncaric (2008). [Here do not consider crater detection from sources such as laser altimetry (Bue and Stepinski 2007), since the horizontal resolution for this type of data is currently only suitable for detecting medium to large craters.] Many of the methods have focused on looking for circular or elliptical arrangements of edges along the crater boundary, e.g., using the Hough transform (Hough 1962). Examples include the works of Cross (1988), Cheng et al. (2002), Honda and Konishi (2002), and Leroy et al. (1999). These boundary-based approaches seem to perform well under certain conditions, for example, for detecting medium to large craters (relative to the image resolution) when there is limited texture in the background due to other features or processes. However, the techniques break down for detecting smaller craters or when applied in more challenging terrain. In addition, these methods do not provide a solution for other landforms, which may not have well-defined boundaries or shapes.

An alternative to boundary-based detection is to look directly at the pixel-level pattern in an image patch. Along these lines, in the late 1960's Chapman (unpublished) made an early attempt to automatically detect lunar craters imaged at low sun angles by looking for adjacent bright-dark regions of the proper relative size. In our own prior work, we have used image-based techniques including principal components analysis (PCA) (Burl et al. 1998) and continuously-scalable template models (Burl et al. 2001) to detect various planetary features (volcanoes, craters, blocks). The learning techniques we consider here are similar in that they also work directly with image patches.

3 Learning algorithms

We formulate the crater detection problem in a supervised machine learning framework, which presumes the existence of a training set $\mathcal{T}$ consisting of tuples $(\mathbf{x}, y)$, where $\mathbf{x}$ is a fixed-length vector of observations and y is a label from the set $\{-1, +1\}$. The label -1 indicates that $\mathbf{x}$ is an instance of the "negative class" (`not-crater`) whereas $+1$ indicates that $\mathbf{x}$ is an instance of the "positive class" (`crater`). A learning algorithm examines the training set and attempts to produce a function $g(\cdot)$ that will correctly map future instances $\mathbf{x}$ to the correct class. Most of the learning algorithms we considered produce a continuous-valued output that can be thresholded to provide a class decision. By moving the threshold, one varies the tradeoff between detection and false alarm rates generating a receiver operating characteristic (ROC) curve. The learning algorithms we evaluated are briefly summarized below with pointers to more detailed references in the literature.

3.1 Feed-forward neural networks

Feed-forward neural networks consist of a set of *units* that produce an output value by applying a nonlinear squashing function (typically sigmoid or tanh) to a weighted combination of their inputs (Hertz et al. 1991). The units are commonly arranged in layers so that the outputs from one layer of units serve as the inputs to the next layer of units. The back-propagation algorithm (Rumelhart et al. 1986) offers a way to train the network by adjusting the network weights so as to minimize the error between the actual outputs of the network and the target outputs (y-values from the training set). Feed-forward neural networks have been used successfully in a number of visual pattern recognition applications (Rowley et al. 1998;

LeCun et al. 1998). Initially, we considered both the standard topology networks with a single hidden layer and convolutional networks similar to those used by LeCun on the MNIST data. Although we were able to reproduce the MNIST results with our implementation of convolutional neural networks, they did not perform well in initial experiments on the crater data, so they were not used in the full-scale experiments.

3.2 Ensemble methods: bagging and boosting

Ensemble methods offer a way to improve accuracy and reliability by training and combining multiple classifiers in an intelligent way. Bagging (Breiman 1996) simply trains multiple classifiers on different randomly selected subsets of the training set and averages or majority votes the individual decisions to produce a composite classifier. Boosting is a more directed approach in which examples that prove difficult to one classifier are reweighted and given enhanced attention by the next classifier. In our experiments, we used AdaBoost (Freund and Schapire 1995), which has previously been used successfully for face detection (Viola and Jones 2001). We used simple feedforward neural networks as the base learning algorithm and performed six rounds of boosting (we also experimented with more rounds, but did not find significantly better results). The weightings placed on the examples by the boosting algorithm were directly incorporated into the backpropagation/gradient descent algorithm used to train the base neural networks, forcing the base learners to "try harder" on the more heavily weighted examples. This use of boosting is consistent with the original methodology envisioned in the machine learning community. In the computer vision community, however, an alternative way of applying boosting has developed, largely fueled by the seminal work of Viola and Jones (2001) on face detection. In particular, they use boosting as a method of feature selection, where features typically consist of Haar wavelet filter responses. Although their results are very good on the face detection problem, we have not evaluated this approach for crater detection.

3.3 Support vector machines

The underlying idea of Support Vector Machines (Vapnik 1995; Burges 1998) is to find a hyperplane that optimally separates the positive from the negative examples, where optimality is defined by the size of the margin, i.e., the smallest distance between a positive and a negative example when mapped onto the hyperplane's normal vector. Using *kernels* (Smola and Schölkopf 2007), the input examples can be implicitly lifted into a higher-dimensional space in which the examples can be more easily separated. The hyperplane decision surface in this higher dimensional space corresponds to a non-linear surface in the original input space. These decision surfaces are specified in terms of a subset of the training examples (the support vectors) and a set of weights (Lagrange multipliers or alpha values). Support vector machines have provided very favorable results on a number of visual detection/classification problems, including face detection (Osuna et al. 1997) and the MNIST digit database.

In our experiments, we used the `libSVM` implementation (Chang and Lin 2001) to carry out the training. The kernel type and other parameters were selected through cross-validation. An SVM model found in this way consists of a set of *support vectors* s_i, a set of signed weighting coefficients $\tilde{\alpha}_i$, and a scalar b with the support vectors being a subset of the training examples (reshaped from 2D patches to 1D vectors). The nonlinear SVM decision function to be applied to a test vector $\mathbf{v}$ is given by:

$$g(\mathbf{v}) = \sum_{i=1}^{n_s} \tilde{\alpha}_i K(\mathbf{s}_i, \mathbf{v}) + b \tag{1}$$

where n_s is the number of support vectors and the $\tilde{\alpha}_i$'s are weights (positive or negative depending on the true class of $\mathbf{s}_i$).

The SVM models that were produced in training were far more complex than the models produced by the other methods. For example, we typically found SVM models with ∼5000 support vectors. Applying such models to image data is computationally demanding (due both to the large number of support vectors and the large number of test patches). Alternative formulations of the SVM objective function, such as the ν-SVM proposed by Schölkopf et al. (2000), allow more explicit control over the number of support vectors. However, as will be seen in Sect. 7 several experiments conducted with ν-SVM did not yield sufficiently accurate detection results.

3.4 CSTM

For comparison, we also evaluated an approximate implementation of the continuously-scalable template matching algorithm (CSTM) described in Burl et al. (2001). This algorithm, which was specifically developed for the problem of crater detection, is based on the idea of generating a family of matched filters at densely-sampled scales using a single example crater as the prototype for the family. The resulting family of filters is then applied to the images in an efficient way by applying PCA to compress the family into a smaller number of basis functions. We approximate this technique by omitting the PCA step and instead performing brute-force matching (calculation of correlation coefficient between each family member and each test patch). This modification sacrifices speed, but does not significantly alter accuracy. The choice of the prototypes for generating the filter families is made from the training set by sequentially choosing the positive example that covers the largest number of positive examples not already covered by an earlier prototype, where an example *covers* another if the correlation coefficient between the two exceeds some pre-determined threshold. Experiments were conducted with up to six prototypes (each defining a scale family).

4 Image-related issues

As noted above, the standard machine learning algorithms are designed to work with fixed-length instances $\mathbf{x}$; however, the size of a crater in an image can range from a few pixels in diameter to 400 pixels (or more). In addition, there are other issues that arise with image data that do not occur, say, in the UCI Machine Learning Repository (Asuncion and Newman 2007) or similar datasets.

4.1 Handling size variation

To manage the problem of size variation, we use a multi-scale "pyramid" representation of the image (Adelson et al. 1984), so that a crater with a particular diameter d_0 in the original resolution data will have an apparent diameter at level l of the image pyramid that is given by:

$$d_l = \frac{d_0}{\lambda^l} \tag{2}$$

where λ is the magnification factor between adjacent pyramid levels (original resolution is $l = 0$).

When choosing positive instances for the training set, a fixed-size window that is centered on the crater is selected from pyramid level l, where l is such that $d_l \in [d_{\min}, d_{\max}]$. At

detection time, the window is scanned[1] across each level of the pyramid and the pattern of pixel values under the window is used to classify the patch as being in the positive or negative class.

In choosing $d_{\min}$ and $d_{\max}$ it is important to note that if the crater diameter relative to the patch size is too small, then important details of the pattern will be lost (also, much of the patch will consist of irrelevant background pixels). If the crater diameter is too large relative to the patch size, then the patch will not contain enough context information. In our experiments, we used $d_{\min} = 7.125$, $d_{\max} = 8.55$, and $\lambda = 2^{0.25}$, with the window size fixed at (19×19) or at (17×17). The magnification factor λ insures that each crater falls within $[d_{\min}, d_{\max}]$ in exactly one of the pyramid levels.

Note that detecting craters at the smallest size (say $d_0 = 5$ pixels) benefits from supplying "negative" pyramid levels in which the original resolution data is interpolated to produce a larger image. We have shown experimentally in earlier work (unpublished) that using negative pyramid levels gives better performance than simply using smaller window sizes (e.g., 11×11) to detect the smallest craters.

4.2 Contrast normalization

Along with size variation, individual image patches may exhibit very different overall average brightness and contrast. Although one can leave it to the learning algorithm to decide to ignore such differences, it is also possible to explicitly normalize the patches. Given an image patch $\mathbf{x}$, we define the normalized version of the patch by:

$$\tilde{\mathbf{x}} = \frac{\mathbf{x} - \mu \mathbf{1}}{\sigma \sqrt{n}} \tag{3}$$

where μ is the mean of the values in $\mathbf{x}$, σ is the standard deviation, n is the number of values, and $\mathbf{1}$ is a vector of ones. Normalization, when used, is applied to both the training and test patches.

4.3 Defining the training set

To create the training sets for the learning algorithms, we start from a set of images for which all of the true craters have been labeled (circled). In most cases, the absolute ground truth is not known; therefore, labelings provided by a human expert are used as a surrogate for true ground truth. Since the size-frequency distribution of craters at the smallest sizes is of most interest for assessing relative ages of surfaces, we want to detect craters down to the smallest sizes. In this study a minimum target diameter of 5 pixels was used (any craters significantly smaller than this size that exist in the ground truth are marked as "don't care"). Given the training images and ground truth labelings, it is straightforward to extract the patches that are true positives from the image pyramids. To provide for some translation invariance, patches that are shifted by one pixel from the true crater centers are also included.

The choice of negative examples for the training set is more difficult. If the image size is $(z_r \times z_c)$, then taking every image patch that doesn't contain a centered crater of the proper size as a negative example would result in millions of negative examples (compared to thousands of positive examples). These numbers are problematic for two reasons: (i) the size of

[1] While this is conceptually the case, several of the implementations use more efficient FFT-based techniques as discussed in Sect. 5.

the training set (even learning algorithms that perform linearly in the number of examples may be close to infeasible to run with millions of examples) and (ii) the huge imbalance between positives and negatives (the learning algorithms will be tempted to simply classify everything as a negative). We deal with these problems by selecting a random subset from all the potential negative examples. Other schemes are certainly possible, for example, applying a coarse "attentional" mechanism to the images and using the false positives from the attentional mechanism as the negative training examples for supervised learning. The size of the negative set is chosen to be a factor of six or so larger than the size of the positive set to capture the greater variability among the negative patches, but still leave enough incentive for the learning algorithm to use the positive examples. When choosing negative examples, we first form the image pyramid for the training images. An exclusion region is placed around any true craters that would be expected to be detected at a given pyramid level. The negative examples are then selected from the non-excluded regions. Each non-excluded pixel in the pyramid has an equal chance of becoming the center of a negative example; hence, more negatives are selected from the finer pyramid levels. We also note that one of the algorithms, the continuously-scalable template model (CSTM), is trained only from positive examples; hence, this algorithm ignores any negative examples in the training set.

4.4 Non-maximum suppression through arbitration

Suppose we have a window centered over a crater, which has the right apparent size relative to the window. If the window is shifted by a pixel or two in any direction, the pattern within the shifted window is still going to look like a crater. Thus, it is to be expected that a single crater may cause multiple detections. Of course, if the goal is to count or catalog craters, then these duplicate detections are a problem. To remedy this situation, the collection of outputs from each detector (a set of circles of different sizes and locations) are passed through *arbitration*. Though there are different arbitration schemes possible, we use one based on spatial clustering. Detections for which the relative area of overlap[2] between circles exceeds a predetermined threshold are presumed to be due to the same object. The circle within a cluster that receives the highest confidence score from the classifier is considered to be the true object and the others are discarded as duplicates.

4.5 Efficient processing

A further complication with images is that the sheer volume of data makes efficient processing critical. This axiom is nowhere more true than in spacecraft applications, where the CPU speeds and RAM lag current desktop workstations by two to three orders of magnitude. For example, the Mars Reconnaissance Orbiter (MRO) launched in 2005 uses the RAD750 (BAE 2000) central processing unit (CPU), which for space-qualified, radiation-hardened processors is state-of-the-art. The RAD750 has 10^7 transistors (an order of magnitude more than in the RAD6000 processors used on the Spirit and Opportunity rovers), operates at a clock speed of 133–166 MHz, and is capable of executing 261 million instructions per second (MIPS). In terms of memory and storage, MRO carries a 160 Gbit solid state recorder (SSR). However, a single HiRISE camera image can consume 28 Gbits, so fewer than six full-size HiRISE images can be simultaneously held in the SSR.

[2]Since craters can be nested (a small crater can occur inside a larger crater), the relative area of overlap measures the ratio between the area of intersection between the two circles and the area of the larger of the two.

The straightforward method for evaluating a classifier decision function, $g(\cdot)$, over an image is *spatial scanning*, which involves moving a fixed-size window across the image (or across each level in a pyramid representation) and applying $g(\cdot)$ to the data $\mathbf{v}$ beneath the window at each scan position. For an SVM classifier, this approach requires that the kernel function K be evaluated n_s times per image patch (once for each support vector). For the common kernel functions (RBF, polynomial), much of the time is used to compute the linear dot products $\mathbf{s}_i^T \mathbf{v}$. This step is followed by applying the nonlinear kernel function, weighting each of the kernel results by the appropriate α value, and summing.

The computational complexity of direct spatial scanning per pixel[3] with an SVM is given approximately by:

$$\begin{aligned}\hat{\mathcal{C}}_{\text{scanning}} &= n_s(d\mathbf{m} + d\mathbf{a} + \mathbf{x} + \mathbf{m} + \mathbf{a}) \\ &= n_s((d+1)\mathbf{m} + (d+1)\mathbf{a} + \mathbf{x}) \end{aligned} \tag{4}$$

where n_s is the number of support vectors, d is the dimensionality of the support vectors, $\mathbf{m}$ is the computational cost of a multiply, $\mathbf{a}$ is the computational cost of an add, and $\mathbf{x}$ is the computational cost of the nonlinear kernel function, e.g., the $\exp(\cdot)$ in the RBF kernel. For typical values of n_s (thousands) and d (several hundred) this cost per pixel is already quite onerous. Fortunately, techniques from signal processing can be applied to significantly reduce the computational burden, while maintaining a relatively low memory footprint.

5 Efficient implementation of SVM for image data

As will be seen in Sect. 7, SVM applied to normalized image patches yields the best detection performance of the methods tested. However, the spatial scanning strategy is extremely slow. Hence, we have developed a more efficient implementation using FFT correlations. This method can be employed for kernels that are reducible to a nonlinear function applied to the *linear* dot product between the support vector and the test vector; RBF and polynomial kernels are two kernels that satisfy this property. For example, for the RBF kernel

$$K_{\text{RBF}}(\mathbf{s}, \mathbf{v}) = \exp\bigl(-\gamma \|\mathbf{s} - \mathbf{v}\|^2\bigr) \tag{5}$$

$$= \exp(-\gamma \cdot \mathbf{s} \cdot \mathbf{s}) \exp(-\gamma \cdot \mathbf{v} \cdot \mathbf{v}) \exp(+2\gamma \cdot \mathbf{s} \cdot \mathbf{v}) \tag{6}$$

For a given support vector $\mathbf{s}$, the first term is a constant. The second term is a function of the dot product of an image patch with itself, i.e., sum of the squares of the pixel values in the image patch. This term can be computed as a correlation between a 2D boxcar filter of ones and a version of the image in which pixel values are squared. There are several methods by which 2D boxcar filtering can be done very efficiently, e.g., separating the 2D boxcar filter into an outer product of 1D boxcar filters and doing the 1D correlations by recursion or by using the integral image approach (Viola and Jones 2001). Thus, the main work in evaluating the RBF kernel over an image comes down to the third term, which does indeed take the form of a nonlinear function applied to the linear dot product $\mathbf{s} \cdot \mathbf{v}$. The linear dot product can be computed over the entire image using correlation.

Essentially, the correlation approach swaps the inner and outer loops as compared with spatial scanning. Instead of computing the kernel function between every support vector

[3]We use a hat over a variable to denote cost per pixel; also, we neglect edge effects and suppose a valid test patch can be centered at each pixel of the image.

and one image patch, then moving on to the next image patch (as in spatial scanning), the correlation approach computes the kernel function between one support vector and every image patch, then proceeds to do the next support vector. Given the right type of kernel as discussed above, the latter can be accomplished through a linear correlation of the support vector (reshaped as a patch) over the image followed by a non-linear function.

At this point, the benefit of the correlation approach may not be apparent as it requires the same amount of computation as spatial scanning, plus memory overhead for an additional scratch array the same size as the image. The real benefit is realized by noting that the linear correlations can be performed efficiently using the Fast Fourier Transform (FFT) which results in a significant computational savings. By going a step further and utilizing the overlap-and-add trick from signal processing (partitioning the image into blocks or bands, computing results on the partitions, and then recombining), the memory footprint can also be kept quite low.

5.1 Full FFT approach

Consider an image $\mathbf{z}$ with z_r rows and z_c columns. The number $\mathbf{z}[i, j]$ is the pixel value at row i and column j, where $i \in [0, z_r - 1]$ and $j \in [0, z_c - 1]$. We will think of (i, j) as the coordinates of the center of the pixel, so the pixel extends over the range $(i - 0.5 : i + 0.5, j - 0.5 : j + 0.5)$.

Suppose we have a filter $\mathbf{f}$ with f_r rows and f_c columns. (Think of $\mathbf{f}$ as one of the support vectors reshaped as a 2D filter.) Direct computation of the correlation between $\mathbf{f}$ and $\mathbf{z}$ basically involves sliding $\mathbf{f}$ around on $\mathbf{z}$ and at each slide position, computing the inner product between $\mathbf{f}$ and the pixels of $\mathbf{z}$ that are directly underneath $\mathbf{f}$. Mathematically, this operation is expressed as follows:

$$\mathbf{r}[i, j] = \sum_{u=0}^{f_r-1} \sum_{v=0}^{f_c-1} \mathbf{f}[u, v] \cdot \mathbf{z}[i + u, j + v] \tag{7}$$

For preciseness, we call $\mathbf{r}$ the *corner-referenced* correlation since $\mathbf{r}[i, j]$ is the correlation[4] value obtained when the upper left corner of $\mathbf{f}$ is directly over pixel (i, j) of the image. Using an alternative reference point (e.g., the center point of the filter) requires some modifications, but is relatively straightforward. To avoid image edge effects, we will only compute and report *valid* correlations in which the full support of $\mathbf{f}$ is on top of $\mathbf{z}$ (no rows or columns of $\mathbf{f}$ overhang the image boundaries). Looking at (7) we see that this means the "valid" range of i and j in $\mathbf{r}[i, j]$ will be: $i \in [0, z_r - f_r]$ and $j \in [0, z_c - f_c]$.

The corner-referenced correlation of (7) can be computed by taking the FFT of the image and multiplying by the FFT of a flipped padded version of the filter and taking an inverse FFT as described in standard texts on signal processing, e.g., Oppenheim and Schafer (1975). Flipping of the filter is necessary to get correlation rather than convolution. The 2D flipping/padding formula is $\mathbf{g}[N_1 - n_1, N_2 - n_2] = \mathbf{f}[n_1, n_2]$, where the first index of $\mathbf{g}$ is interpreted modulo N_1 and the second index is interpreted modulo N_2.

5.2 Overlap-and-add

One drawback of the full FFT approach is that two complex-valued scratch arrays that are the same size as the image are needed to hold the intermediate FFT results. In an onboard

[4]Note that the correlation $\mathbf{r}$ is *not* the same as the *correlation coefficient*, which is usually denoted by ρ and takes values between -1 and 1.

setting, RAM is often at a premium[5] so it is desirable to consider additional tricks, such as overlap-and-add (Oppenheim and Schafer 1975), to reduce the memory footprint.

The main idea of overlap-and-add is to decompose the signal to be filtered into a set of mutually-exclusive blocks of contiguous pixels. In 2D this can be accomplished by partitioning the image into blocks or stripes (bands). For the block decomposition, let the image be divided into a set of $(Q_r \times Q_c)$ blocks where each block is of size $(h_r \times h_c)$ pixels. Let $\mathbf{b}_{q_r,q_c}$ be the (q_r, q_c) block, where $q_r \in [0, Q_r - 1]$ and $q_c \in [0, Q_c - 1]$. Then, the image $\mathbf{z}$ can be expressed as:

$$\mathbf{z} = \sum_{q_r=0}^{Q_r-1} \sum_{q_c=0}^{Q_c-1} \mathbf{b}_{q_r,q_c} \tag{8}$$

where

$$\mathbf{b}_{q_r,q_c}[i, j] \triangleq \begin{cases} \mathbf{z}(i, j) & \text{if } (i, j) \in \text{the } (q_r, q_c) \text{ block} \\ 0 & \text{otherwise} \end{cases} \tag{9}$$

Since correlation is a linear shift invariant operation,[6] the correlation of $\mathbf{f}$ with $\mathbf{z}$ can be expressed in terms of the correlations of $\mathbf{f}$ with the basic blocks $\mathbf{b}_{q_r,q_c}$ or with shifted versions of the basic blocks. By properly matching the sizes of the basic blocks according to the size of the filter and retaining an appropriate amount of zero padding in each block, the full correlation of $\mathbf{f}$ with $\mathbf{z}$ can be computed through a series of power of 2 FFTs of size $(2h_r \times 2h_c)$. Since all forward FFTs are applied to real-valued data, the Hermitian symmetry property can be used to save a factor of 2 in computation and storage. Pseudocode for the algorithm is presented in Table 1, while a more complete discussion of the implementation details is given in Burl and Wetzler (2004).

6 Computational and memory requirements

We now consider the complexity of computing the SVM decision over an image of size $(z_r \times z_c)$ using n_s support vectors of size $d = (f_r \times f_c)$. Note that the full cost on the multi-resolution pyramid representation will require summing the computational requirements over the various pyramid levels and taking the memory requirements from the finest-scale pyramid level. The cost of computing the pyramid itself or of any post-processing (nonmax suppression) is not included in the analysis.

6.1 Computational requirements

6.1.1 Spatial scanning method

As discussed earlier, in the direct method an $(f_r \times f_c)$ window is scanned across the image and at each scan position the kernel function between the data in the window and the n_s

[5] For example, in an automated image analysis application targeted for onboard science analysis on the MER rovers (Castano et al. 2007), the image processing algorithms are limited to 3 MB of RAM and the images themselves are nominally 1024×1024 with 12 bits per pixel.

[6] By linear, shift-invariant we mean: (1) the correlation of $\mathbf{f}$ with $\alpha_1\mathbf{z}_1 + \alpha_2\mathbf{z}_2$ is equal to $\alpha_1\mathbf{r}_1 + \alpha_2\mathbf{r}_2$, where $\mathbf{r}_i$ is the correlation of $\mathbf{f}$ with $\mathbf{z}_i$ and (2) the correlation of $\mathbf{f}$ with $\mathcal{S}(\mathbf{z})$ is equal to $\mathcal{S}(\mathbf{r})$ where $\mathcal{S}$ is a shift operator and $\mathbf{r}$ is the correlation of $\mathbf{f}$ with $\mathbf{z}$.

Table 1 Computing correlation over an image using blocked FFTs with overlap-and-add

Pseudocode

```
Input: filter f of size fr x fc
       image z of size zr x zc

Output: result r of size zr x zc

1. Choose hr = least power of 2 greater than or equal to fr-1.
   Choose hc = least power of 2 greater than or equal to fc-1.
     Let Qr = ceil(zr/hr) and Qc = ceil(zc/hc).
2. Place the filter values into an array g of size (2hr X 2hc)
       via the 2D flipping formula with N1 = 2hr and N2 = 2hc.
3. Compute the FFT of g -> G.
4. for qr = 0 : Qr-1
     for qc = 0 : Qc-1
   a.  Copy (read) the image block from row qr*hr : qr*hr+hr-1
         and col qc*hc : qc*hc+hc-1 into the bottom right
         corner of a (2hr X 2hc) work array a. The other parts
         should be filled with zeros.
   b.  Compute the FFT of a -> A.
   c.  Multiply G and A (complex point-by-point multiplication)
   d.  Compute the inverse FFT of the product -> p_{qr,qc}
   e.  Combine the partial results into the full result.
       r[(qr-1)hr:(qr+1)hr-1, (qc-1)hc:(qc+1)hc-1] += p_{qr,qc}
       where any indices on LHS that are out-of-bounds are
       omitted.
     end
   end
```

support vectors is computed. Neglecting edge effects and minor constants, the computational cost is:

$$\mathcal{C}_{\text{scanning}} = n_s((d+1)\mathbf{m} + (d+1)\mathbf{a} + \mathbf{x})z_r z_c \tag{10}$$

Expressing the cost as the number of operations per image pixel, we find:

$$\hat{\mathcal{C}}_{\text{scanning}} = n_s((d+1)\mathbf{m} + (d+1)\mathbf{a} + \mathbf{x}) \tag{11}$$

Measured in terms of real-valued multiplications, the cost per pixel is:

$$\hat{\mathcal{C}}_{\text{scanning}} \approx n_s d \tag{12}$$

since typically $d \gg 1$.

6.1.2 Full FFT method

For comparison purposes, we will first consider the full FFT method for calculating the SVM decision over an image as discussed in Sect. 5.1. One could consider pre-computing the FFT of each of the support vectors, storing the results in secondary storage (disk or solid state recorder), then loading the FFTs as needed. By fetching the FFT of the next filter while the current filter is being used in calculations, the impact of the (slow) secondary storage speed could be eliminated. Unfortunately, for large image sizes, storing all of the filter FFTs

could require a substantial amount of secondary storage ($n_s z_r z_c$). For the crater detection application, this approach is untenable: with $n_s \sim 5000$, $z_r = 1280$, $z_c = 1280$, we would need 50+ GB of secondary storage. Even with lossless compression, which might reduce this by a factor of two or three, it would still be too large. Also, with a multi-resolution pyramid representation of the image, it would be necessary to have the filter FFTs stored for each of the possible image sizes. Given these issues, pre-storage of the filter FFTs is not a reasonable assumption for the full FFT method. Thus, an additional $(z_r \times z_c)$ FFT for each of the support vectors is included in the computational cost.

For the full FFT method, we will need to take a $(z_r \times z_c)$ FFT of the image itself, a $(z_r \times z_c)$ FFT for each of the support vector filters, complex multiplies between the image FFT and each of the support vector filter FFTs at $z_r \cdot z_c$ pixels, and an inverse FFT of size $(z_r \times z_c)$ for each support vector. Thus, the cost measured in terms of the number of real multiplications is given by:

$$\begin{aligned} \mathcal{C}_{\text{full FFT}} = {} & (2n_s + 1) \cdot \mathcal{C}\left(\text{FFT}(z_r \times z_c)\right) \\ & + n_s z_r \cdot z_c \cdot 2 + 5 z_r z_c + n_s z_r \cdot z_c \end{aligned} \tag{13}$$

The second term represents the multiplication between the image FFT and each support vector filter FFT. A complex multiply corresponds to four real multiplications; however, with real-valued support vector filters and images, the Hermitian symmetry property of the DFT can be exploited to reduce the total number of real multiplies by half. The third term represents the calculation of the "self" inner product of each patch of the image with itself; these can be computed recursively at a constant cost of five operations per pixel independent of the actual filter size. Equation (13) also makes the assumption that the multiplication by $-\gamma$ and the exponential function that occur in the RBF kernel can be implemented as a table lookup, so that cost is neglected here. (However, see the discussion in Sect. 7.4 on this issue.) In addition, each kernel result must be weighted by its α value and summed over the support vectors. This adds the last term.

The computational complexity of a 2D complex FFT of size $(N_1 \times N_2)$ when N_1 and N_2 are powers of two (measured in number of real-valued multiplications) is listed in Oppenheim and Schafer (1975) as:

$$\mathcal{C}\left((N_1 \times N_2) \text{ FFT of complex data}\right) \approx \frac{N_1 N_2}{2} \log_2(N_1 \cdot N_2) \tag{14}$$

When all the filters/signals involved are real-valued, the DFT has a Hermitian symmetry property, which allows the computational complexity to be reduced by a factor of 2. Thus,

$$\mathcal{C}\left((N_1 \times N_2) \text{ FFT of real data}\right) \approx \frac{N_1 N_2}{4} \log_2(N_1 \cdot N_2) \tag{15}$$

Substituting this result into (13), we find:

$$\mathcal{C}_{\text{full FFT}} \approx n_s z_r z_c \left[\frac{1}{2} \log_2(z_r z_c) + 3\right] + \cdots \tag{16}$$

where the terms represented by ... are dominated by (much less than) $n_s z_r z_c$. Expressing the cost as the number of operations per image pixel, we obtain:

$$\hat{\mathcal{C}}_{\text{full FFT}} \approx n_s \left[\frac{1}{2} \log_2(z_r z_c) + 3\right] \tag{17}$$

6.1.3 Blocked FFT method

In the blocked method, we need a $(2h_r \times 2h_c)$ FFT for each of the $Q_r \cdot Q_c$ image blocks. These calculations can be performed once and cached to secondary storage so that they do not need to be repeated for each support vector filter. We also need a $(2h_r \times 2h_c)$ FFT for each of the support vector filters. Unlike in the full FFT case, these smaller fixed-size FFTs can be precomputed and cached to secondary storage. Loading the various FFT data from secondary storage can be overlapped with the ongoing calculations so the cost of using this slower storage can largely be ignored. Recall that in the full FFT method we ran into two problems with pre-storage of the filter FFTs. First, because the FFTs were the same size as the full image, the amount of storage was prohibitive for resource-constrained environments. Second, because the filtering is typically applied at multiple resolutions, the varying sizes of the FFTs needed posed additional storage problems. Both of these issues are resolved in the blocked FFT method. The amount of secondary storage required is only $8n_s h_r h_c$, which is roughly 8 times the amount of storage needed to keep the support vectors themselves. (Note: there is an extra factor of two since the FFTs are complex-valued.) Also, the size of the FFTs is based on the filter size and is therefore not linked to the size of the image, so a single set of filter FFTs can be used for all levels of an image pyramid. In the complexity calculations below, we will therefore assume that the filter FFTs are pre-calculated and stored in secondary storage. For purposes of estimating the computational complexity it is convenient to think of an outer loop over the support vectors with an inner loop going over blocks in the image. For each block, we must perform $(2h_r \times 2h_c)$ complex multiplications and an inverse FFT of the same size. We repeat these steps for each support vector, so the overall cost is:

$$\begin{aligned} \mathcal{C}_{\text{blocked FFT}} &= (n_s + 1) Q_r Q_c \cdot \mathcal{C}\left(\text{FFT}(2h_r \times 2h_c)\right) + n_s 8 Q_r Q_c h_r h_c + 5 z_r z_c + n_s z_r z_c \\ &\approx n_s \left[\log_2(h_r h_c) + 11\right] z_r z_c + \cdots \end{aligned} \tag{18}$$

where again the Hermitian symmetry property of the DFTs is exploited in calculating the FFT cost and the filter multiplication cost. Expressing the number of operations per image pixel, we obtain:

$$\hat{\mathcal{C}}_{\text{blocked FFT}} \approx n_s \left[\log_2(h_r h_c) + 11\right] \tag{19}$$

For typical filter and image sizes that we have used, the computational complexity of the blocked method is approximately a factor of 1.5 larger than the complexity of the full FFT method. The difference would have been closer to a factor of four if the full FFT method were allowed to take advantage of pre-computing the filter FFTs, but for reasons discussed earlier, that approach is disallowed for the full FFT method, but permitted for the blocked FFT method. The blocked method has advantages in terms of memory footprint, as well, which will be called out in more detail in the next section. Also, because of the spatial locality of the blocked method, it may have certain advantages over the full FFT method when used with support vector techniques that exploit early stopping such as the sequential reduced set approach described in Romdhani et al. (2001) or the query-tuned approach of (DeCoste 2003).

6.2 Memory requirements

In the memory calculations below, we do not include the size of the input image or the size of the output result in the memory requirements. Depending on the application, the relevant

portions of the image can be read from secondary storage (disk) as needed rather than having the whole image loaded into memory at once. Similarly, the relevant portions of the output could be calculated and written piecemeal to secondary storage.

6.2.1 Spatial scanning method

The direct spatial scanning method can be implemented with a negligible amount of primary memory.

6.2.2 Full FFT method

FFT calculations can be done "in-place" (using the same memory for the input and output). Hence, it is possible to compute the SVM decision over the whole image with two real-valued arrays with size equal to the size of the FFTs being performed. We use the real-valued FFT routines in Frigo and Johnson (2003) to take advantage of the fact that an $(N_1 \times N_2)$ FFT of a real-valued signal can be stored in a real array of size $(N_1 \times N_2)$ (called a half-complex array in Frigo and Johnson 2003). Thus, the total storage is $2z_r z_c$. For RBF kernels, which also require the inner products of each patch with itself, it is convenient to keep an extra array of these values in memory although it could also be saved to secondary storage and pulled up as needed.

6.2.3 Blocked FFT method

As above, the FFT calculations can be done "in-place". The amount of storage for doing the FFT operations is therefore similar: two real-valued arrays with size equal to the FFTs being performed. There is some additional complexity, however, that results from the use of overlap-and-add. Basically, the final inner product values between a filter and all the blocks in a particular band across the image is not completely known until the filter has been applied to the next band down. The set of partial results cannot be used in the kernel function (we need the full result since the kernel is nonlinear), so we must include storage for the partial results, which amounts to slightly more than one band across the image of height h_r. The memory needed is therefore: $8h_r h_c + h_r(z_c + h_c)$.

6.3 Summary of theoretical requirements

For easy reference, Table 2 summarizes the theoretical computational and memory requirements of each approach. Keep in mind that the theoretical computational requirements count only the number of real multiplications, so actual computation time may differ considerably from the predictions in the table.

7 Experiments

7.1 Methodology

Although there are a large number of planetary images available on the Internet, detailed ground truth in machine readable form is generally not available. Hence, we tested the various algorithms on a set of seven Viking Orbiter images of size 1280×1280, for which ground truth labellings were established several years ago by one of the authors (MCB). The images contain $\sim$1100 craters down to 5 pixels in diameter.

Table 2 Summary of computational and memory requirements for each approach for one pyramid level of size $(z_r \times z_c)$

Method	$\hat{\mathcal{C}}$	$\mathcal{M}$
Scanning	$n_s d$	ϵ
Full FFT	$\approx n_s[\frac{1}{2} \cdot \log_2(z_r z_c) + 3]$	$2z_r z_c$
Blocked FFT	$\approx n_s[\log_2(h_r h_c) + 11]$	$8h_r h_c + h_r(z_c + h_c)$

The second column, $\hat{\mathcal{C}}$ is the computational cost per pixel. The third column, $\mathcal{M}$ is the memory usage, but does not include storage space for the input image and the output result. The parameters are as follows: n_s = number of support vectors, d = dimensionality of support vectors = $f_r \cdot f_c$ = number of pixels in the support vector when reshaped as a 2D image patch, (z_r, z_c) = size of the image, (h_r, h_c) = least power of two greater than of equal to $(f_r - 1, f_c - 1)$, respectively. Note that the direct scanning method has essentially no working memory requirements. Also, these results do not include an analysis of the nonmaxima suppression (arbitration) algorithm that is applied to the raw detections as a post-processing step

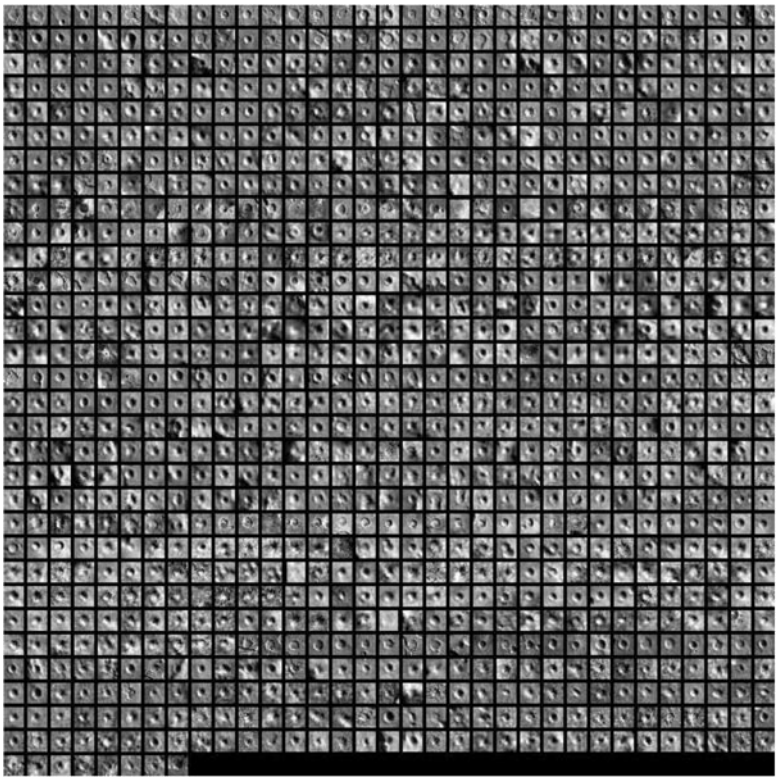

Fig. 2 The 1031 positive training examples from *fold1* shown as a mosaic of (17×17) chips. Note that these are selected from the appropriate pyramid level such that the size of the crater within the fixed-sized chip fits within $d_{\min}$ and $d_{\max}$

All experiments were conducted in a cross-validation manner. Six of the seven images were used to generate a training set. Figure 2 shows the 1031 positive training examples from fold1 as a collection of (17×17) chips. Figure 3 shows 1089 randomly chosen negative training examples from fold1. (In the experiments, randomly chosen negative examples are

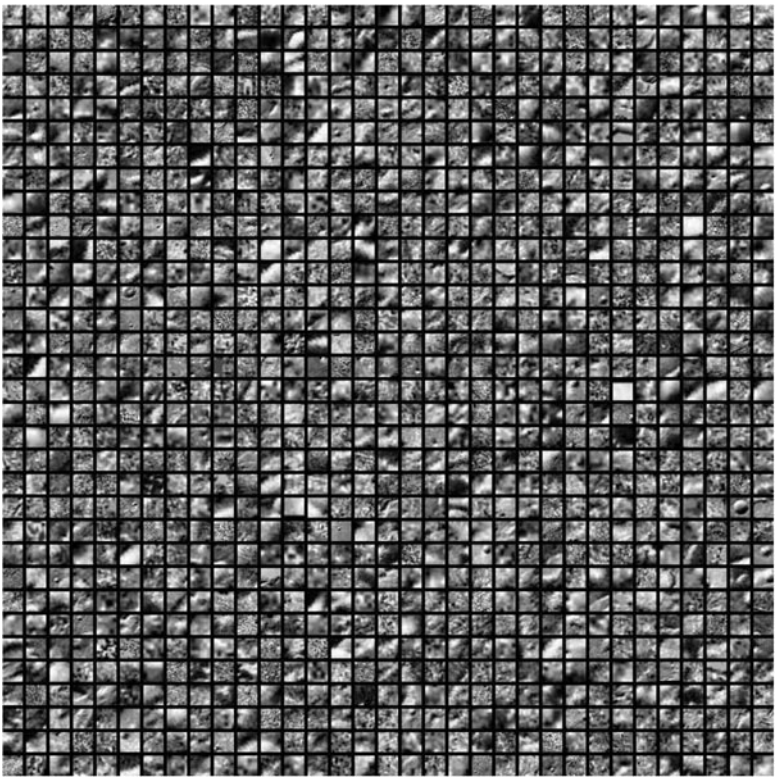

Fig. 3 A random sample of 1089 negative training examples from *fold1* shown as a mosaic of (17 × 17) chips

used from each fold. Also, in the experiments the number of negative examples is chosen to be six times the number of positive examples, but to keep the patch sizes in the figure from being too small, the number of false positives shown in Fig. 3 was chosen to be comparable to the number of true positives in Fig. 2.) Each learning algorithm derives a detector (the function $g(\cdot)$) from the training set. For algorithms that depend on additional parameters such as the regularization C and RBF bandwidth (γ) needed by SVM, these were determined by doing a cross-validation among the six images in the training fold. Once the parameters were chosen, a single classifier/detector was learned from the full six training fold images. Each detector was then applied to the seventh (testing fold) image to produce a labeling (set of circles of various sizes at various locations). The raw labeling was processed by the spatial clustering algorithm described earlier to remove duplicate detections. This process was repeated over each of the seven folds with each image taking a turn as the holdout (test image). The final labelings were automatically scored using a distance criteria based on relative area of overlap as in Wetzler et al. (2005):

$$d(C_1, C_2) = 1 - \sqrt{\mathcal{A}(C_1 \cap C_2)/\max(\mathcal{A}(C_1), \mathcal{A}(C_2))} \tag{20}$$

where C_1 and C_2 are two circles and $\mathcal{A}(\cdot)$ is the area.

For each ground truth crater, the best matching detection whose confidence is above the detection threshold T and whose overlap distance d is less than ω is identified. Any other

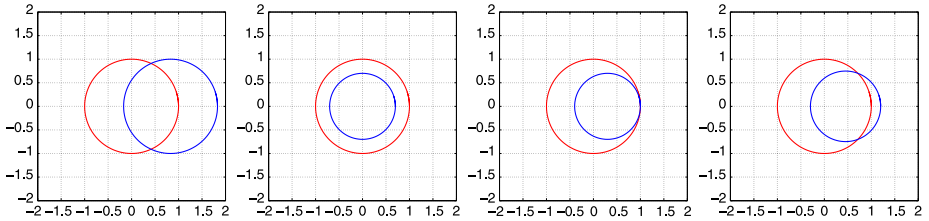

Fig. 4 (Color online)*Circles* that "just match" according to their relative area of overlap using $\omega = 0.3$. For *circles* with equal diameter and lateral offset, the offset can be at most about 0.415 times the diameter. For *concentric circles*, the smaller diameter must be at least 0.7 times the larger diameter

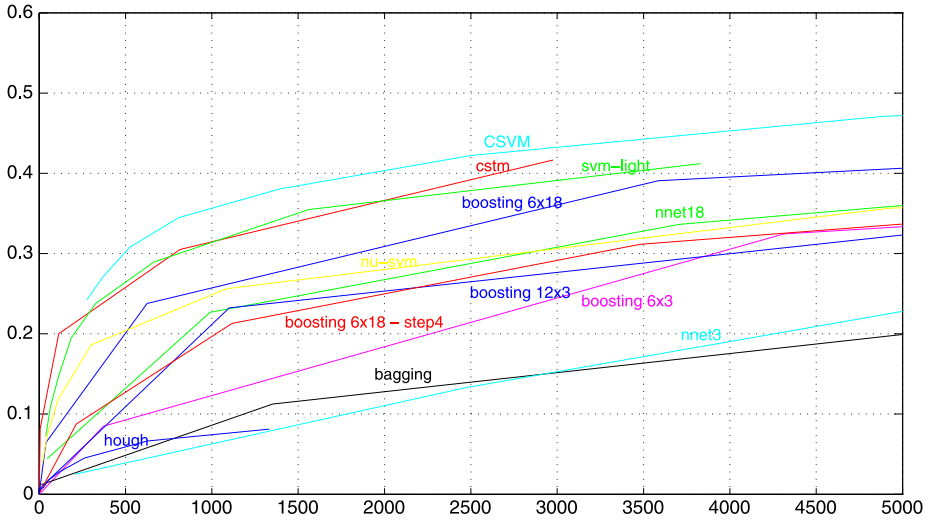

Fig. 5 (Color online) ROC plot showing probability of detection (*y-axis*) versus number of false alarms (*x-axis*) for various algorithms. The ROC performance for neural networks, bagging, boosting, and ν-SVM fall below the CSTM curve but above the Hough transform curve. The best result, labeled CSVM, is from SVM. For computational reasons, the SVM curve shown here was obtained by skipping the window four pixels at a time rather than one pixel as with the other methods

detections that match, but are not the best match, are reported as false alarms. Detections that are not matched to any ground truth marker are also counted as false alarms. The overlap distance threshold is set to $\omega = 0.3$ to require fairly accurate localization and sizing as shown in Fig. 4.

7.2 Initial experiments

Figure 5 presents the detection performance for various learning approaches as ROC curves, which show the trade-off between probability of detection (y-axis) and number of false alarms (x-axis) as the detection threshold T is varied.[7] Strictly speaking, these curves are

[7]If desired, these curves can be translated into recall-vs-precision through standard formulas: precision $= \frac{N_1 \cdot y}{N_1 \cdot y + x}$ and recall $= y$, where N_1 is the number of positive examples in the test set ($\sim$1100 here).

not ROC curves, but instead FROC (free-response ROC) curves, since the x-axis is a *number* of false alarms, rather than a *probability* of false alarm. Additionally, these curves assess not only detection ability, but also accurate localization and sizing. Hence, "chance performance" cannot be readily determined, except through extensive simulations where random circles are drawn from some size distribution, placed randomly on the image, and scored.

Clearly, the Hough transform does not perform particularly well. Various realizations of feed-forward neural networks, bagging, and boosting are above the Hough transform but well below CSTM. The curve `nnet3` is for a feed-forward neural network with 3 hidden units. While one would not expect great performance from such small networks, they were used as base learners for bagging and for some of the boosting experiments, so it is useful to see their standalone performance. A larger neural network `net18` was also evaluated. The curves `boosting_6x3`, `boosting_12x3`, and `boosting_6x18` show the results for several boosting experiments; the first number indicates the number of rounds and the second number indicates the number of hidden units used in the neural network base learners. A fourth boosting experiment is shown as `boosting_6x18-step4`. In this experiment the same detector from the `boosting_6x18` was used, but instead of scanning across the image one pixel at a time, the window was moved in discrete jumps of four pixels. The motivation behind this experiment was that all the SVM curves on this plot were generated using a skip-by-four strategy raising the concern that skip-by-four might somehow be improving performance (e.g., by limiting exposure to potential false alarms). The boosting skip-by-four experiment clearly shows that this was not the case and that skip-by-four was likely to be degrading the SVM performance. We then proceeded to develop the more computationally-efficient SVM implementation of Sect. 5.2, which allowed the SVM to skip-by-one with reasonable run time. Also, note that SVM even with skip-by-four (the curve labeled as `CSVM`) slightly outperforms `CSTM`. Using the ν-SVM formulation to explicitly reduce the number of support vectors (yellow curve), however, results in performance well below `CSTM` and the full SVM.

7.3 Expanded SVM experiments

After observing that the SVM approach, even with skip-by-four, was slightly better than the other approaches with skip-by-one, we went back and developed the more efficient implementation of Sect. 5 that allows the exact SVM result to be computed at every pixel. In Fig. 6, the curve labeled `SVM-step4` is from the initial skip-by-four experiment. The new skip-by-one curve is labeled `SVM-step1` and does indeed significantly outperform `SVM-step4`. In a final experiment, we trained the SVM detector with normalized patches and also normalized the test patches as part of the efficient FFT-based implementation procedure. This normalization produced further gains in performance as shown by the `SVM-norm` curve (this is a skip-by-one result, as well). For reference, a human performance point is also provided as a small filled circle at approximately ($x = 300$, $y = 0.75$). This point was acquired by comparing two "ground truth" labellings produced by one of the authors (MCB) several years apart. This point provides an estimate of the consistency with which humans can perform this task. Comparing the two human "ground truth" labellings shows that the inconsistencies are due to several factors: oversight, inconsistent handling of partial craters on the image boundaries, localization and sizing errors on the smaller craters, and genuine ambiguity in the image patches. The vertical line through the human performance point allows detailed comparison of the algorithms at the same false alarm rate as the human labeler.

Figure 7 shows the detections obtained with the SVM-norm method on `mi00n242`. Thick circles show SVM detections judged to be correct by the automatic scoring procedure

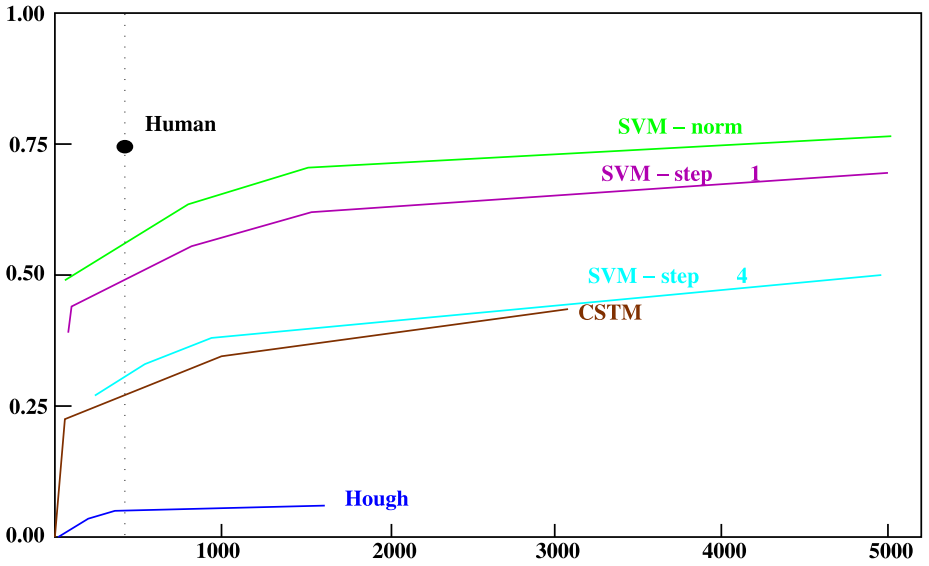

Fig. 6 (Color online) ROC plot showing probability of detection (*y-axis*) versus number of false alarms (*x-axis*) for various algorithms for the expanded set of SVM experiments. The SVM using normalized patches for training and testing and scanning by one pixel (`SVM-norm`) provides the best performance. For comparison, human performance (as determined by two labellings of the data set done three years apart) is shown as the *solid circle*. Neural networks, bagging, boosting, and ν-SVM are omitted from this diagram for clarity, but appear in Fig. 5

(true positives), while squares show false alarms (false positives). Missed craters (false negatives) are shown as thin circles. In this image there are 301 craters in the ground truth with diameter $\geq$5 pixels. The SVM-norm detector successfully finds and localizes 197 (65.5%) of these with 72 false alarms.

7.4 Run time performance

The run time performance for a reduced-size version of the SVM having 799 support vectors was measured on a Linux workstation with a 3 GHz AMD Opteron processor with two cores. All tests were conducted on a single pyramid level of size $(z_r \times z_c)$. Comparisons between the sliding window (aka direct scanning) method and the blocked FFT method were performed.

Due to the importance of 2D FFTs in the block-based SVM algorithm, we first evaluated the performance of the FFT implementation (FFTW 3.0) for FFTs of different sizes. Figure 8 shows the ratio between the actual FFT run time and the theoretical prediction (from (15)) converted from number of operations to time. For larger FFT sizes (say at or above (64×64)), the agreement is quite good (ratio close to 1). For an FFT size of (32×32) the actual cost was about 30% more than the theoretical prediction. For smaller sizes, the agreement was remarkably poor.

Next we performed an experiment in which an SVM with 799 support vectors of size (17×17) was applied to input images of different sizes. (One can think of these input images simply as different pyramid levels.) The run time was recorded for various parts of the algorithm:

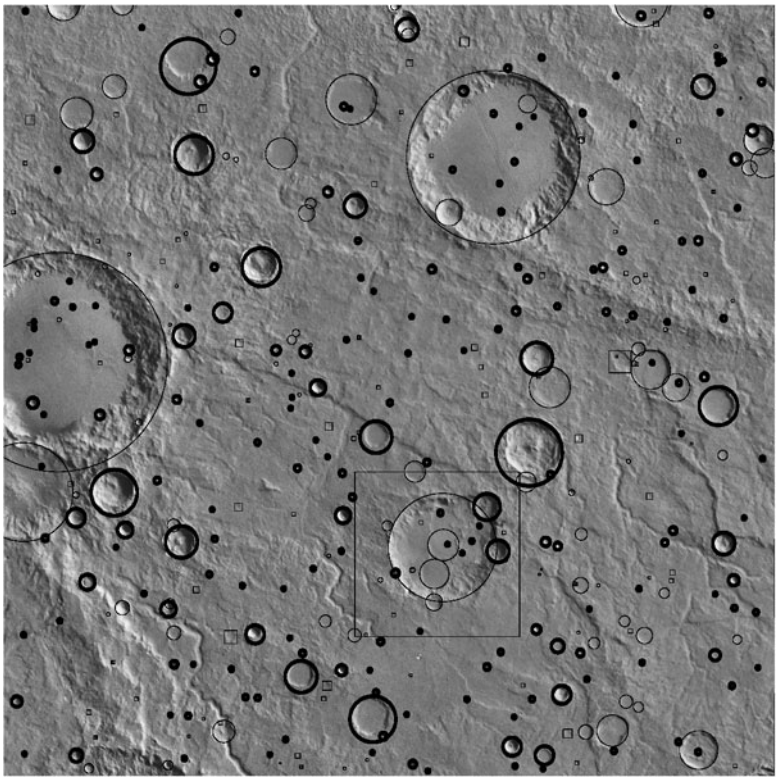

Fig. 7 Detections obtained with the SVM-norm method on `mi00n242`. *Thick circles* show true positives, squares show false positives, and *thin circles* show false negatives. The SVM-norm detector successfully finds and localizes 197 out of 301 craters (65.5%) with 72 false positives. From Wetzler et al. (2005)

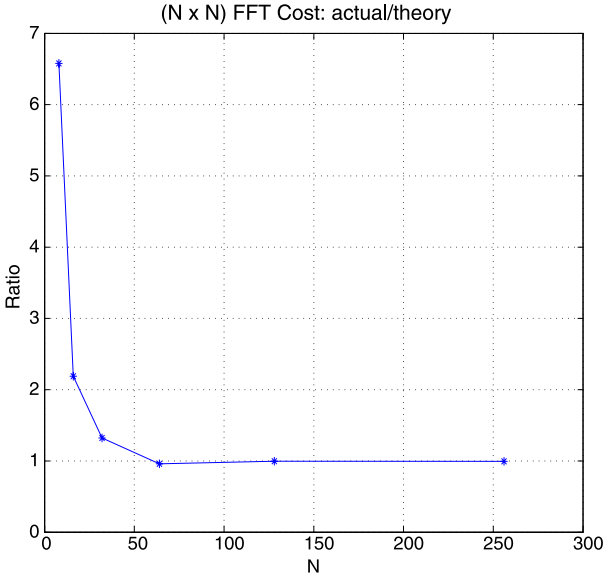

Fig. 8 (Color online) For FFT sizes at or above (64×64), the actual FFT time cost closely follows the theoretical model (15) with a conversion factor from operations to time of 3.9 ns/operation. For smaller sizes, the actual FFT cost is much greater than the theoretical prediction, presumably due to the greater impact of overhead. Note that for the blocked FFT approach, the FFT size is approximately twice the patch size, so an FFT of size (32×32) corresponds to a patch size of (17×17)

Table 3 Run times for different portions of the blocked FFT crater detection algorithm for an SVM with 799 support vectors of size (17 × 17)

$(z_r \times z_c)$	**IB**	**MULT**	**IFFT**	**OLAP**	**ACC**	**TOT**
160 × 160	0.003	0.838	0.645	0.581	0.936	3.197
320 × 320	0.014	3.349	2.584	2.509	5.036	14.216
640 × 640	0.057	13.511	10.443	11.419	21.103	59.369
1280 × 1280	0.227	53.776	41.329	50.052	73.521	230.283
2560 × 2560	0.965	214.880	165.564	192.468	299.233	918.492

Table 4 Run time performance of the sliding window and blocked FFT method using an SVM with 799 (17 × 17) support vectors over a single pyramid level of size $(z_r \times z_c)$. The blocked FFT provides an order of magnitude speedup

$(z_r \times z_c)$	Sliding Window (s)	Blocked FFT (s)	Speedup
160 × 160	30.949	3.197	9.7
320 × 320	138.318	14.216	9.7
640 × 640	582.123	59.369	9.8
1280 × 1280	2379.080	230.283	10.3
2560 × 2560	9657.727	918.492	10.5

- t_{IB}: time to compute and cache FFTs of image blocks
- t_{mult}: time to perform all the complex multiplications between the FFTs of image blocks and the support vector filter FFTs (step 4c in Table 1). This time comprises $Q_r \cdot Q_c \cdot n_s$ repetitions of the basic multiplication loop.
- t_{IFFT}: time to compute the inverse FFTs of the product of image block FFTs and support vector filter FFTs (step 4d in Table 1). This time comprises $Q_r \cdot Q_c \cdot n_s$ repetitions of the IFFT.
- t_{OLAP}: time to combine the partial results in the overlap and add step (step 4e in Table 1). This time comprises $Q_r \cdot Q_c \cdot n_s$ repetitions of the overlap and add step.
- t_{acc}: time to apply the nonlinear kernel operation, patch normalization, and accumulate results from the different support vector filters into the final result, i.e., the SVM decision statistic calculated at every pixel in the image; post-processing steps such as thresholding and non-max suppression were not included in the cost.
- t_{tot}: total time to go from input image to final result. Note: the total time may be greater than the sum of the parts, because the partial timings shown do not represent all steps.

Table 3 shows the run times for different portions of the blocked FFT algorithm as function of input image size. As expected the overall time, t_{tot}, scales in proportion to the number of input image pixels.

For comparison we also ran the sliding window approach at a single image size. Table 4 reports the run times. The speedup provided by the blocked FFT method is approximately a factor of ten.

Finally, we ran an experiment looking at the performance of the blocked FFT algorithm for different sized support vectors (image patches). For this experiment, an SVM with 799 support vectors of size $(f_r \times f_c)$ was applied to a single (640 × 640) image. The run times are reported in Table 5. Interestingly, the total run times do not closely follow the theoretical predictions of (19). Looking more closely at the partial results can offer some insight. First, the two main contributions to the theoretical prediction are the **MULT** and **IFFT** cost. The **IFFT** cost scales as expected for the larger FFT sizes, but as shown in Fig. 8, the actual

Table 5 Run times for different portions of the blocked FFT crater detection algorithm for an SVM with 799 support vectors of size ($f_r \times f_c$) over an image of size (640 × 640). Interestingly, the total run times do not follow well the theoretical predictions of (19). See discussion in text

($f_r \times f_c$)	**IB**	**MULT**	**IFFT**	**OLAP**	**ACC**	**TOT**
5 × 5	0.431	34.149	25.816	30.756	18.496	131.106
9 × 9	0.135	18.134	11.906	14.769	19.855	71.333
17 × 17	0.057	13.504	10.438	11.560	20.073	58.463
33 × 33	0.039	12.169	11.354	9.267	19.471	54.247
65 × 65	0.042	11.914	16.547	8.487	18.053	56.962
129 × 129	0.043	12.526	20.491	7.401	15.506	58.540

Table 6 Run time performance of the sliding window and blocked FFT method using an SVM with 799($f_r \times f_c$) support vectors over a single pyramid level of size (640 × 640). The blocked FFT provides a factor of 10 speedup for our standard (17 × 17) filter sizes. Sliding window results are omitted for the (65 × 65) and (129 × 129) cases due to excessive run times

($f_r \times f_c$)	Sliding Window (s)	Blocked FFT (s)	Speedup
5 × 5	68.252	131.106	0.5 (slower!)
9 × 9	179.083	71.333	2.5
17 × 17	582.123	58.463	10.0
33 × 33	2074.943	54.247	38.2
65 × 65	–	56.962	–
129 × 129	–	58.540	–

IFFT does not follow the theoretical model for smaller filter sizes. The **OLAP** cost, which is ignored in the theoretical calculation because it does not involve any multiplications actually adds a significant amount of time. Also, the **ACC** time, which contributed only a small amount to the theoretical cost, is significant. Much of the reason for the high **ACC** cost can be traced to computation of the exp(·) function for the RBF kernel. We remarked that this step could be implemented as a table lookup at minimal cost, but our current implementation still uses the standard math library function. The cost appears to be equivalent to six to eight real multiplies (per support vector per pixel) making this step non-negligible. Also, our current implementation did not exploit the Hermitian symmetry property when doing the required set of complex multiplies, so this term is a factor of two higher than necessary. (However, Hermitian symmetry was used in the FFT calculation—(15) vs. (14).)

For comparison, we also ran the sliding window approach applied at a given image size with different sized filters. Table 6 reports the run times. The speedup provided by the blocked FFT method is approximately a factor of 10 for our standard (17 × 17) filter size, and is substantially more for larger filter sizes.

As noted, the run times above are for a 3 GHz AMD Opteron processor with two cores. We currently have a rock segmentation algorithm called *Rockster* (Burl et al. 2010), running onboard the Mars Exploration Mission's Opportunity rover, which has a RAD6000 radiation-hardened processor operating at about 20 MIPS. For this application the speed difference between the RAD6000 and a desktop workstation is a factor of about 1500 (workstation time is 0.4 s, while RAD6000 time is 600 s). Even starting from a (160 × 160) image, the RAD6000 appears to be too slow to run the 800-support vector crater detection algorithm

in a reasonable amount of time. Moving up to the more modern RAD750, which is a 266 MIPS processor, might allow processing of a (160 × 160) image (with all pyramid levels) onboard in several tens of minutes. To handle larger sizes, it appears necessary to go to reduced set methods to find a more compact SVM with fewer support vectors. Fortunately, the reduced set speedup is synergistic with the FFT-based speedup we have considered thus far.

As an alternative to algorithmic improvements, Paul Springer and colleagues (Springer et al. 2008; Upchurch and Springer 2009) are investigating the potential use of multi-core architectures for compute-intensive spaceborne applications. The SVM-based object detection code has been one of the first applications ported to their testbed. Preliminary indications are that the object detection application is well-suited for the multicore environment (nearly perfect linear speedup) due to the ready divisibility of the input data and the fact that the majority of the work is in the FFT correlations. On the downside, however, the SVM code, at least as written, makes extensive use of floating point calculations and the current Tilera Tile64 testbed platform does not have floating point capability in hardware leading to a substantial performance hit (Upchurch and Springer 2009). But multi-core architectures are another potential pathway for getting the computation time down far enough to be practical for space-based operation on larger image sizes.

8 Conclusion

Enabling spacecraft with more human-like perception capabilities will open up new avenues of exploration and increased science return on investment. We have shown that state-of-the-art learning algorithms such as the support vector machine (SVM) algorithm of Vapnik (1995) can produce crater detection performance close to that of skilled humans in at least a restricted version of the problem space. The SVM approach was significantly better than the other methods tried, including a boundary-based (Hough transform) algorithm and a patch-based algorithm (CSTM) designed specifically for the purpose of detecting craters. A major drawback with the SVM approach, however, is that the strong detection performance comes at a relatively high computational cost. The number of support vectors in the most accurate learned models was ∼5000. Using ν-SVM (Schölkopf et al. 2000) to explicitly control the number of support vectors led to SVMs with a few hundred support vectors; however, the detection performance was significantly degraded. Using a blocked-FFT method and the overlap-and-add technique to implement the SVM detector enabled exact computation of the SVM decision function with an order of magnitude reduced computational complexity and minimal memory footprint, moving the algorithm closer to feasibility for onboard applications.

As noted in Sect. 1, there are other approaches directed toward improving the run-time performance of SVMs, such as Burges (1996), Schölkopf et al. (2000), Romdhani et al. (2001), DeCoste (2003), Kienzle et al. (2004), Tang and Mazzoni (2006). We note that the reduced set method and the ν-SVM method can be used in conjunction with the blocked FFT method providing multiplicative speed-up. The sequential methods, which effectively reduce the average number of support vectors that are evaluated per sample, can also benefit from our approach provided that the spatial distribution of target-like objects or patterns in the data is sufficiently sparse.

Acknowledgements The authors wish to thank W. Merline, C. Chapman, B. Enke, and W. Colwell of the Southwest Research Institute for assistance with the crater detection application. The authors also thank R. Honda of Kochi University for generating the Hough transform results.

References

Adelson, E. H., Anderson, C. H., Bergen, J. R., Burt, P. J., & Ogden, J. M. (1984). Pyramid methods in image processing. *RCA Engineer*, *29*(6), 33–41.

Asuncion, A., & Newman, D. J. (2007). UCI Machine Learning Repository. University of California, School of Information and Computer Sciences, Irvine http://www.ics.uci.edu/~mlearn/MLRepository.html.

BAE Systems, & Product Sheet (2009) RAD750 family of radiation-hardened products. http://www.baesystems.com/BAEProd/groups/public/documents/bae_publication/bae_pdf_eis_rad750.pdf, last accessed (2009/11/13).

Barlow, N. G. (2003). Revision of the catalog of large martian impact craters. In *Sixth Int. Conf. on Mars*.

Bernard, D., Dorais, G., Gamble, E., Kanefsky, B., Kurien, J., Man, G. K., Millar, W., Muscettola, N., Nayak, P., Rajan, K., Rouquette, N., Smith, B., Taylor, W., & Tung, Y.-W. (1999). Spacecraft autonomy flight experience: the DS1 remote agent experiment. In *Proceedings of AIAA*, Albuquerque, NM.

Bornstein, B., Fukunaga, A., Castano, A., Biesiadecki, J., Castano, R., Chien, S., Greeley, R., Whelley, P., Neakrase, L., & Lemmon, M. (2007). Onboard science on the Mars exploration rovers: cloud and dust devil detection. In *Lunar and Planetary Science XXXVIII*.

Breiman, L. (1996). Bagging predictors. *Machine Learning*, *24*(1), 123–140.

Bue, B., & Stepinski, T. F. (2007). Machine detection of martian impact craters from digital topography. *IEEE Transactions on Geoscience and Remote Sensing (TGRS)*, *45*(1), 265–274.

Burges, C. (1998). A tutorial on support vector machines for pattern recognition. *Data Mining and Knowledge Discovery*, *2*(2), 121–167.

Burges, C. (1996). Simplified support vector decision rules. In *13th Int. Conf. on Machine Learning* (pp. 71–77).

Burl, M. C., Asker, L., Smyth, P., Fayyad, U. M., Perona, P., Crumpler, L., & Aubele, J. (1998). Learning to recognize volcanoes on Venus. *ML*, *30*(2/3), 165–194.

Burl, M. C., Merline, W. J., Colwell, W., Bierhaus, E. B., & Chapman, C. R. (2001). Automated detection of craters and other geological features. In *Int. symposium on artificial intelligence, robotics and automation for space*, June 2001.

Burl, M. C., Thompson, D., Bornstein, B., & deGranville, C. (2010). Rockster-MER: memory-efficient onboard rock segmentation (Technical Memo). B-0142, also JPL New Software Report, NPO-#47954, Nov. 16.

Burl, M. C., & Wetzler, P. G. (2004). Resource-constrained application of support vector machines to sensor data. In *Workshop on data mining in resource-constrained environments, held in conjunction with the SIAM int. conference on data mining*. Apr. 2004.

Castano, R., Mazzoni, D., Tang, N., Doggett, T., Chien, S., Greeley, R., Cichy, B., & Davies, A. (2005). Learning classifiers for science event detection in remote sensing imagery. In *iSAIRAS*, Munich, Germany.

Castano, R., Anderson, R. C., Estlin, T., Gaines, D., Bornstein, B., Burl, M., Chouinard, C., Thompson, D., Castano, A., & Judd, M. (2006). End-of-day science with the OASIS system. In *Eos Trans. AGU, Fall Meet. Suppl.* (pp. P41B–1264), Abstract. San Francisco, CA, Dec. 2006.

Castano, R., Estlin, T., Gaines, D., Chouinard, C., Bornstein, B., Anderson, R. C., Burl, M., Thompson, D., Castano, A., & Judd, M. (2007). Onboard autonomous rover science. In *IEEE aerospace conference*, Big Sky, MT, Mar. 2007.

Chang, C.-C., & Lin, C.-J. (2001). LIBSVM: a library for support vector machines. http://www.csie.ntu.edu.tw/cjlin/libsvm.

Cheng, Y., Johnson, A. E., Matthies, L. H., & Olson, C. F. (2002). Optical landmark detection for spacecraft navigation. In *13th annual AAS/AIAA space flight mech mtg.*, Feb. 2002.

Cheng, Y., Johnson, A. E., & Matthies, L. H. (2005). MER-DIMES: a planetary application of computer vision. In *IEEE computer society conference on computer vision and pattern recognition (CVPR)*.

Chien, S., Sherwood, R., Tran, D., Castano, R., Cichy, B., Davies, A., Rabideau, G., Tang, N., Burl, M., Mandl, D., Frye, S., Hingemihle, J., D'Augustino, J., Bote, R., Trout, B., Schulman, S., Ungar, S., Van Gaasback, J., Boyer, D., Griffin, M., Burke, H., Greeley, R., Doggett, T., Williams, K., Baker, V., & Dohm, J. (2003). Autonomous science on the EO-1 mission. In *iSAIRAS*, Nara, Japan, May 2003.

Cross, A. M. (1988). Detection of circular geological features using the Hough transform. *International Journal of Remote Sensing*, *9*(9), 1519–1528.

DeCoste, D. (2003). Anytime query-tuned kernel machines via Cholesky factorization. In *SIAM int. conf. on data mining*, San Francisco, CA, Apr. 2003.

Freund, Y., & Schapire, R. E. (1995). A decision-theoretic generalization of on-line learning and an application to boosting. *Journal of Computer and System Sciences*, *55*(1), 119–139.

Frigo, M., & Johnson, S. (2003). FFTW User's Manual.

Hertz, J., Krogh, A., & Palmer, R. (1991). *Introduction to the theory of neural computation*. Boulder: Westview Press.

Honda, R., & Konishi, O. (2002). Data mining system for planetary images crater detection and categorization. citeseer.nj.nec.com/436870.html.

Hough, P. V. C. (1962). Method and means for recognizing complex patterns. U.S. Patent 3069654, Dec. 1962.

Johnson, A. E., & Matthies, L. E. (1999). Precise image-based motion estimation for autonomous small body exploration. In *iSAIRAS* (pp. 627–634).

Kienzle, W., Bakir, G., Franz, M., & Schölkopf, B. (2004). Efficient approximations for support vector machines in object detection. In C. E. Rasmussen et al. (Eds.), *DAGM 2004*, *Lecture Notes in Computer Sciences* (vol. 3175, pp. 54–61). Berlin: Springer.

LeCun, Y., Bottou, L., Bengio, Y., & Haffner, P. (1998). Gradient-based learning applied to document recognition. In *Proc. of the IEEE*, Nov. 1998.

Leroy, B., Medioni, G., Johnson, A. E., & Matthies, L. H. (1999). Crater detection for autonomous landing on asteroids. In *Workshop on perception for mobile agents*, Jun. 1999.

Oppenheim, A. V., & Schafer, R. W. (1975). *Digital signal processing*. New York: Prentice-Hall.

Osuna, E., Freund, R., & Girosi, F. (1997). Training support vector machines: an application to face detection. In *IEEE computer society conference on computer vision and pattern recognition (CVPR)* (pp. 130–136).

Romdhani, S., Torr, P., Schölkopf, B., & Blake, A. (2001). Computationally efficient face detection. In *International conference on computer vision (ICCV)*

Roweis, S. (2007). Making the sky searchable: rapid indexing for automated astrometry. In *Data mining for aeronautics, space, and exploration systems conference (DMASES)*, Mountain View, CA.

Rowley, H. A., Baluja, S., & Kanade, T. (1998). Neural-network based face detection. *IEEE Transactions on Pattern Analysis and Machine Inteligence*, *20*(1), 23–38.

Rumelhart, D. E., Hinton, G. E., & Williams, R. J. (1986). Learning representations by backpropagating errors. *Nature*, *323*, 533–536.

Salamuniccar, G., & Loncaric, S. (2008). Open framework for objective evaluation of crater detection algorithms with first test-field subsystem based on MOLA data. *Advances in Space Research*, *42*(1), 6–19.

Schölkopf, B., Smola, A. J., Williamson, R. C., & Bartlett, P. L. (2000). New support vector algorithms. *Neural Computation*, *12*, 1207–1245.

Smola, A., & Schölkopf, B. (Eds.) (2007) http://www.kernel-machines.org/.

Springer, P., Upchurch, E., Stalzer, M., Mauch, S., McCorquodale, J., Lindheim, J., & Burl, M. C. (2008). Porting some key Caltech and JPL applications to a PS3 cluster—a wild ride. In *High performance embedded computing workshop*, Lexington, MA, Sep. 2008.

Sung, K.-K., & Poggio, T. (1998). Example-based learning for view-based human face detection. *IEEE Transactions on Pattern Analysis and Machine Inteligence*, *20*(1), 39–51.

Tang, B., & Mazzoni, D. (2006). Multiclass reduced-set support vector machines. In *ICML* (pp. 921–928).

Upchurch, E., & Springer, P. (2009). Multi-core architectures for emerging NASA applications: some results for Tilera's tile 64 and maestro. In *Space mission challenges for information technology (SMC-IT)*, Pasadena, CA, Jul. 2009.

Vapnik, V. (1995). *The nature of statistical learning theory*. New York: Springer.

Viola, P., & Jones, M. (2001). Rapid object detection using a boosted cascade of simple features. In *CVPR*.

Wettergreen, D., Cabrol, N., Baskaran, V., Calderon, F., Heys, S., Jonak, D., Luders, R., Pane, D., Smith, T., Teza, J., Tomkins, P., Villa, D., Williams, C., & Wagner, M. D. (2005). Second experiments in the robotic investigation of life in the Atacama desert of Chile. In *Eighth int. symp. on ai, robotics, and automation in space (iSAIRAS)*, Sep. 2005.

Wetzler, P., Honda, R., Enke, B., Merline, W. J., Chapman, C. R., & Burl, M. C. (2005). Learning to recognize small impact craters. In *Seventh IEEE workshop on applications of computer vision (WACV/MOTION'05)* (pp. 178–184). Breckenridge, CO, Jan. 2005.