Feature Construction Using Evolution-COnstructed Features for General Object Recognition

Kirt D. Lillywhite
Brigham Young University - Provo

Follow this and additional works at: https://scholarsarchive.byu.edu/etd
Part of the Electrical and Computer Engineering Commons

BYU ScholarsArchive Citation
https://scholarsarchive.byu.edu/etd/2974

This Dissertation is brought to you for free and open access by BYU ScholarsArchive. It has been accepted for inclusion in All Theses and Dissertations by an authorized administrator of BYU ScholarsArchive. For more information, please contact scholarsarchive@byu.edu, ellen_amatangelo@byu.edu.
Feature Construction Using Evolution-COnstructed Features
for General Object Recognition

Kirt Dwayne Lillywhite

A dissertation submitted to the faculty of
Brigham Young University
in partial fulfillment of the requirements for the degree of
Doctor of Philosophy

Dah-Jye Lee, Chair
James K Archibald
Bryan S. Morse
Dan A. Ventura
Brent E. Nelson

Department of Electrical and Computer Engineering
Brigham Young University
April 2012

Copyright © 2012 Kirt Dwayne Lillywhite
All Rights Reserved
ABSTRACT

Feature Construction Using Evolution-COnstructed Features for General Object Recognition

Kirt Dwayne Lillywhite

Department of Electrical and Computer Engineering

Doctor of Philosophy

Object recognition is a well studied but extremely challenging field. Human detection is an especially important part of object recognition as it has played a role in machine and human interaction, biometrics, unmanned vehicles, as well as tracking and surveillance. We first present a hardware implementation of the successful Histograms of Oriented Gradients (HOG) method for human detection. The implementation significantly speeds up the method achieving 38 frames a second on VGA video while testing 11,160 sliding windows per frame. The accuracy remains comparable to the CPU implementation. Analysis of the HOG method and other popular object recognition methods led to a novel approach for object detection using a feature construction method called Evolution-COnstructed (ECO) features. Most other approaches rely on human experts to construct features for object recognition. ECO features are automatically constructed by uniquely employing a standard genetic algorithm to discover series of transforms that are highly discriminative. Using ECO features provides several advantages over other object detection algorithms including: no need for a human expert to build feature sets or tune their parameters, ability to generate specialized feature sets for different objects, and no limitations to certain types of image sources. We show in our experiments that ECO features perform better or comparable with state-of-the-art object recognition algorithms making it the first feature construction method to compete with features created by human experts at general object recognition. An analysis is given of ECO features which includes a visualization of ECO features and improvements made to the algorithm.

Keywords: ECO features, object detection, feature construction, genetic algorithm, self-tuned, AdaBoost
ACKNOWLEDGMENTS

I want to thank my Father in Heaven who I turned to when frustrated or in need of ideas. I want to thank Dr. Lee, my advisor, for being a mentor and a friend to me and believed enough in me to be crazy enough to advise so many students at once. My family was a source of strength and support throughout this process. My wife never thought when we got married that I would go to graduate school but she cheered me along the way and put up with our poverty. My lab mates made the experience fun and have always been there to help even when they also had pressing deadlines. Each of us are standing on the shoulders of giants and that becomes clearly apparent when reading the work of so many other researchers. I appreciate those listed in the bibliography and the thousands of others who really could be listed in the bibliography for the work they have done before me.
Table of Contents

List of Tables ix

List of Figures x

1 Introduction 1

2 Background 5

3 Hardware Implementation of Histograms of Oriented Gradients 8
   3.1 Background ................................................. 8
   3.2 Overview of GPU ......................................... 10
   3.3 Histograms of Oriented Gradients ......................... 11
   3.4 Implementing on the GPU ................................ 14
      3.4.1 Split Image into Three Channels ..................... 16
      3.4.2 Gradient Computation ................................. 16
      3.4.3 Compute Gradient Magnitude and Orientation .......... 17
      3.4.4 Create Histograms .................................. 18
      3.4.5 Support Vector Machine Classification .............. 19
   3.5 Results .................................................. 20
   3.6 Discussion ............................................... 21

4 ECO Feature Algorithm 23
List of Tables

3.1 GPU Implementation - Time Budget ................................................. 15
3.2 Summary of GPU Speedup ............................................................... 22
4.1 List of Image Transforms ............................................................... 24
6.1 Number of Images in Caltech Motorbikes Dataset ......................... 42
6.2 Number of Images in Caltech Faces Dataset ................................. 42
6.3 Number of Images in the Caltech Airplanes Dataset .................... 44
6.4 Number of Images in the Caltech Cars Dataset ............................ 44
6.5 Comparison on Caltech Datasets .................................................... 45
6.6 Number of Images in the INRIA Person Dataset ............................ 46
6.7 Comparison to State-of-the-art on INRIA Person Dataset .............. 47
6.8 Confusion Matrix on Volcanoes on Venus .................................. 50
6.9 Five Fold Classification on BYU Fish Dataset ............................ 54
7.1 Comparison to State-of-the-art Methods on the INRIA Person .......... 78
List of Figures

1.1 The Steps of Object Recognition .................................................. 2
1.2 Who Performs What Object Recognition Tasks ................................. 3

3.1 Sliding Window Example ............................................................... 12
3.2 Histograms of Oriented Gradients - Cell Overlap ................................ 12
3.3 Steps in the GPU Implementation ................................................... 14
3.4 Coalescing GPU Reads .................................................................. 17
3.5 Examples from INRIA Person Dataset .............................................. 21
3.6 Compare GPU Accuracy to CPU ..................................................... 21

4.1 Example ECO Features ................................................................. 25
4.2 Example of ECO Feature Subregions ............................................... 26
4.3 Examples of Crossover and Mutation .............................................. 27
4.4 Overview of ECO Feature Algorithm ............................................. 31

5.1 Catching Exceptions .................................................................... 33
5.2 Random Number Generator Code .................................................. 36
5.3 Code to Write ECO Features to File .............................................. 37

6.1 Examples of Caltech Motorbikes ..................................................... 41
6.2 Examples of Caltech Faces ............................................................ 42
6.3 Examples of Caltech Airplanes ...................................................... 43
6.4 Examples of Caltech Cars Dataset ........................................ 44
6.5 Performance on Caltech Datasets ........................................ 46
6.6 Examples from the INRIA Person Dataset ............................. 47
6.7 Performance on INRIA Person Dataset with Various Number of Features ................................. 48
6.8 Examples of Volcanoes on Venus ......................................... 49
6.9 Comparison on Volcanoes on Venus ...................................... 52
6.10 Examples of BYU Fish Dataset ........................................... 53
6.11 Example Images From MNIST Dataset ................................. 55
6.12 Sample of Mistakes on MNIST Dataset ................................. 56
6.13 Example Images From the Daimler Pedestrian Classification Benchmark Dataset ................................. 57
6.14 Performance of ECO Features on the Daimler Pedestrian Classification Benchmark Dataset ................................. 58
7.1 Visualization of ECO Features on Caltech Faces ......................... 61
7.2 Visualization of ECO Features on Caltech Airplanes ..................... 62
7.3 Average Caltech Airplane .................................................. 63
7.4 Visualization of ECO Features on Caltech Motorbikes .................... 64
7.5 Visualization of ECO Features on Caltech Cars .......................... 66
7.6 Visualization of ECO Features on INRIA Person Dataset .................. 68
7.7 Visualization of ECO Features on INRIA Person Dataset .................. 69
7.8 Visualization of ECO Features on INRIA Person Dataset .................. 70
7.9 Visualization of ECO Features on BYU Fish ................................ 71
7.10 Visualization of ECO Features on Volcanoes on Venus ................. 73
7.11 Example Search Space .................................................... 74
7.12 Speciation Diversity ...................................................... 75
7.13 Comparison on INRIA Person With and Without Speciation ............... 76
<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>7.14 Compare Histogram of Oriented Gradients to ECO Features With and Without Speciation</td>
<td>77</td>
</tr>
<tr>
<td>7.15 Accuracy of Adaboost Model Varying ECO Feature Length</td>
<td>79</td>
</tr>
<tr>
<td>7.16 Accuracy Without Subregions</td>
<td>80</td>
</tr>
<tr>
<td>7.17 OpenMP Example Code</td>
<td>82</td>
</tr>
</tbody>
</table>
Chapter 1

Introduction

Object recognition is a very challenging task that many researchers are currently pursuing [1]. The difficulty of detecting and labeling objects in images is due in part to issues such as lighting conditions, object pose and articulation, distortions, high intraclass variation, image sensor noise, wide variability in the types of cues necessary to detect the specific object type, partial occlusions, as well as naturally varying parameters of the objects.

Overcoming these obstacles in order to achieve robust object recognition would be beneficial for many scientific fields and applications which include automotive safety [2], surveillance [3,4], video indexing [5], image classification [6], content-based image retrieval [7], tracking [8], robotics [9], and a host of others. Despite the large number of applications and incentives that would benefit from robust object recognition, object recognition is not yet widely adopted in industry.

One of the main goals of computer vision is to take raw sensor data, the input signal, and create a set of symbols that represents the data. In object recognition these symbols are referred to as features. Machine learning techniques are commonly used to take these features and then classify them as either belonging to the object of interest or not. In general, machine learning algorithms take in symbols, find patterns in the symbols, and use mathematical methods to separate the symbols into classes. Machine learning frees the user from having to identify rules for classification and in general is more accurate at creating rules than human experts are. Machine learning techniques, however, are most successful when the set of features uniquely describes the object of interest. The image processing used to create a higher-level representation of the input signal bridges the semantic gap that exists between the raw input signal and what is needed by the machine learning algorithm. Agarwal and Roth state it this way: “... we suggest that in order to extract high-level,
conceptual information such as the presence of an object in an image, it is essential to transform the raw, low-level input (in this case, the pixel grayscale values) to a higher-level, more ‘meaningful’ representation that can support the detection process” [10].

The various methods that have been used to obtain high quality features can be categorized into three groups: feature selection, feature extraction, and feature construction. The following definitions are taken from Motoda and Liu [11].

**Feature Selection** is a process that chooses a subset of features from the original features so that the feature space is optimally reduced according to a certain criterion.

**Feature Extraction** is a process that extracts a set of new features from the original features through some functional mapping.

**Feature Construction** is a process that discovers missing information about the relationships between features and augments the space of features by inferring or creating additional features.

Figure 1.1: The steps of object recognition with the forms that the data takes during the process.

Figure 1.0 shows where feature selection, extraction, and construction fit into the process of object recognition. Feature construction will generate a set of symbols from the raw data that is obtained from the imaging sensor. Those symbols are enhanced by choosing a subset of the most important symbols through feature selection, or creating new symbols through some functional mapping by performing feature extraction. Then the symbols are classified as belonging to the object or not.
Figure 1.2: Shows a current look at what tasks are performed by whom.

Feature construction is mostly performed by humans currently, while the classification step is almost always performed by computers. Feature extraction and selection on the other hand have been performed by both but with the scales tipping more toward computers. Figure 1.1 displays a current estimated distribution of which object recognition steps are performed by humans versus computers. In the object recognition literature, the pattern of human experts creating features and then using machine learning algorithms for classification is clear [10,12–24].

In the same way that the application of machine learning has improved the accuracy of object recognition, we believe that feature construction can have the same kind of impact on improving the quality of features. And as stated before, higher quality features are more able to uniquely describe the object of interest, producing more accurate object recognition. One major reason for this is the ability of a computer to find patterns in large amounts of data that humans simply cannot do. In the same way that machine learning frees the user from having to generate their own rules for classification, feature construction frees the user from having to generate their own features.
The main focus of this work is to produce a high quality feature construction algorithm capable of general object recognition. Rather than relying on human experts our method, which we call Evolution-COnstructed features [25–27], uses simulated evolution to construct series of transforms that convert the input signal of raw pixels into high quality features. ECO features are the first feature construction method that performs better, or as well as human experts, at general object recognition. Using ECO features also provides many benefits.

1. Good features can be discovered without the use of a human expert.

2. Non-intuitive features can be constructed that are not normally considered by human experts.

3. ECO features are not limited to certain image sources including data originating from CMOS Sensors, synthetic aperture radar (SAR), infrared (IR), and potentially others such as magnetic resonance imaging (MRI), computed tomography (CT), X-ray, etc.

4. ECO features can be learned off-line for any object type. In other systems the human expert creates features that are good for one class of objects but may do poorly on other object types.
Chapter 2

Background

Feature selection is becoming increasingly important as the amount of information captured increases. There is a good deal of literature on feature selection [28–30]. Narendra and Fukunaga use a branch and bound method to avoid an exhaustive search of all feature subsets but still select the best subset according to their criterion [31]. Liu et al. also develop a feature selection method to avoid exhaustive search for a bag-of-visual words applications [32]. Dash and Liu evaluate the effectiveness of various feature selection methods using an inconsistency measure and explore various search strategies for feature selection. Wang et al. use particle swarm optimization to do feature selection in a stochastic way [33]. Pedrycz and Ahmad do feature selection, again in a stochastic manner, using both a genetic algorithm and particle swarm optimization but using structure retention as their criteria [34]. These methods highlight a progression from exhaustive methods, to search based methods that find the best subset possible, to stochastic methods to deal with the size of the feature selection space.

Feature selection is also used for a few other applications. Huang et al. use feature selection to find a subset of features for a support vector machine [35]. Bala et al. use a genetic algorithm and decision trees to select features for visual concepts [36]. Dollár et al. reduce the dimensionality of an initial random set of features using Adaboost [37] in order to detect humans in images.

Feature extraction is most commonly seen as a way to reduce the dimensionality of the feature space by using a function that finds linear combinations of the most important features. Linear discriminant analysis and principle component analysis are very common methods for feature extraction [38–40]. Variations of linear discriminant analysis have also
been done for feature extraction [41–43]. Sherrah et al. use genetic programming to decide whether to do feature selection or extraction [44] as a pre-processing step before classification.

Feature construction methods appear much less frequently in the literature than feature selection and feature extraction methods. Feature construction has been used to improve general machine learning algorithms. Both [45] and [46] use genetic programming to construct features to improve the results of a decision tree. They build trees of operators that are applied to primitive features to construct richer features.

Feature construction has also been applied to object recognition using evolutionary techniques to build trees of mathematical operators and primitive features. Vafaie and De Jong [47] use genetic programming to construct features and then reduce the number and redundancy of their features through feature selection. Their trained features are then used by a decision tree to detect eyes. Brumby et al. [48] apply the same technique to find water in multi-spectral aerial-photography. Roberts and Claridge [49] use pixel statistics as their primitive features and then pull pasta shapes from a noisy background. Bulitko et al. do automatic image interpretation that searches for transforms that allow remote sensing images to be automatically segmented [50]. Krawiec and Bhanu use series of transforms found through evolution to construct features [51–56]. Although somewhat similar to the ECO feature algorithm, in the end they use a set of feature extraction operators (various moments and other pixel statistics) to construct a real-valued feature vector of predetermined length. This technique forces a specific dimensionality and constrains the features to be of a certain class, namely those that can be generated from the set of extraction operators selected by the authors.

None of the feature construction techniques that we could find in the literature, and that are discussed here, performs well at general object detection. Generally they are tested on a single dataset with various constraints. I believe that the real power of a feature construction algorithm is manifest when it is able to automatically generate sufficiently unique features to perform well at general object recognition.

There are many works on general object recognition. Viola and Jones, in a very well known work, developed an algorithm using Adaboost to select basic integral image features and build a cascade of detectors so that background images are quickly classified. Mohan et
al. built a parts based detector with Haar wavelets to represent the image. They then use a support vector machine for classification [18]. Belongie et al. create a shape descriptor that represents shape with a set of discrete points sampled from the contour of the shape and then use K nearest neighbors for classification [13, 57]. Agarwal and Roth use Förstner operator to find representative parts of the target object and create image patches that can then be used to represent the object [10]. While by itself it is not an object recognition algorithm, Lowe developed an image feature he called SIFT, that became the basis for features in many object recognition algorithms [58]. Schneiderman and Kanade built a classifier based on the statistics of localized parts. Each part is described using a subset of wavelet coefficients [19]. Dalal and Triggs created a feature called histograms of oriented gradients that excelled at many object detection tasks [59, 60] and was modeled in part from SIFT features. Many works since then use some form of histograms of oriented gradients [12, 15, 16, 21, 61]. Laptev uses histograms as features, weighted Fisher linear discriminant as a weak classifier, and then AdaBoost for classification [17]. Serre et al. Bileshi, and Siagian and Itti developed biologically inspired features that are based on Gabor filters [14, 62, 63]. Yu et al. use a shape codebook of geometric information among visual words and then use probabilistic voting for classification [24].
Chapter 3

Hardware Implementation of Histograms of Oriented Gradients

Before developing the ECO features algorithm we worked with the Histograms of Oriented Gradients (HOG) algorithm created by Dalal and Triggs [59]. At the time it represented the best method for detecting humans in images and worked well on many other datasets as well. Many of the ideas for ECO features came while reading about the HOG method and working on its implementation. In their paper, Dalal and Triggs explain the various parameters and trials that they conducted to develop the method. It told a story of significant work and the same story could be found in many other object recognition papers. Developing feature sets requires trial and error and most often builds on the work of others. The human expert is limited by time and can only try a small number of hand crafted solutions. Trial and error becomes his mantra with success coming here and there. ECO features were born out of the idea that computers are better suited for such a task.

The HOG method was accurate but it was also slow, making it impossible to use for many applications that required real-time performance. This chapter outlines a GPU implementation of the HOG method.

3.1 Background

Human detection has long been an important part of computer vision. From machine and human interaction, biometrics, unmanned vehicles, to tracking and surveillance, human detection and recognition has played an important role. Accuracy in many of these applications is very important. However, properly identifying humans in live videos is very difficult given the large variations in human pose, color, illumination, and size. Due to the high degree of difficulty, human detection will continue for some time to be an important part of research in computer vision.
There are many previous works dealing with object detection. A few methods concerning human detection using appearance-based methods will be summarized here. An over-complete dictionary of Haar wavelets has been used with a support vector machine to detect humans [64]. Human detection has been done by using human parts detectors and then combining them probabilistically [65–67]. Using parts descriptors helps deal with partial occlusion. Human detection has also been extended to night time using a night vision camera [68]. In order to speed up computation and lower false positive rates, stereo vision is used to identify parts of the image that are too far away and do not need classification [69]. Motion and simple detectors, used as weak classifiers for Adaboost, are used to identify humans and can work on low resolution images with very good results [70].

The Histograms of Oriented Gradients (HOG) method for human detection was presented in [59]. Since the original papers, there are many research projects that have expanded on the work. The method has been extended to infrared images [71]. Paisitkriangkrai et al. made a study of several state-of-the-art human detectors including HOG and make a comparison to their own human detector which has similar accuracy but runs a little faster [72]. In order to speed up the HOG method, a rejection cascade of weak classifiers using variable sized HOGs was used [73].

Real-world applications carry the addition requirement that human detection be done in real time. The difficulty, however, lies in the number of computations that have to be done in order to accurately detect humans. Clock speeds on microprocessors are not increasing and the amount of instruction level parallelism that can be exploited is diminishing as microprocessor developers are faced with very difficult design barriers [74]. Because of this, the industry has turned more to parallel architectures. Parallel architectures are anything but new, but many challenges still remain in making the technology a viable solution for scientific computing.

One exciting development for the scientific and high performance computing communities is the use of a GPU for general purpose computing, denoted as GPGPU. GPUs are becoming capable of performing more than very specific graphics related tasks, broadening their appeal as a capable coprocessor. The GPU has become a commodity item driven by the multi-billion dollar gaming industry. New video games are constantly demanding additional
computational capabilities of these devices with new games adding physics computations to the GPU’s task list. Due to their computational abilities and price, GPUs are becoming a very popular option for scientists to use.

Although GPUs are becoming more general purpose, they differ significantly from general purpose CPUs. The biggest difference is the GPU’s focus on parallel computing. A GPU is designed from the outset to run thousands of threads of execution simultaneously. In order to do this, GPUs dedicate much more silicon to ALUs. CPUs, on the other hand, focus more on larger caches, complicated dynamic branch prediction hardware, etc. Only recently have commercial CPUs started to feature parallel execution through multiple cores. These cores, however, are very different from the lightweight execution cores of the GPU.

The rest of this chapter is organized into six sections. Section 3.2 gives an overview of the GPU and section 3.3 gives an overview of HOG. Section 3.4 describes our implementation on the GPU, giving many of the results of the GPU implementation versus a software implementation. Section 3.5 gives a brief summary of our results. Finally, Section 3.6 gives our conclusion.

3.2 Overview of GPU

The NVIDIA 8800 GTX was used for this work which uses the Compute Unified Device Architecture or CUDA. CUDA’s main advantage is that it exposes the fast shared memory on each multiprocessor and provides fast access to the GPU’s main memory. CUDA allows developers to access the GPU through the C programming language that has been augmented with minimal syntax to help deal with code that runs on the GPU. NVIDIA provides a C compiler for the GPU code. Each function launched from the PC to the GPU is called a kernel and has parameters that specify the number of parallel sections and the number of threads to run in that parallel section. Threads that work together in a parallel section is called a thread block.

Each multiprocessor is a SIMT (single-instruction multiple-threads) unit that creates, schedules and executes threads, each of which executes independently on its own processor and its own register set. Threads within a single multiprocessor can work cooperatively through shared memory. Light weight barriers are used to synchronize thread execution.
Each thread of execution can be referenced first by a 3-D block index and then by a 3-D thread index, though 2-D and 1-D indexing can be used and the unspecified dimensions are assumed to be zero. The number of thread blocks can greatly exceed the number of multiprocessors but must run independently of one another. Multiple blocks can run on the same multiprocessor at the same time if there are enough available resources, namely shared memory and registers. This helps maximize hardware usage when threads stall for events such as global memory loads. The 8800 GTX is capable of running 12,288 concurrent threads given that there are enough resources on the GPU for a particular task.

Programmers have access to several kinds of memories.

- Shared memory located on each multiprocessor, which if accessed properly, can be as fast as register accesses. The memory is banked so that threads can access the memory concurrently.

- Constant memory which is global read-only memory that is cached.

- Texture memory which is global read-only memory that is cached in 2-D and provides fast access to textures. Textures have various addressing modes and data filtering. These properties make it well suited for working with images.

- Local memory which is read and write global memory that is used when local variables cannot fit in registers.

- Global memory which is, obviously, global memory that is also read and write memory for general purpose uses.

### 3.3 Histograms of Oriented Gradients

The HOG method is an appearance-based approach which attempts to capture local shape information in histograms. We chose to use the HOG algorithm because, at the time, it represented the state-of-the-art in human detection.

The HOG method uses a sliding window method, as shown in Figure 3.0. Each window represents a location at which a human can be detected and covers a $64 \times 128$ pixel area.
Figure 3.1: A sliding window example. The first step shows the image with just the first window. Step two shows the first two windows. Step three has the first row of windows plus the first window of the second row. The last step shows all the windows on the image at once, highlighting the amount of overlap there is among windows.

area in the image. In order to make the method more scale-invariant, this same sliding window technique is applied to the image at different scales.

Figure 3.2: Four adjacent blocks, 1 - 4, showing how cells are shared. The cells are labeled with the blocks that contain them.

Each window is made up of 105 blocks — 7 blocks across and 15 down. A block is then made up of four cells — 2 cells across and 2 down. Adjacent blocks share cells as shown
in Figure 3.1. Each cell covers an 8×8 pixel region of the image and blocks cover a 16×16 pixel region. Within each cell a nine bin histogram is made.

Now that it has been shown how the image is broken down into blocks and cells, we will explain how the histograms are computed. First as a pre-processing step, the square root of each pixel value is taken for every channel. Images are three-channel RGB, using eight bits per channel. The image intensity gradients are then found for each color channel in the x and y directions formed using simple 1-D masks, \([-1, 0, 1]\) and \([-1, 0, 1]^T\). Since there are three channels, the channel that has the greatest gradient magnitude, computed from the x and y vector components, is taken as the gradient value for that pixel. The result of this is a single image buffer with a gradient value for each pixel.

After finding the initial gradient magnitudes each block is multiplied by a Gaussian mask that is centered over the block and is the same size as the block in order to give more importance to values at the center of the block. One implication of this is that each cell is shared by multiple blocks but will be multiplied by different values in the Gaussian mask for each block that they belong to. Information is then stored by block rather than by cell and each cell will contribute different values to each block that it is a part of. This effects where parallelism can be exploited.

Once the gradient magnitude values have been calculated, they can then be placed in nine-bin histograms. For each pixel in a cell, the orientation of the x and y gradients determines which of the nine bins in the cell to use. Before assigning to a bin, the orientation is mapped to a value between 0° and 180°, which effectively removes the sign of the orientation. Once the bin is determined, the bin is incremented by the gradient magnitude at that pixel. Once all the cells are created, each block is normalized using an L2 norm. The overlapping of blocks and normalization help deal with variations due to illumination and poor image contrast.

Once all 105 blocks in a window have been processed, the support vector machine descriptor is formed. The descriptor, for each window, is the concatenation of all the histograms in all the blocks – the four histograms in a single block are concatenated together and in turn all the blocks are concatenated together. Since there are 105 blocks with 4 his-
tograms each and 9 bins per histogram, there are 3,780 values in the support vector machine descriptor. The descriptor is then passed to a linear support vector machine for classification.

### 3.4 Implementing on the GPU

First, we implemented our own CPU implementation of the HOG algorithm, despite the fact that there was code available on-line. The primary reason was that the code ran very slowly because of the flexibility needed to test the parameters of the algorithm. Our implementation uses default values that do not need to be modified, allowing our implementation to run considerably faster and give a more reasonable comparison to the GPU implementation. All software runs on an Intel Pentium 4 clocked at 3.4 GHz with one gigabyte of RAM.

We made many optimizations in our software version but only a few that we have time to mention. The biggest optimization we have done was to create all the histograms up front rather than to do them as we slid our window over the image. Because each window has considerable overlap with other adjoining windows, and the consistent stride of eight pixels that we were using, all the histograms could be fashioned up front. Afterwards, each window descriptor can be built by taking the appropriate histograms. This drastically reduced the number of histograms that had to be created, or rather recreated. Also OpenCV was used heavily to provide some optimization. OpenCV is an open source imaging library started by Intel Cooperation. OpenCV’s main goal is to provide basic imaging routines that have been highly optimized for real-time image processing. These two optimizations accounted for a large part of the speedup that were obtained in our software implementation compared to the original implementation.

![Flowchart](Image)

**Figure 3.3:** Steps in the GPU implementation.
The GPU implementation was designed to run on video which has a few implications. First of all, it was assumed that each image in the video sequence was the same size as the first image frame in the video. This allows the data structures to be created in the beginning and reused throughout the video sequence. The data structures on the GPU were made to work with the first image at its largest scale, and then reused for each scale of the image and for each image frame in the video sequence. Creating and destroying data structures for each image size can be very time consuming. By assuming that a video is being used, that uses a consistent image size throughout the video sequence, this time can be saved. For timing purposes throughout this section images are $640 \times 480$ pixels.

When using a GPU one consideration has to be the overhead of memory transfers for the host system to the GPU. In our design, however, not much memory bandwidth is used between the host system and the GPU. The image is transferred to the GPU, which for a $640 \times 480$ image, took 460 microseconds. The GPU computes a classification for every window which is then transferred back to the CPU. The transfer is so small it is hard to measure accurately. The overall overhead of using the GPU is small, especially considering that the CPU can continue working, if needs be, as the GPU processes.

The goal of the GPU implementation was to reach 30 frames per second or 33 milliseconds per frame. So each piece of the CPU implementation was used as a gauge of how much of the time budget should be allocated to that piece. Table 3.0 summarizes the budget allocations and consumption of each piece.

The following subsections give an overview of how parts of the algorithm were implemented on the GPU. These steps are shown in Figure 3.2.

<table>
<thead>
<tr>
<th>Operation</th>
<th>Budget Alloc.</th>
<th>Used</th>
</tr>
</thead>
<tbody>
<tr>
<td>Split Image</td>
<td>1.48%</td>
<td>1.84%</td>
</tr>
<tr>
<td>Scale, Sqrt, and Gradient</td>
<td>33.6%</td>
<td>7.45%</td>
</tr>
<tr>
<td>Get Mag. and Orientation</td>
<td>14.4%</td>
<td>2.8%</td>
</tr>
<tr>
<td>Create Histograms</td>
<td>39.7%</td>
<td>15.6%</td>
</tr>
<tr>
<td>SVM Classification</td>
<td>10.8%</td>
<td>41.8%</td>
</tr>
<tr>
<td>Total Time Per Frame</td>
<td>100%</td>
<td>78.8%</td>
</tr>
</tbody>
</table>

**Table 3.1:** Summary of time budget for each piece of the GPU implementation.
3.4.1 Split Image into Three Channels

To calculate the gradients in our HOGs the maximum gradient over the three color image channels is needed which requires the image to be split into three channels. On the CPU, splitting an image took six milliseconds and 20 milliseconds aggregated for all scales. Splitting the image alone took 60% of our time budget on the CPU. GPUs are mostly known for their computational abilities but they also have a high memory bandwidth which is evident when splitting the image into three channels. To split the image into three channels, the GPU takes 0.607 milliseconds, giving a $9 \times$ speedup. For reasons explained later, there is no reason to split the image at other scales and so the aggregated time for splitting the image into three channels over the different scales is still 0.607 milliseconds and gives an aggregated speedup of $33 \times$. Rather than eating up 60% of our time budget this function then uses only 1.8% of the time budget, which was more than adequate.

To get good performance, it is important that the GPU accesses memory efficiently because each global memory access incurs 400 - 600 clock cycles. The GPU allows for memory accesses to be coalesced, meaning that certain memory accesses can be combined to reduced the total number of memory transactions that have to be made. The rules to coalescing are different for the different GPUs. With the GPU used in our project the memory transactions could not be coalesced because each thread needed to access consecutive 32 bit words. The original image uses eight bits per channel, giving 24 bits to store a single pixel. Figure 3.3 illustrates why the memory transactions could not be coalesced. The texture memory is optimized for caching 2-D data structures allowing an image in global memory to be accessed more flexibly. There are still access patterns that could cause thrashing in the cache but the cache is going to be forgiving of alignment requirements.

3.4.2 Gradient Computation

The biggest benefit to using the GPU comes when creating a pyramid of images in order to make human detection scale invariant. On the CPU, resizing takes approximately 60 milliseconds for each image. This uses our entire time budget and more. In our GPU implementation there is no implicit image resizing. When the image is split into three channels, each channel is stored to a texture memory location. Textures can be indexed
using floating point values and have several filtering modes, including a bilinear interpolation filter. When the x and y gradients are being created they access the textures according to the current scale, using a bilinear filter, effectively resizing the image. There is no penalty for filtering and thus image resizing does not use any of our time budget.

Another step that was thrown into the gradient kernel is the image square root that is done to the original image. Rather than do the actual square root, a LUT is used to look-up the square root value for each pixel and each channel. Since there are only 256 floating point values in constant memory, the entire LUT can fit in the constant memory cache allowing fast access.

The kernel uses a large number of registers, 31 registers per thread, which decreases the number of threads and blocks that can run on a multiprocessor concurrently. The kernel, however, does run very fast compared to the CPU implementation. The CPU implementation takes 60 milliseconds to scale the images, 8 milliseconds to do the image square root, and 385 milliseconds to compute the gradients. All this is done on the GPU in 2.46 milliseconds giving a speedup of 180× and coming in at 13% of our time budget.

3.4.3 Compute Gradient Magnitude and Orientation

The previous step creates gradient values for each of the three channels and in this step, the max gradient across channels is found and the orientation computed. Since the
kernel only uses nine registers and very little shared memory, it can have a high thread count and have multiple thread blocks run at once. The kernel also has a high computational complexity. For every pixel there are three global memory reads, three square roots, six multiplications, three additions, two short branch length conditional statements, one arctangent, one floating point modulus operation, and two global writes. Since every pixel is independent from all other pixels, there is a very high amount of parallelism that can be exploited. The GPU supports branch predication which can be used here because of the short length of the two conditional branches. This keeps the parallel threads from diverging from one another. If the threads were allowed to diverge then not all the threads would be executing the same code and would defeat the purpose of a SIMT processor. All the pixels being independent also allow all the reads and writes to be coalesced.

The GPU gives a large speedup when compared with the CPU implementation. The CPU implementation takes 200 milliseconds to compute the gradient magnitude and orientation. The GPU takes .938 milliseconds which is greater than a 200× speedup, and accounts for only 2.8% of our time budget. The high computational complexity and high memory bandwidth needed, coupled with the high amount of parallelism maps the function very well to the GPU.

3.4.4 Create Histograms

After the gradient magnitude and orientation have been found, the histograms can be generated. Each thread block operates on one histogram. There are 36 values in the histogram whose access pattern is random, owing to the fact that access depends on the orientation of gradient. There is no hardware support for atomic adds to a floating point value in shared memory for any NVIDIA GPUs at the time of this work. A software trick would be possible if the GPU supported double precision values, like that done in the histogram256 example [75] in the NVIDIA CUDA SDK, but our GPU did not have that ability. Multiple threads cannot write to the same shared memory address which limits the number of threads that could be used in this kernel. This kernel uses a 4×4 thread block resulting in only 16 total threads per block. Each thread keeps its own histogram so that writes to shared memory do not conflict. The resulting histograms for each thread are then combined when
they are done. The 16 threads first load the gradient magnitude and orientation values into shared memory for the current histogram, which results in eight loads per thread. The kernel uses 2,592 bytes of shared memory per thread block, as reported by the compiler and uses 21 registers per thread. As the magnitude values are read in, they are multiplied by a Gaussian mask that is stored in constant memory. There are 256 Gaussian values in a 16×16 mask, which easily fit in the constant memory cache. The histograms are formed as explained in Section 3.3. After the histogram is made, it is then normalized using L2 normalization. As a result of adding the histograms from multiple threads together, the histograms are scattered around shared memory. Before writing out the histograms to global memory, they are rearranged in shared memory so the writes can be coalesced.

Despite the low thread count in a thread block the kernel offers a considerable speedup over the CPU implementation. The CPU implementation takes 540 milliseconds to create the histograms whereas the GPU takes 5.17 milliseconds. The approximate speedup is 100× and consumes 16% of the time budget.

3.4.5 Support Vector Machine Classification

In order to train the support vector machine, the INRIA Person Dataset [76] was used. A portion of it was used during the training process and another portion during testing. Initially 8,000 negative examples and 2,416 positive examples were used for training. Choosing the negative examples is important since the non-human class is much broader and more complex. So the training includes a bootstrapping step [77] which uses false positives from the previous training step as examples to continue training with, giving a total of 16,743 total negative examples.

The svmlight [78] support vector machine library was used in the training process. A linear kernel was used in order to make the classification fast and results in an inner product of the window descriptor and the linear weights that were found during the training process. To do the classification one thread block was used for each window, allowing each window to be done in parallel. The histograms for the window are staged in shared memory a piece at a time. Each thread works on a piece of what was staged in shared memory, taking histogram values and multiplying by the linear support vector weights, and then storing its
part of the result to its own shared memory location. Each thread has its own write location to avoid race conditions, creating an array of partial results. After each thread finishes its part of the result, parallel array reduction is used to sum the array and obtain the final classification.

Since there are 36 values per histogram it was impossible to meet the alignment requirements of the GPU we were using in order to coalesce the reads. There are a large number of reads because of the large overlap of windows leading to poor performance. To remedy this, eight bins were used for each histogram in the cells rather than nine giving 32 total bins (eight bins \times four cells) per block and allowed the alignment requirements to be met for coalescing. Using eight bins does reduce the accuracy of the algorithm some but allowed the code to run quickly.

The CPU implementation takes 146 milliseconds for classification per image and the GPU using nine bins takes 186 milliseconds and 14 milliseconds for eight bins. The GPU implementation has approximately a 10\times speedup over the CPU implementation. Coalescing the reads on the GPU accounted for a 13\times speedup. Classification then accounts for 42\% of our time budget.

3.5 Results

The GPU provided excellent performance for human detection compared to an Intel Pentium 4 at 3.4 GHz. As Table 3.1 shows, each kernel provided one or two orders of magnitude speedup. Overall the human detection runs at 38 frames per second on 640\times480 images, testing 11,160 windows per image.

Our CPU implementation did help make reasonable comparisons for the GPU implementation. The original CPU implementation for the HOG method took 13.7 seconds per 640\times480 image. In comparison, the GPU implementation is 527 times faster. The 50\times speedup over our own optimized CPU implementation shows more accurately the processing power that the GPU provides.

Figure 3.5 shows the accuracy of the GPU implementation compared to the original binary produced by the authors of the HOG method. Accuracy is fairly comparable, with the
Figure 3.5: A sampling of a few true positives and false positives made by the algorithm. This highlights the poses, lighting conditions, colors, etc. that humans can take on. False positives tend, but are certainly not limited to, to be man made objects.

Figure 3.6: Comparison of GPU implementation’s accuracy to the original work’s accuracy on the INRIA dataset.

GPU implementation showing a slight advantage when both use eight bins for the histograms and a slight disadvantage when the original uses nine bins per histogram.

3.6 Discussion

The GPU had several properties that made it well suited for image processing. Textures on the GPU provide many image processing capabilities, the memory bandwidth is high, and computations can processed quickly. Overall the GPU shows that it is a capable coprocessor and that many scientific application can benefit from using it.
Table 3.2: Summary of speedup on various code pieces

<table>
<thead>
<tr>
<th>Operation</th>
<th>CPU Time (µs)</th>
<th>GPU Time (µs)</th>
<th>Approximate Speedup</th>
</tr>
</thead>
<tbody>
<tr>
<td>Split Image</td>
<td>20,000</td>
<td>607</td>
<td>30×</td>
</tr>
<tr>
<td>Scale, Sqrt, and Compute Gradient</td>
<td>453,000</td>
<td>2,460</td>
<td>180×</td>
</tr>
<tr>
<td>Get Magnitude and Orientation</td>
<td>200,000</td>
<td>938</td>
<td>200×</td>
</tr>
<tr>
<td>Create Histograms</td>
<td>540,000</td>
<td>5,170</td>
<td>100×</td>
</tr>
<tr>
<td>SVM Classification</td>
<td>146,000</td>
<td>13,800</td>
<td>10×</td>
</tr>
<tr>
<td>Total Time Per Frame</td>
<td>1,350,000</td>
<td>26,000</td>
<td>50×</td>
</tr>
</tbody>
</table>

Programming for the GPU using CUDA forces one to think differently about how they code; however, the new paradigm works well and is easy to learn. The hardware is flexible enough that the user does not have to think about alignment requirements, etc., but there are a large number of optimizations that can be made to achieve greater efficiency. Small modifications can easily lend to a speedup of 2× or more. To make optimizations, the developer must look at the kinds of memory that are being used, access patterns to memory, register usage, number of threads per block, number of blocks that can run simultaneously on a multiprocessor, how threads can work cooperatively, and how to avoid costly branches. While this amounts to quite a few things the developer must keep in mind, the performance advantages can be huge and after time become natural to the developer.

Using a time budget was helpful when programming on the GPU especially to gauge when to stop possibly endless optimizations attempts. It also gave a clear idea about the parts of the program that could use the most work and where optimization could potentially give large dividends.

The HOG method is a very well designed feature set and since its introduction has been the basis for many other feature sets. However it is a good example of the drawbacks of features generated by human experts. Developing feature sets requires trial and error and most often builds on the work of others. The human expert is limited by time and can only try a small number of hand crafted solutions. Understanding the HOG method and implementing it was great motivation to move into automated feature construction methods for object recognition.
Chapter 4

ECO Feature Algorithm

Much of what is done in object recognition aims to achieve what the human vision system is capable of. This tends to cause many to model systems and algorithms after various aspects of the human vision system. This work, to some extent, has also been patterned after human biological structures. The visual cortex of the human brain is made up of hierarchical layers of neurons with feedforward and feedback connections that grow and evolve from birth as the vision system develops. These connections throughout the visual cortex layers make up stages of human vision processing. While later stages tend to contain most of the feedback paths, connections from early stages of the hierarchy are mainly feedforward. These early stages of processing in the visual cortex are sensitive to visual stimuli such as lines of certain lengths and angles [79], spatial frequencies, and colors [80]. Serre et al. [81] suggest that basic image transforms such as derivative-of-Gaussian and Gabor filters model these early processing stages of the visual cortex. We too focus on modeling the early stages of processing by implementing basic image transforms connected in a feedforward manner. In a similar way that feedforward neural connections between these visual cortex layers are created and evolved in humans, our genetic algorithm evolves parameters of transforms and the feedforward processing paths they make up. These processing paths of transforms become the features in our algorithm that describe objects.

4.1 What is an ECO Feature?

ECO features are constructed using a genetic algorithm which creates an ordering of basic image transforms. The set of transforms that are available to the genetic algorithm are shown in Table 4.0. According to the definition of an ECO feature, Equation 4.1, the ECO feature output vector, $V$, is created by applying each of $n$ transforms to a subregion,
$I(x_1, y_1, x_2, y_2)$, of an input image, $I$. Each transform, $T_i$, of the series is applied to the output of the previous transform, $V_{i-1}$, using the transform parameters in vector $\phi_i$, which are also set by the genetic algorithm. Figure 4.0 shows graphical examples of two ECO features.

**Table 4.1:** A list of image transforms available to compose ECO features with the number of parameters the genetic algorithm must set for each transform.

| Image Transform      | $|\phi|$ | Image Transform          | $|\phi|$ |
|----------------------|--------|--------------------------|--------|
| Gabor filter         | 6      | Sobel operator           | 4      |
| Gradient             | 1      | Difference of Gaussians   | 2      |
| Square Root          | 0      | Morphological Erode      | 1      |
| Gaussian Blur        | 1      | Adaptive Thresholding    | 3      |
| Histogram            | 1      | Hough Lines              | 2      |
| Hough Circles        | 2      | Fourier Transform        | 1      |
| Normalize            | 3      | Histogram Equalization   | 0      |
| Convert              | 0      | Laplacian Edge           | 1      |
| Median Blur          | 1      | Distance Transform       | 2      |
| Integral Image       | 1      | Morphological Dilate     | 1      |
| Canny Edge           | 4      | Harris Corner Strength   | 3      |
| Rank Transform       | 0      | Census Transform         | 0      |
| Resize               | 1      | Pixel Statistics         | 2      |
| Log                  | 0      |                          |        |

\[ V = T_n(V_{n-1}, \phi_n), \tag{4.1} \]
\[ V_{n-1} = T_{n-1}(V_{n-2}, \phi_{n-1}), \]
\[ \vdots \]
\[ V_1 = T_1(I(x_1, y_1, x_2, y_2), \phi_1). \]

In addition to generating a sequence of transforms and their operating parameters, the genetic algorithm also selects which portion of the image the transforms will operate on. Any subset of the image may be selected as the input to the transform series, $I(x_1, y_1, x_2, y_2)$, from the whole image down to a subregion as small as a single pixel. Examples of subregions
Figure 4.1: Two example ECO features. The first example shows an ECO feature where the transforms are applied to the subregion where $x_1 = 12$, $y_1 = 25$, $x_2 = 34$, and $y_2 = 90$ from Equation 4.1. The values below the transforms are the parameter vectors $\phi_i$, also from Equation 4.1.

are shown in Figure 4.1, which shows how the selection of the subregion causes ECO features to specializes at an aspect of the target object. Rather than making any assumptions about what the salient regions of the image are, and defining a criteria for their selection, the genetic algorithm is used to automatically search for the subregion parameters $x_1$, $y_1$, $x_2$, $y_2$. In this way the saliency of a subregion is not determined by the subregion itself, but in its ability, after being operated on by the transforms, to help classify objects. The use of subregions allows each ECO feature to specialize at identifying different aspects of the target object.

4.2 Constructing ECO Features

ECO features are constructed using a standard genetic algorithm (GA) [82]. GAs, in general, are used for optimization and searching large spaces efficiently. They start with a population of creatures, representing possible solutions, which then undergo simulated evolution. Each creature is made up of genes which are the parameters of that particular solution. A fitness score, which is designed specifically for the problem, is computed for each creature and indicates how good the solution is. At each generation, creatures are probabilistically selected from the population to continue on to the next generation. Creatures with higher fitness scores are more likely to be selected. Other creatures are made through crossover, which combines the genes of two creatures to form one. Finally the genes of each creature in the population can possibly be mutated according to a mutation rate, which effectively
creates a slightly different solution. The algorithm then ends at some predefined number of generations or when some criteria is satisfied.

In our algorithm, GA creatures represent ECO features. Genes are the elements of an ECO feature which include the subregion \((x_1, y_1, x_2, y_2)\), the transforms \((T_1, T_2, ..., T_n)\), and the parameters for each transform \(\phi_i\). The number of genes that make up a creature is not of fixed length since the number of transforms can vary and each transform has a different number of parameters. Initially, the genetic algorithm randomly generates a population of ECO features and verifies that each ECO feature consists of a valid ordering of transforms.

In order to assign a fitness score to each ECO feature a weak classifier is associated with it. A single perceptron is used as the weak classifier as defined in Equation 4.2. The perceptron maps the ECO feature input vector \(V\) to a binary classification, \(\alpha\), through a weight vector \(W\) and a bias term \(b\).

\[
\alpha = \begin{cases} 
1 & \text{if } W \cdot V + b > 0 \\
0 & \text{else}.
\end{cases}
\] (4.2)

Training the perceptron generates the weight vector \(W\) according to Equation 4.3. Training images are processed according to Equation 4.1 and the output vector \(V\) is the input to the perceptron. The error, \(\delta\), is found by subtracting the perceptron output, \(\alpha\), from the actual
image classification \( \beta \). The perceptron weights are updated according to this error and a learning rate \( \lambda \).

\[
\delta = \beta - \alpha, \quad (4.3)
\]

\[
W[i] = W[i] + \lambda \cdot \delta \cdot V[i].
\]

A fitness score, \( s \), is computed using Equation 4.4, which reflects how well the perceptron classifies a holding set. In Equation 4.4, \( p \) is a penalty, \( t_p \) is the number of true positives, \( f_n \) is the number of false negatives, \( t_n \) is the number of true negatives, and \( f_p \) is the number of false positives. The fitness score is an integer in the range \([0, 1000]\).

\[
s = \frac{t_p \cdot 500}{f_n + t_p} + \frac{t_n \cdot 500}{p \cdot f_p + t_n}. \quad (4.4)
\]

**Figure 4.3:** Examples of crossover and mutation.

Equation 4.4 has two nice properties. Unlike classification accuracy, this fitness is not sensitive to unbalanced numbers of negative and positive training examples. For instance, if there are far more negative examples in the training set, a fitness score based on classification
accuracy would favor a weak classifier that classifies everything as negative, although it has no ability to discriminate positive examples from negative examples. Equation 4.4 also allows us to add a penalty $p$ to false positives in order to bias the weak classifiers towards those with low false positive rates. Whenever a creature’s fitness score is over a threshold it is added to a pool from which AdaBoost draws candidates.

After a fitness score has been obtained for every creature, a portion of the population is selected to continue to the next generation. A tournament selection method is used to select which creatures move to the next generation. A tournament selector takes $N$ creatures at random and the creature with the best fitness score continues to the next generation. By adjusting $N$, the ability for creatures with lower fitness scores to move to the next generation can be tuned. Higher values of $N$ prohibit creatures with low fitness scores to move on to the next generation. Currently $N$ is set to 2 which allows weaker creatures to move on. After selection has taken place, the rest of the population is composed of new creatures created through crossover as shown in Figure 4.2. Through the process of crossover it is possible for ECO features to have a transform length, $n$, longer than 8 which is the cap placed on gene length when they are created. Once the next generation is filled, each of the parameters in the creatures can be mutated, also shown in Figure 4.2. This whole process of finding features is summarized in Algorithm 1.

### 4.3 Training AdaBoost

After the genetic algorithm has found good ECO features, Adaboost [83] is used to combine the weak classifiers to make a stronger classifier. Algorithm 2 outlines how AdaBoost is trained. $X$ represents the maximum number of weak classifiers allowed in the final model, which in our case is 400. $X$ is set to 400 because on all of the datasets that were classified, none of them produced better results using more than 100 weak classifiers and the assumption was made that 400 would be sufficient for future datasets. The number of weak classifiers in the model can be reduced after training if desired. The normalization factor in Algorithm 2 is set so that the sum of the error over all the training images is 1. After training, the resulting AdaBoost model consists of a list of perceptrons and coefficients that indicate how much to trust each perceptron. The coefficient for each perceptron, $\rho$, is calculated...
Algorithm 1 FindingFeatures

for Size of population do
    Randomly create ECO feature. Select $x_1, y_1, x_2, y_2, T_1(\phi_1), \ldots, T_n(\phi_n)$
end for

for number of generations do
    for every ECO feature do
        for every training image do
            Process image with feature transformations
            Train ECO feature’s perceptron
        end for
        for every holding set image do
            Process image with feature transformations
            Track perceptron’s error
        end for
        Assign fitness score to the ECO feature
        Save ECO feature if fitness score > threshold
    end for
    Select ECO features that make it to next generation
    Create new ECO features using crossover
    Apply mutations to the population
end for

using Equation 4.5 where $\delta_w$ is the error of the perceptron over the training images. See Algorithm 2 for more details about how the error is calculated.

When ECO features are created they are not allowed to have more than 8 transforms, but through crossover it is possible for an ECO feature to grow longer. In all of the experiments, ECO features consisting of more than 8 transforms never performed well enough to be selected by Adaboost.

$$\rho = \frac{1}{2} \cdot ln \frac{1 - \delta_w}{\delta_w}. \quad (4.5)$$

4.4 Using the AdaBoost Model

Figure 4.3 shows an example of classifying an image with an Adaboost model containing three ECO features. The figure shows each feature operating on its own subregion of the image (see Equation 4.1). As can be seen, it is possible for the subregions of different features to overlap. Also, as the subregions pass through the transforms, the intermediate
Algorithm 2 \textit{TrainAdaBoost}

\begin{algorithmic}
\State Set of training images $M$
\For {every training image, $m$} 
\State Initialize $\delta_M[m] = 1/|M|$ 
\EndFor 
\For {$x = 0$ to $X$} 
\For {every perceptron, $w$,} 
\For {every training image, $m$} 
\If {wrongly classified} 
\State $\delta_w^+ = \delta_M[m]$ 
\EndIf 
\EndFor 
\EndFor 
\State Select perceptron with minimum error, $\Omega$
\If {$\delta_\omega[\Omega] \geq 50\%$} 
\State BREAK 
\EndIf 
\Calculate coefficient of perceptron using Equation 4.5
\For {every training image, $m$} 
\State $c = \begin{cases} 
1 & \text{if classified correctly by } \Omega \\
-1 & \text{else}
\end{cases}$
\State $\delta_M[m] = \frac{\delta_M[m]e^{-\rho \cdot c}}{\text{Normalization Factor}}$
\EndFor 
\EndFor 
\end{algorithmic}

results may vary in size from one to the next. Each feature is accompanied by its trained perceptron. The output of each perceptron is combined according to Equation 4.6 where $X$ is the number of perceptrons in the Adaboost model, $\rho_x$ is the coefficient for the perceptron $x$ (see Equation 4.5), $\alpha_x$ is the output of perceptron $x$ (see Equation 4.2), $\tau$ is a threshold value, and $c$ is the final classification given by the Adaboost model. The threshold $\tau$ can be changed to vary the tradeoff between false positives and false negatives.

$$c = \begin{cases} 
1 & \text{if } \sum_{x=1}^{X} \rho_x \cdot \alpha_x > \tau \\
0 & \text{else.}
\end{cases} \quad (4.6)$$
Figure 4.4: ECO features and their corresponding perceptrons are combined using Adaboost to classify an image as object or non-object.
Chapter 5

ECO Feature Implementation

5.1 ECO Feature Code Testing

In the ECO feature algorithm, nothing is more constant than change. The genetic algorithm randomly selects transforms, and parameters of those transforms, when creating ECO features, randomly changes those parameters during the mutation phase, creates new ECO features through cross-over, randomly selects parents for cross-over, and in general had a huge number of possible combinations of operating parameters. To ensure stability and robustness it was important to very thoroughly test several aspects of ECO features.

5.1.1 Testing the Validity of an ECO Feature

One problem that can occur when creating ECO features randomly is that some transforms expect their input to be of a certain type and/or depth and other transform’s output would not be compatible. A convert transform was added to ease some of those restrictions, but this only helped if it was selected by the genetic algorithm to be between two incompatible transforms. One possible method to fix this problem would be to explicitly dictate what transform could follow which transforms but this is difficult and fragile. Since OpenCV was used as the image processing library, some features of OpenCV were exploited to solve this problem. Whenever OpenCV detects that some parameter options will not work it raises an exception. Figure 5.0 shows how the OpenCV function \texttt{filter2D} was used to check if it worked successfully. When creating an ECO feature, it is created at random and then used to process an example image. If an exception is caught while processing any of the transforms then the parameter \textit{worked} is set to false, as shown in Figure 5.0. The ECO feature is discarded and a new random ECO feature is created and again tested. This
method of creating ECO features has a bias toward fewer transforms in an ECO features, as
the longer more complicated ECO features are more likely to cause an exception.

ECO features not only have to be tested at creation but also during the crossover
stage of the genetic algorithm. During crossover, however, there is a small possibility that
a successful crossover cannot be made. To account for this, crossover is attempted a fixed
number of times, with the parent ECO features being chosen at random. If after a fixed
number of times a successful crossover cannot be made, an ECO feature is made from scratch.
Creating an ECO feature from scratch during crossover is a rare event, and the capability
was added only to ensure that an infinite loop does not occur.

Mat Gabor_gene::process(Mat& input, bool& worked) {
    worked = true;
    Mat dest;
    try {
        filter2D(input, dest, -1, gabor_kernel, Point(-1,-1), 0,
                 BORDER_REPLICATE);
    }
    catch (Exception& e) {
        worked = false;
    }
    return dest;
}

Figure 5.1: Code from the Gabor filter transform that highlights catching exceptions when
calling OpenCV code.

5.1.2 Transform Parameters

As new transforms were added to the ECO feature algorithm the parameters of new
transforms had to be tested. The parameters of any transform are set by the genetic algo-
rithm and can possibly be modified during the mutation phase. It was assumed that any
change to the parameters would not cause any exceptions to occur. It was possible to check
after every mutation that the ECO feature could still be processed without causing a run-
time error but it was less time consuming and simpler to make sure that a change to the
parameters would not cause an exception. An example of a parameter change that could
cause a run-time error is when the image depth of the destination image is changed. The
image depth is the type, such as 8/16/32 bit integer value or a single or double precision
floating point value. Some transforms can only process certain image depths, so if the output
of one transform changes it can cause future transforms to cause an exception.

Before adding a transform to the ECO feature algorithm, a small program was made
to test the parameters. A function was created that would modify the parameters and,
if it was found that a certain range or combinations of parameters would not work, then
the function was modified to fix the problem. Once it was determined that no random
combination of parameters would cause any run-time errors, the transform, with its function
for modifying parameters, was added to the algorithm. The same function was used when
creating the transform and when mutating the transform.

Another purpose of the separate program for testing individual transforms was to
find ranges for parameters that not only did not cause runtime errors later, but also resulted
in interesting transforms. With the search space already being so large it was undesirable to
let the parameters of transforms be values that did nothing. There are not many restrictions
on values that parameters could take on, but if it was clear that certain values or ranges of
values would not be useful they were removed as possible values.

5.2 Using the Supercomputer

Without the use of the supercomputer on the campus of Brigham Young University
this research could not have been done. The supercomputer was a vital tool to its success.
While a normal computer is sufficient to train on a single dataset once, the supercomputer
allowed simultaneous testing on multiple datasets and to more quickly test changes and
additions to the algorithm. At the current time the supercomputer has two large systems,
marylou5 and m6, that handle most of the computational load, and a few other small systems
to handle jobs that require specific hardware such as large amounts of memory or a GPU.
The two large system capabilities are given below.
**marylou5** is composed of 320 compute nodes with two Intel Nehalem quad-core processors and at least 24 gigabytes of memory per node. Benchmarks put the system capabilities at 26.76 TFlops.

**m6** is composed of 512 compute nodes with two Intel Westmere hex-core processors and 24 gigabytes of memory per node. Benchmarks put the system capabilities at 42.83 TFlops.

The program for finding ECO features mapped well to the requirements and restrictions of the supercomputer. The program capitalized on the multiple processors and cores on each node of the supercomputer but ran several jobs in parallel with relatively short run times. Another approach would have been to use the Message Passing Interface (MPI) to create a program that ran over several computers with a longer run time. The shorter jobs run in parallel and, although it requires reloading images from the dataset, work more efficiently with the scheduler of the supercomputer since it is easier to schedule short jobs. This also avoids the more complicated and more error prone approach using MPI. No communication is needed between the parallel executions of the program since each finds ECO features independently from each other.

Early on in the project, each image was read from disk each time that it was used, which is normal if images are used only once. The disk access on the supercomputer is very slow and the supercomputers administrators noticed right away that the ECO feature algorithm caused a lot of disk bandwidth on the supercomputer. Images in the ECO feature algorithm have to be constantly revisited and this caused a lot of disk accesses. To overcome this problem, images were loaded into memory once at start time, and this greatly increased the speed of the algorithm. Even on the largest datasets that were used, with thousands of images, the memory requirements were modest (< 200 megabytes) since the images are small.

Using the supercomputer also allowed a bug related to the random number generator to be found. The random numbers were being generated as shown in Figure 5.1 which is fairly typical in C/C++. The problem is the seed being used for srand is determined by the clock that only has a one second resolution. When starting multiple jobs at once on
the supercomputer, multiple instances of the ECO features program would have the same exact seed and thus would generate the exact same results. The ECO features program uses several parts of the BOOST libraries\(^1\) which also provides several random number generators and distributions. The BOOST libraries provide generators and distribution that allow more fine grained control and may be used to for demanding numerics and security domains. We added a Merseen Twister [84] pseudo-random number generator with a uniform distribution.

```c
srand(time(NULL));
int i = rand();
float f = drand48();
```

**Figure 5.2:** Standard C Code to generate random numbers.

### 5.3 Reading and Writing ECO Features

ECO features are written out to file in binary format to save space. Figure 5.2 shows the functions for reading and writing the Gabor filter transform to file. Each transform is an object and all transforms inherit from the same parent class. Each transform has a serialize virtual function with an output file stream as the only parameter. Each transform writes out what transform it is and its parameters. After the transforms are written to file the perceptron is also written to file. The perceptron stores the number of weights, or inputs, to the perceptron and the value of those weights. When loading an ECO feature, the type of transform is read, followed by the appropriate function called to load the transform and return a transform object. The ECO feature then stores a list of the transform objects. The ECO feature then loads the perceptron, also as an object, and stores it alongside the list of transforms.

\(^1\)http://www.boost.org/
void Gabor_gene::serialize(ofstream & os) {
    // write out which transform this is
    os.write(reinterpret_cast<char *>(&gene_type), sizeof(int));
    // Write out parameters
    os.write(reinterpret_cast<char *>(&sigma), sizeof(float));
    os.write(reinterpret_cast<char *>(&theta), sizeof(float));
    os.write(reinterpret_cast<char *>(&lambda), sizeof(float));
    os.write(reinterpret_cast<char *>(&psi), sizeof(float));
    os.write(reinterpret_cast<char *>(&gamma), sizeof(float));
    os.write(reinterpret_cast<char *>(&kernel_size), sizeof(int));
}

Gabor_gene* Gabor_gene::load_from_file(ifstream & fin) {
    float tmp_sigma, tmp_theta, tmp_lambda, tmp_psi, tmp_gamma;
    unsigned int tmp_kernel_size;
    fin.read(reinterpret_cast<char*>(&tmp_sigma), sizeof(float));
    fin.read(reinterpret_cast<char*>(&tmp_theta), sizeof(float));
    fin.read(reinterpret_cast<char*>(&tmp_lambda), sizeof(float));
    fin.read(reinterpret_cast<char*>(&tmp_psi), sizeof(float));
    fin.read(reinterpret_cast<char*>(&tmp_gamma), sizeof(float));
    fin.read(reinterpret_cast<char*>(&tmp_kernel_size), sizeof(int));
    return new Gabor_gene(tmp_sigma, tmp_theta, tmp_lambda, tmp_psi,
        tmp_gamma, tmp_kernel_size);
}

Figure 5.3: Code to write and read in ECO features to file.

5.4 Handling Mutations

When the genetic algorithm ends a generation, it selects ECO features to go on to
the next generation. Some of those ECO features will be later mutated but most will not
be. Those that have not been mutated have no need to have their fitness score reevaluated.
Storing whether an ECO feature has been mutated helps speed up each generation after the
first.

If a mutation occurs the perceptron associated with the ECO feature is discarded and
a new one created where weights have been zeroed out. This is done not only because the
ECO feature could be drastically different because of the mutation but because the size of
the output of the ECO feature could change, which forces the perceptron to have additional
inputs. Perceptrons are also discarded and recreated at crossover.
5.5 Full Image Detection

All of the datasets that were used in the work consist of image crops of the target object. When using a sliding window approach this is acceptable because only a single window will be tested at once. Figure 3.0 shows how a sliding window approach works. The window, with an approximate aspect ratio of the target object, is moved over the image, a fixed step size each move. Generally a large set of negative examples is used to represent the background of the target image as if full image detection is being done. It is larger than the positive training examples because it represents a broader image class and because full images typically are dominated, area wise, by the background. By training over a large number of negative crops with the positive crops, training is effectively done as if on full images where everything but the target object is labeled as being background.

In addition to sliding the window over the image, generally the image is also scaled and then reprocessed. In this way detection is scale invariant. As was mentioned, the window is approximately the aspect ratio of the target object and by scaling the image the target object can be detected at different scales. The number of times the image needs to be scaled depends on the range in scales that the target object is expected to be in.

After a sliding window approach, especially where the image is also being scaled, there can be a number of detections for the same object in the image. A non-maximal suppression method is needed to attempt to reduce detections to one per object. Because of the information that is gained by detections in the same spatial regions, it is possible that accuracy on the full image is better than by treating all the windows independently.

With the number of windows that have to be tested on a single image, which is apparent in Figure 3.0, and then again for each scale, processing time on full images is very time consuming. Using 640 × 480 images, a sliding window step of eight pixels in both the x and the y direction, and eight different scales, the ECO feature method took approximately five minutes to process a single image. This made working on datasets that used full images unwieldy and impossible to use.

There were several other datasets that we would have liked to have tested on but were time consuming because they involved full image detection. This includes the Caltech
Pedestrian dataset [85] and the pascal challenges\(^2\). We wanted to be able to test on a large number of datasets rather than concentrate on a few that required more time.

Alternatives to a sliding window do exist. For example Shah et al. use SIFT descriptors to identify salient regions which could belonging to the object class [20]. The sparsity of these features greatly reduce the number of locations that need to be tested and the SIFT descriptors are scale invariant as well. Spinello et al. dismiss using feature points to find salient regions because the number of features is small compared to the size of the humans they were trying to detect [21]. They segment the image and use the segments as regions to identify a human and use scale invariant features to avoid having to search for humans at different scales, a method also used by Leibe et al. [86]. Lampert et al. propose a branch-and-bound scheme that converges to a globally optimal solution [87]. To find multiple objects the method is performed repeatedly, removing previous solutions as they are found. Using an alternate to a sliding window would greatly increase the speed of ECO features when using full images but it has not yet been pursued.

As the ECO feature method was being designed, it was obvious up front that the method would be computationally expensive. Fast hardware and good design would be necessary to make the method perform in real time. As this work started with the design of the Histograms of Oriented Gradients method on a GPU, at the onset it was planned that eventually the ECO features method would be implemented on a GPU. During the time of this research, GPU implementations of many of OpenCV functions were added to OpenCV with plans to include all of them. This opens the door in the future to easily use a GPU to find and use ECO features.

\(^2\)http://pascallin.ecs.soton.ac.uk/challenges/VOC/voc2011/
Chapter 6

Experiments

To test the ECO features algorithm, a variety of datasets were used. No parameters were tuned for any dataset; each was trained and tested in the exact same way. For each dataset tested, the best published results for that dataset were used for comparison. These methods are described below.

**Fergus [88]** Fergus et al. create a constellation of parts and then represent the shape, appearance, occlusion, and relative scale using a probabilistic representation. Classification is done using a Bayesian approach. Their method looks for salient regions over both location and scale.

**Serre [89]** Serre et al. detect objects by building a hierarchical system that alternates between simple and complex layers. The simple layers combine their inputs with a bell-shaped tuning function and the complex layers combine their inputs through a maximum operation. The first layer consists of a battery of Gabor filters at various scales and orientations.

**Schwartz [90]** Schwartz et al. augment HOG features with texture and color. They perform feature extraction using partial least squares analysis to reduce the dimensionality of their feature set. Their software is available online\(^1\).

**Tuzel [91]** Tuzel et al. use covariance matrices as object descriptors that are represented as a connected Riemannian manifold. Their main contribution is a novel method for classifying points that lie on a connected Riemannian manifold using the geometry of the space.

\(^1\)http://www.umiacs.umd.edu/ schwartz/softwares.html
Burl et al. identify volcanoes using a two pass system, one phase to identify candidate regions and another phase to do the final classification. They use PCA to reduce the input dimensionality and then use a Bayesian classifier.

6.1 Caltech Datasets

Four datasets are used from the Caltech datasets [93]: motorbikes, faces, airplanes, and cars. For each dataset\(^2\), the images were split into training and test sets according to Fergus et al. [88] so that good comparisons could be made to other methods.

6.1.1 Caltech Motorbikes

The Caltech motorbikes dataset consists of road, dirt, and bullet bikes from different time periods. Some motorbikes have a white background as they were taken from websites, and others are bikes with normal city scape backgrounds. All motorbikes are shown from a side profile and oriented so that the front of the motorbike is on the right side of the image. There are some partial occlusions, mostly people standing in front the motorbike. The negative images consist entirely of roads without any motorbikes present. Example images from the dataset are shown in Figure 6.0 and Table 6.0 shows how many images are in the dataset. All images in the dataset were scaled to 100 × 60 and converted to greyscale.

\(^{2}\text{http://www.robots.ox.ac.uk/~vgg/data3.html} \)

Figure 6.1: Example images of the Caltech motorbikes datasets after being scaled and having color removed.
Table 6.1: Number of images in the Caltech Motorbikes datasets.

<table>
<thead>
<tr>
<th>Training Images</th>
<th>Testing Images</th>
</tr>
</thead>
<tbody>
<tr>
<td>400</td>
<td>400</td>
</tr>
</tbody>
</table>

6.1.2 Caltech Faces

The images in the Caltech faces dataset were scaled to $100 \times 132$ and then converted to greyscale. The scaled images retain the aspect ratio of the original images but in a smaller form. Examples of both face and non-face objects from the dataset are shown in Figure 6.1. There are some difficulty due to poor lighting, partial occlusions, some drawn faces, variation due to hair styles, sexes, clothing, etc. The non-face images are random image patches taken from indoor and outdoor scenes. The number of images in the dataset are given in Table 6.1.

Figure 6.2: Example images of the Caltech faces datasets after being scaled and having color removed.

Table 6.2: Number of images in the Caltech faces datasets.

<table>
<thead>
<tr>
<th>Training Images</th>
<th>Testing Images</th>
</tr>
</thead>
<tbody>
<tr>
<td>218</td>
<td>212</td>
</tr>
</tbody>
</table>
6.1.3 Caltech Airplanes

The Caltech airplanes dataset consists mostly of commercial planes but also has military, small engine, and older planes in the dataset. All of the planes are oriented with the front of the plane to the right. Images have been scaled to $225 \times 73$ and converted to greyscale. Like the Caltech faces dataset the non-airplane images are random image patches taken from indoor and outdoor scenes. Examples from the dataset are shown in Figure 6.2 and the number of images in the dataset are shown in Table 6.2.

![Figure 6.3: Example images of the Caltech airplanes datasets after being scaled and having color removed.](image-url)
Table 6.3: Number of images in the Caltech airplanes dataset.

<table>
<thead>
<tr>
<th>Training Images</th>
<th>Testing Images</th>
</tr>
</thead>
<tbody>
<tr>
<td>400</td>
<td>400</td>
</tr>
</tbody>
</table>

6.1.4 Caltech Cars

The Caltech cars dataset consists of images of cars taken from behind the vehicle. Vehicles vary in make and model and the distance the vehicle is from the camera taking the image. All images were scaled to $150 \times 118$ and converted to greyscale. Like the Caltech motorbikes dataset the non-car images are of roads without any vehicles directly in the image. Examples images are given in Figure 6.3 and the number of images in the dataset are shown in Table 6.3.

![Example images of the Caltech cars datasets after being scaled and having color removed.](image)

Table 6.4: Number of images in the Caltech cars datasets.

<table>
<thead>
<tr>
<th>Training Images</th>
<th>Testing Images</th>
</tr>
</thead>
<tbody>
<tr>
<td>400</td>
<td>400</td>
</tr>
</tbody>
</table>
6.1.5 Results on Caltech Datasets

Table 6.4 shows the performance of ECO features, perfectly classifying all four Caltech datasets, alongside the results of other methods. Fergus et al. [88] and Serre et al. [89] represent the current best published results on the Caltech datasets. Schwartz et al. [90] did not publish results on the Caltech dataset but we tested their method on the Caltech datasets, and it was perfect in its classification.

<table>
<thead>
<tr>
<th>Method</th>
<th>Fergus [88]</th>
<th>Serre [89]</th>
<th>Schwartz [90]</th>
<th>ECO</th>
</tr>
</thead>
<tbody>
<tr>
<td>Motorbikes</td>
<td>95.0</td>
<td>98.0</td>
<td>100.0</td>
<td>100.0</td>
</tr>
<tr>
<td>Faces</td>
<td>96.4</td>
<td>98.2</td>
<td>100.0</td>
<td>100.0</td>
</tr>
<tr>
<td>Airplanes</td>
<td>94.0</td>
<td>96.7</td>
<td>100.0</td>
<td>100.0</td>
</tr>
<tr>
<td>Cars</td>
<td>95.0</td>
<td>98.0</td>
<td>100.0</td>
<td>100.0</td>
</tr>
</tbody>
</table>

Additional experiments were performed to see how many ECO features were actually required to perfectly classify the datasets. Figure 6.4 shows the results on the Caltech datasets using fewer ECO features. The axes in Figure 6.4 are labeled with the standard metrics of precision and recall as defined in Equations 6.1 and 6.2 where \( t_p \) is the number of true positives, \( t_n \) is the number of true negatives, \( f_p \) is the number of false positives, and \( f_n \) is the number of false negatives. The number of ECO features used is reduced by lowering the parameter \( X \) used by Adaboost. \( X \) starts at 400 ECO features and then one by one the least influential ECO feature was removed. At most 25 features were needed to perfectly classify the faces and cars datasets while the airplanes dataset only needed ten features and the motorbikes dataset only needed four features to obtain perfect results. These results show that reducing the number of features, even drastically, still leads to good accuracy overall.

\[
\text{precision} = \frac{t_p}{t_p + f_p}, \quad (6.1)
\]
Figure 6.5: Performance on the Caltech datasets using varying number of ECO features.

\[ \text{recall} = \frac{t_p}{t_p + f_n}. \]  

(6.2)

Table 6.6: Summary of how the INRIA Person Dataset was used. The label Pos stands for positive human examples and the label Neg stands for negative human example.

<table>
<thead>
<tr>
<th>Training Set</th>
<th>Testing Set</th>
</tr>
</thead>
<tbody>
<tr>
<td>Find Features</td>
<td>Train AdaBoost</td>
</tr>
<tr>
<td>Pos</td>
<td>Neg</td>
</tr>
<tr>
<td>1,489</td>
<td>1,576</td>
</tr>
</tbody>
</table>
Figure 6.6: Example images from the INRIA Person dataset.

6.2 INRIA Person Dataset

The next step has to test ECO features using the INRIA Person Dataset [76]. Examples of the dataset are shown in Figure 6.5. The dataset is very challenging because it contains humans with large variations in pose, lighting conditions, background, clothing, and partial occlusion. Table 6.5 gives a break down of how the dataset was used.

Table 6.7: Comparison to state-of-the-art methods on the INRIA Person dataset.

<table>
<thead>
<tr>
<th>Method</th>
<th>Miss rate at $10^{-4}$ false positive rate</th>
</tr>
</thead>
<tbody>
<tr>
<td>HOG</td>
<td>12%</td>
</tr>
<tr>
<td>Dollár [37]</td>
<td>7%</td>
</tr>
<tr>
<td>Tuzel [91]</td>
<td>7%</td>
</tr>
<tr>
<td>Dollár [94]</td>
<td>4%</td>
</tr>
<tr>
<td>Schwartz [90]</td>
<td>3%</td>
</tr>
<tr>
<td>ECO Features</td>
<td>3%</td>
</tr>
</tbody>
</table>

Table 6.6 gives a summary of results for the state-of-the-art classification algorithms on the INRIA Person dataset. A false positive rate of $10^{-4}$ is a common comparison point for algorithms on this dataset. Equations 6.3 and 6.4 define the miss rate and false positive rate that are used in the table and elsewhere in this work, where $f_p$ is the number of false
positives, $f_n$ is the number of false negatives, $t_p$ is the number of true positives, and $t_n$ is the number of true negatives.

$$\text{miss rate} = \frac{f_n}{t_p + f_n},$$  \hfill (6.3)

$$\text{false positive rate} = \frac{f_p}{t_p + f_p + t_n + f_n}.$$  \hfill (6.4)

![Graph showing performance of ECO features on the INRIA Person Dataset using varying number of features.](image)

**Figure 6.7:** Performance of ECO features on the INRIA Person Dataset using varying number of features.

Figure 6.6 shows the results when varying the number of ECO features used. 400 ECO features were used at the start and then one by one the least influential ECO feature
was removed. Since identical results are obtained using 100 or more ECO features, no results using more than 100 ECO features are shown to make the graph more readable. Reducing the number of features by 90% gives only a 7% reduction in recall at a 90% precision rate.

6.3 Volcanoes on Venus – SAR Imagery

The volcanoes of Venus dataset [95] is one result of the Magellan mission to Venus from 1989 through 1994 and contains 134 1024 × 1024 SAR images. With the volume of images resulting from the mission, automated methods for analyzing the data were sought for [92].

![Figure 6.8: Example images from the volcanoes on Venus dataset. The first row are positive examples and the second row are negative examples.](image)

Figure 6.8: Example images from the volcanoes on Venus dataset. The first row are positive examples and the second row are negative examples.

The volcanoes on Venus dataset is a very challenging dataset to classify even for humans. The top row in Figure 6.7 shows examples of volcanoes and the bottom row contains sample images without a volcano. Although the images represent the highest SAR resolution available at the time, they still lack detail to make the volcanoes obviously distinguishable from the rest of the planet. In an effort to quantify uncertainty, examples were labeled by geologists into five categories: category 1 - almost certainly a volcano, category 2 - probably a volcano, category 3 - possibly a volcano, category 4 - a visible pit but not much evidence of other volcano traits, category 5 - not a volcano. This labeling represents the ground truth
for the dataset. A confusion matrix of labeling given by two geologists is shown in Table 6.7 and indicates how difficult the dataset is, given that the ground truth is so difficult to establish. A confusion matrix compares the classification of two entities, generally the predicted classification and the actual classification. Here the labeling of one geologist is compared against the other geologist. If they agreed on every volcano then the confusion matrix, for example, would show that whenever geologist A labeled the volcano as category 1 geologist B would also label it as category 1.

**Table 6.8:** Confusion matrix created by two expert geologist and together represent the ground truth for the dataset.

<table>
<thead>
<tr>
<th></th>
<th>Label 1</th>
<th>Label 2</th>
<th>Label 3</th>
<th>Label 4</th>
<th>Label 5</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>geologist A</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Label 1</td>
<td>19</td>
<td>9</td>
<td>13</td>
<td>1</td>
<td>4</td>
</tr>
<tr>
<td>Label 2</td>
<td>8</td>
<td>8</td>
<td>12</td>
<td>4</td>
<td>8</td>
</tr>
<tr>
<td>Label 3</td>
<td>4</td>
<td>6</td>
<td>18</td>
<td>5</td>
<td>29</td>
</tr>
<tr>
<td>Label 4</td>
<td>1</td>
<td>5</td>
<td>1</td>
<td>24</td>
<td>16</td>
</tr>
<tr>
<td>Label 5</td>
<td>3</td>
<td>5</td>
<td>37</td>
<td>15</td>
<td>X</td>
</tr>
</tbody>
</table>

ECO features were trained using multiple $30 \times 30$ pixel regions from each image. The dataset is partitioned into five parts called HOM4, HOM38, HOM56, HET5, and HET36. HOM means that the images are fairly homogeneous and HET means that the images were taken from different parts of the planet and have more variability, and the number indicates the number of images in the partition. Within each partition, cross-validation is performed rotating which images are used for testing. Half of the volcano examples taken from the ground truth were used for training and the other half for testing. An equal number of non-volcano samples were sampled randomly from the rest of the image, taking care that there was no overlap with labeled volcanoes. When training the AdaBoost models, not all 400 weak classifiers were included because the error rates for the weak classifiers reached 50% (see Algorithm 2).

An attempt was made to do a comparison to Burl et al. [92] but there was insufficient information to make a fair and accurate comparison. It was possible to make a comparison
against the Schwartz method [90] by downloading available code\(^3\). Their method was trained and tested using the same image sets as was used for ECO features. The results of this test are shown in Figure 6.8. ECO features outperformed Schwartz on every test and did significantly better on the HET partitions. It appears that the Schwartz method [90] performs better on the HOM\(4\) partition for miss rates greater than 50\%, but this is only a result of the low number of images in the partition. Removing the last point in the graph gives a better view of the results, since both have 100\% miss rate at zero false positives.

### 6.4 BYU Fish Dataset

Assessing the population, size, and taxonomy of fish is important in order to manage fish populations, regulate fisheries, and evaluate the impact of man made structures such as dams. Currently, fish surveys are done using catch and release methods, diving, and fish tagging, all of which are costly and time consuming. Automating this process can save valuable resources, increase the quantity of data available, and improve accuracy over current methods that are error-prone. Catching and removing invasive fish species are also a concern [96–98].

Similar research has been published in this area for monitoring and measuring fish. Zion et al. use an underwater sorting machine to sort three species of fish in a fishery pond [99]. They use features that they designed by hand from fish silhouettes and a Bayes classifier to determine the species. Rova et al. use deformable template matching and a support vector machine to differentiate between two very similarly shaped fish [100]. Lee et al. compared contour segments between fish and a database to identify four target species [101]. Chambah et al. use hand selected shape, color, texture features, motion features, and a Bayes classifier to identify fish in an aquarium [102]. Cadieux et al. use silhouettes and again a set of hand selected features with a combination of a Bayes classifier, a learning vector quantization, and a neural network to classify fish [103].

Previous research methods for automated fish identification and taxonomy have depended on a human expert to design the features the identification algorithm uses. Adapting

\(^3\)http://www.umiacs.umd.edu/~schwartz/softwares.html
to other environments with a different fauna is difficult, time consuming, and costly. Using ECO features allows the system to easily adapt to new fauna and circumstances.

The fish images used are from field study images taken by Brigham Young University’s biology department. The fish were captured, photographed, and released. There are four fish species represented in the dataset: Yellowstone Cutthroat, Cottid, Speckled Dace, and
Whitefish. Samples of each species from the dataset are shown in Figure 6.9. In the dataset there are 246 Yellowstone Cutthroat, 121 Cottids, 140 Speckled Dace, and 174 Whitefish.

![Figure 6.9: Examples of the four fish species. From the top row to the bottom row the species are Yellowstone Cutthroat, Cottid, Speckled Dace, and Whitefish.](image)

The raw images were pre-processed in order to make a dataset appropriate for object recognition. The image was rotated so that the head of the fish was on the right side of the image. Then, each image in the fish dataset was cropped and resized to a standard $161 \times 46$ pixels.

No color information is used because in many fish species recognition applications, color is either not present or not reliable due to water opaqueness and inability to control lighting conditions. The ECO features then have to key into shape information in order to distinguish one species from another.

### 6.4.1 Cross Validation

Five fold cross validation was performed to test the ability of ECO features to distinguish each fish species. Each image in the dataset of a chosen species was treated as the
positive example and all the other species made up the negative examples. Using five fold cross validation one fold is treated as the test set and the remaining four folds are used for training the ECO features. For each fold and species, 10 to 16 ECO features are found before error rates rise above 50% during AdaBoost training. Once the ECO features are found, the images in the current fold are used to compute a classification accuracy.

Table 6.9: The classification accuracy when doing five fold cross validation (one fold for testing and four for training) for each species. One species is treated as the positive examples and the other species form the negative examples.

<table>
<thead>
<tr>
<th>Species</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>Y. Cutthroat</td>
<td>99.3</td>
<td>99.3</td>
<td>100</td>
<td>100</td>
<td>100</td>
</tr>
<tr>
<td>Cottid</td>
<td>100</td>
<td>100</td>
<td>100</td>
<td>99.3</td>
<td>98.4</td>
</tr>
<tr>
<td>Speckled Dace</td>
<td>99.3</td>
<td>99.3</td>
<td>100</td>
<td>100</td>
<td>99.3</td>
</tr>
<tr>
<td>Whitefish</td>
<td>97.8</td>
<td>100</td>
<td>98.6</td>
<td>98.6</td>
<td>99.2</td>
</tr>
</tbody>
</table>

The results are given in Table 6.8 with an average classification accuracy of 99.4%. This shows that the ECO features performed very well in discriminating between the four fish taxonomies. It took approximately three minutes to find the ECO features for a single fold during cross validation. Finding the ECO features is an off-line process and only needs to be run as the location and fauna change, as new species are added, or some other phenomena changes the environment. This makes using ECO features very adaptable and easy to use, without sacrificing the accuracy of the system.

6.5 MNIST Dataset

The MNIST dataset is composed of 60,000 training images and 10,000 test images of handwritten digits\(^4\). Examples from the dataset are shown in Figure 6.10. The dataset is of interest because of several important applications that would benefit from very accurate recognition of handwritten characters. The dataset has received a lot of attention over the years.

\(^4\)http://yann.lecun.com/exdb/mnist/
Unlike the other datasets that have been used so far, the MNIST dataset requires multi-class classification, each class being a single numerical digit. To do multi-class classification, a binary classification model was first made for each digit. This is done by using the images of all the other digits as the negative examples. A one-vs-all method was used where each binary model classified the example as belonging to its digit and the binary classifier that was the most confident dictated what digit the example was. This is one of the simplest methods for binary classification and there are some problems with it. There is nothing that constrains each AdaBoost model that was trained to vote in the same confidence range, making the relative comparison between models difficult. There were many other methods for creating a multi-class classifier [104] that could have been attempted, but that was not the point of this research. Overall, the goal was not to spend too much time on any given dataset to improve results since the ECO features algorithm is meant for general object recognition and not for any specific datasets.

Only the 60,000 images that were provided in the dataset were used for training. Many works deform the training images in order to augment the training images. Ciresan et al. use a six layer neural network with up to 2500 neurons in a layer and deform the training images every epoch during training. They run for 20-30 epochs meaning that their training set consists of 1.2-1.8 million training images. They use a GPU in order to deal with the training of such a large neural network and the deformation of training images.
Many previous works also operate on the raw pixel data. In the introduction of this work it was stated that higher level symbols or features were needed in order to get good classification results. The images in the dataset are all $28 \times 28$ pixels in size and because of the low dimensionality of the images and the number of training images, especially once augmented with deformed training images, it allows the raw pixel data to be used.

Using multi-class classification resulted in a classification accuracy of 97.92% compared to 99.65% classification accuracy obtained by Ciresan et al. Figure 6.11 shows 20 of the 208 examples that were misclassified. There were 269 examples where more than one model predicted that the example belonged to its class. By themselves the binary classifiers averaged 99.4% classification accuracy. With the amount of attention that the dataset has received and the effort that others put into the dataset by expanding the number of training images considerably, we believe that ECO features perform very well considering that only the default settings were used and no extra work was done to improve the results on this
particular dataset. We also believe that more time spent on multi-class classification would improve the results on this particular dataset.

6.6 Daimler Pedestrian Classification Benchmark Dataset

The Daimler Pedestrian Classification Benchmark dataset is a collection of pedestrian and non-pedestrian images cut out from video of an outdoor urban environment introduced by Munder and Gavrila [105]. The dataset is divided into three training sets and two testing sets with 4800 pedestrian images and 5000 non-pedestrian images per set. All images are scaled to $18 \times 36$ pixels. Figure 6.12 shows examples of images from this dataset. Due to the low resolution of the images the pedestrians are difficult to make out.

![Figure 6.13: Example images from the Daimler Pedestrian Classification Benchmark dataset.](image)

In order to compare to the original work [105] the same method for training and testing was employed. During training, three AdaBoost models were created, each one trained on two of the three training sets. Then each model was tested on each of the two testing sets. From this six ROC curves could be obtained. Figure 6.13(a) shows the mean and variance over the six ROC curves for the ECO feature method and Figure 6.13(b) shows
the same information for three of the best methods used by Munder and Gavrila. Figure 6.13(c) shows the work of Tuzel et al. [91] compared to the three best methods by Munder and Gavrila using miss rate vs false positive rate rather than an ROC curve, with Tuzel et al. having the lower miss rate. ECO features perform as well as Munder and Gavrila’s best method. Other methods that perform nearly the same are Maji et al. [61] and Jung and Kim [106]. Tuzel et al. are able to perform best on this dataset.

Figure 6.14: Performance of (a) ECO features, (b) Munder and Gavrila, (c) and Tuzel et al. on the Daimler Pedestrian Classification Benchmark dataset.
Chapter 7

Further Analysis of ECO Features

As has been presented, the performance of ECO features on object detection is, on individual datasets, among the best, and generalizes well across multiple datasets. It is difficult, however, to understand why ECO features perform as well as they do and what is happening in the underlaying framework. The difficulty lies in the use of a genetic algorithm; in general it is difficult to understand the results they produce [82]. The genetic algorithm is given a fitness score to maximize but is not told how to maximize it. The same is true of biological evolution from which genetic algorithms are modeled. The process of biological evolution and its fitness score is understood but it is difficult to say exactly how a certain species came into being. However, just as with biological evolution there are some conclusions that can be made. Here, the focus turns from the raw performance to taking a closer look at how ECO features are composed, and an attempt to uncover what the genetic algorithm is finding.

7.1 Visualization

Since ECO features are performing a series of image transforms, what the genetic algorithm is doing can, in many cases, be understood by analyzing the images produced after each transform, $V_i$ (Equation 4.1). These images are produced by averaging all the positive or negative examples, after each transform, and then normalizing the results so they can be viewed as an image. Normalization is done according to Equation 7.1. While the normalization is necessary to view the average output of the transforms as an image, it also has some negative effects. The normalization causes the perceived contrast differences to be relative, and the actual image intensity magnitudes are not evident. The normalization of the average over the positive and negative examples is done separately which makes some
comparisons between the two difficult. Despite the disadvantages, the images do provide very clear information about what the genetic algorithm is finding.

\[ V_i(x, y) = \frac{V_i(x, y) - \min(V_i)}{\max(V_i) - \min(V_i)} \]  

(7.1)

The output of the final transform, \( V \), becomes the input to the perceptron of the ECO feature. Those inputs are multiplied by the perceptron weights, \( W \). The greater the magnitude of the perceptron weight, the more important the input that is connected to that weight is. So if the perceptron weights are viewed as an image, in the same way that the output of the other transforms is viewed as an image, the relative importance of the inputs to the perceptron can be viewed. The positive and negative magnitude weights are viewed separately so that the importance of the weights for classifying the image as object or non-object can be seen. The following subsections analyze the visualization of ECO feature trained on several of the datasets that were used. The visualization gives an understanding of what information the ECO features found on any given datasets.

### 7.1.1 Visualize Caltech Faces

Figure 7.0 shows a visualization of four ECO features trained on the Caltech faces dataset. Many of the ECO features trained on this dataset cover the subregion that includes the eyes, nose, and mouth. ECO feature A performs three erode iterations and then an adaptive threshold with a blocksize large enough to cover the entire subregion. This brings out some of the darker regions of the face. Positive perceptron weights show that they key into the eyes and the region between the nose and mouth. The negative perceptron weights show that bright regions in other parts of the face will cause the image to be classified as non-face.

ECO feature B looks at just the right side of the face performing a histogram equalization followed by a Laplacian transform. It is a little difficult to see from the perceptron weights exactly what information is being keyed into, but it appears that it keys into the sharp contrast line formed between the face and the background as well as from the hair line to the face.
Figure 7.1: Visualization of four ECO features trained on the Caltech faces dataset. Transforms are as follows: A.1-erode, A.2-adaptive threshold, B.1-histogram equalization, B.2-Laplacian, C.1-gradient, C.2-normalize, D.1-erode, D.2-dilate, and D.3-Hough circle. A.3, B.3, C.3 and D.4 are a visualization of the negative magnitude perceptron weights and A.4, B.4, C.4, and D.5 are the positive magnitude perceptron weights.

ECO feature C first performs a "simple gradient" in the y direction (see A.25 in the appendix for more information about how the gradient transform works) that is then normalized. The eyes and partly the mouth stand out strongest in the positive examples. The negative weights of the perceptron show that smooth regions in the y direction are expected in the regions around the eyes and mouth.

ECO feature D performs two erodes, a dilate, and then a Hough circle transform. It is obviously looking for the half circle that the chin on a human face forms.
7.1.2 Visualize Caltech Airplanes

Figure 7.1 shows the visualization of four ECO features that were trained on the Caltech airplanes dataset. ECO feature A is very typical of many of the ECO features on the dataset because it focuses on finding the shape of the brighter airplane against a darker background. This ECO feature shows that the typical shape of an airplane is more like a commercial airplane, which is consistent with the dataset that features more commercial planes than anything else. ECO feature A performs a gradient transform using a Kirsch compass kernel in the x direction followed by three iterations of a dilate transform. The positive perceptron weights are strongest on the shape of the plane, particularly the tail of the plane. The negative weights of the perceptron are strongest outside where the airplane is suppose to be.

<table>
<thead>
<tr>
<th>A</th>
<th>A.1</th>
<th>A.2</th>
<th>A.3</th>
<th>A.4</th>
</tr>
</thead>
<tbody>
<tr>
<td>B</td>
<td>B.1</td>
<td>B.2</td>
<td>B.3</td>
<td>B.4</td>
</tr>
<tr>
<td>C</td>
<td>C.1</td>
<td>C.2</td>
<td>C.3</td>
<td>C.4</td>
</tr>
<tr>
<td>D</td>
<td>D.1</td>
<td>D.2</td>
<td>D.3</td>
<td>D.4</td>
</tr>
</tbody>
</table>

Figure 7.2: Visualization of four ECO features trained on the Caltech airplanes dataset. The transforms represented are as follows: A.1-gradient, A.2-dilate, B.1-dilate, B.2-Hough circle, C.1-Laplacian, C.2-difference of Gaussians, C.3-histogram equalization, D.1-adaptive threshold, and D.2-Gaussian. A.3, B.3, C.4, and D.3 are visualizations of the negative magnitude weights of the perceptron and A.4, B.4, C.5, and D.4 are visualizations of the positive magnitude weights of the perceptron.

ECO feature B focuses on identifying the nose of the airplane. Again the typical shape of the nose of the airplane looks more like a commercial airplane and does not match exactly
the airplane shown in the figure. ECO feature B performs a dilate, then converts from an
eight bit unsigned to an eight bit unsigned which is not shown in the figure because the
transform does nothing, followed by a Hough circle transform. The Hough circle transform
allows the ECO feature to try to recognize the rounded front of the airplane. This is an
element of how ECO features can contain elements that do not help but do not hurt the
fitness score of the ECO feature during training.

ECO feature C appears to be trying to identify texture. Figure 7.2 shows the average
of all the images in the training set of the Caltech airplanes dataset. The subregion of ECO
feature C, however, is at the top of the image and as shown in the average airplane image,
this appears most often not to intersect with the airplane in the image. Many of the images
in the Caltech airplanes dataset show the aircraft in the sky, where the background does not
have much texture. The random images that make up the negative examples, however, tend
to have a lot of texture and it seems this is what ECO feature C is trying to find. ECO
feature C performs a Laplacian transform followed by a difference of Gaussians transform
and a histogram equalization transform. There is no shape information that is discernible
in the visualizations of the transforms.

ECO feature D is looking to find the difference in contrast between an airplane and
its background. The ECO feature first performs an adaptive threshold and then blurs the
image with a Gaussian. The adaptive threshold pushes the pixels making up the plane to
black and the background to white. The negative perceptron weights are strongest along
where the airplane should be, and the positive perceptron weights are strongest outside of
where the airplane should be.

**Figure 7.3:** The average over all the airplanes in the training set of the Caltech airplanes
dataset.
7.1.3 Visualize Caltech Motorbikes

Figure 7.4: Visualization of four ECO features trained on the Caltech motorbikes dataset. Transforms are as follows: A.1-histogram equalization, A.2-erode, B.1-normalize, B.2-Sobel, B.3-dilate, C.1-Gabor, C.2-Hough circle, D.1-Sobel, and D.2-integral image. A.3, B.4, C.3, and D.3 are positive perceptron weights and A.4, B.5, C.4, and D.4 are negative perceptron weights.

Figure 7.3 shows a visualization of four ECO features trained on the Caltech motorbikes dataset. ECO feature A performs an histogram equalization followed by four iterations of an erode transform. The Caltech motorbikes is different from the faces and airplanes datasets in that the negative examples are all of one type, road scenes. This is evident when looking at the averaged negative images after the transforms are performed on them and the perceptron weights for ECO feature A. They reflect that there is some information to be
gained by knowing something about the negative examples. The negative perceptron weights are very bright around where the white lines of the road are typically found and where the tire of the motorbike is generally found.

ECO feature B also exploits information about the negative examples by keying into the horizon that is found in the negative examples. ECO feature B is a normalize transform followed by a Sobel operator and three iterations of a dilate transform.

There are quite a few small subregions used by ECO features that were trained on the Caltech motorbikes dataset with ECO feature C as an example. The visualizations have been resized because the images were only $24 \times 23$ pixels in size. ECO feature C is a Gabor filter followed by a Hough circle transform. The ECO feature is using the Hough circle transform to find the back half curve of the front tire.

ECO feature D is interesting because averaged images over the positive and negative examples look fairly similar, much of which is a product of how the averaged images are normalized for display as discussed at the start of this section. ECO feature D is a Sobel operator followed by an integral image. There are three regions in the integral image where large values indicate that it is a motorbike and regions that indicate that it is road.

7.1.4 Visualize Caltech Cars

As with the Caltech motorbikes dataset, the Caltech cars dataset uses street scenes as the negative examples. The consistency in the negative examples provides more information that can be leveraged by the ECO features. ECO feature A appears to be detecting the shadow of the vehicle which is much darker on average than the street scenes that make up the negative examples. Using a vehicle’s shadow was recently proposed by Rosebrock et al. as a means to localize vehicles [107]. ECO feature A performs a Gabor filter followed by a Gaussian blur. The positive weights of the perceptron show that it is looking for the edges around the car to be bright but not in the area that should be under the vehicle. The negative weights show that if there is no shadow it should be classified as road.

ECO feature B is looking for the left edge of a vehicle. This region will typically be darker than the road. ECO feature B performs a histogram equalization, a dilate, and
Figure 7.5: Visualization of four ECO features trained on the Caltech cars dataset. Transforms are as follows: A.1-Gabor, A.2-Gaussian blur, B.1-histogram equalization, B.2-dilate, B.3-Harris corner, C.1-histogram, C.2-integral image, D.1-Gabor, D.2-gradient, D.3-dilate, D.4-Gabor, and D.5-median blur. A.3, B.4, C.3, and D.6 are visualizations of the positive magnitude weights of the perceptron and A.4, B.5, C.4, D.7 are visualizations of the negative magnitude perceptron weights.
a Harris corner strength transform. The Harris corner strength transform shows very low corner strength on the road, where there will be strong corners for the vehicle.

ECO feature C has a very small subregion and the visualization has been scaled up in order to make it easier to see. This ECO feature uses a histogram and then an integral image to find the gradient in intensity from the vehicle to the road.

ECO feature D is an example of an ECO feature that is difficult to be understood; even after looking at the visualization. With some ECO features it is difficult to understand the visualization because it uses a transform that produces an output that does not lend itself to visualization, such as a DFT or a histogram. ECO feature D is different because its transforms can be visualized. ECO feature D performs a Gabor filter, a Kirsch compass gradient in the y direction, four iterations of a dilate, another Gabor filter with a large kernel window, and then finally a median blur transform.

### 7.1.5 Visualize INRIA Person Dataset

Figure 7.5 shows visualizations of two ECO features that were trained on the INRIA person dataset. The two ECO features were chosen with subregions that cover the entire image to give an idea of what information is typical found on the entire object. The transforms in both examples show a prominent human shape when averaged over the positive examples with perceptron weights that trigger on that shape information. The two ECO features shown here are typical of many of the ECO features for the INRIA person dataset as it focuses on this basic human shape that features prominently the head and feet and a basic geometric outline of the body. The average over the negative example shows how random the examples are because the average looks close to white noise.

Figure 7.6 shows three ECO features that focus on specific parts of the human body: the head, waist, and torso respectively. ECO feature A does a Harris corner strength transform with a support window that is larger than the subregion it is operating on and then creates an integral image from that. There is one region of that integral image that indicates that the object is non-human, which means that if the sum of the Harris corner transform up to that region is in a certain range, it will not classify the image as human. ECO feature B appears to be triggering off of the edges of the human body at the waist, but it is different
Figure 7.6: Visualization of two typical ECO features that show how shape information is being exploited to classify humans from non-humans. Transform represented are A.1-Gaussian blur, A.2-dilate, A.3-difference of Gaussians, A.4-adaptive threshold, B.1-difference of Gaussians, B.2-Sobel operator, B.3-normalize, and B.4-census. A.5 and B.5 are visualizations of the negative magnitude weights of the perceptron and A.5 and B.5 are visualizations of the positive magnitude weights of the perceptron.

in that it does not use any transforms that would be considered edge detection. ECO feature C shows an example of an ECO feature that is hard to visualize. The final transform is a pixel statistics transform that stores various pixel statistics in a one dimensional array.

Figure 7.7 show two interesting ECO features. ECO feature A shows a subregion where very little of the human is present in the subregion, thus placing emphasis on the texture of the background image. Evidently if a lot of texture appears the image usually belongs to the negative examples. ECO feature B uses a distance transform that creates a
unique result. The distance transform calculates the distance to the nearest black pixel for every pixel. If you averaged the distance transform over a large number of random images you would see something like that in Subfigure B.2 averaged over the negative examples. In this case, however, the negative images are brighter on the upper half of the image than the lower, which is consistent with outdoor images. The positive examples show the same sort of image except in the middle of the image where the human is darker. Neither of these examples would be typical of transforms that a human expert would try.
Figure 7.8: Visualization of two ECO feature that are different from others trained on the INRIA person dataset. The transforms represented are: A.1-histogram equalization, A.2-distance transform, A.3-Sobel operator, B.1-histogram equalization, B.2-resize, B.3-difference of Gaussians, B.4-Harris corner strength, and B.5-Laplacian. A.4 and B.7 are visualizations of the negative magnitude weights of the perceptron and A.5 and B.6 are visualizations of the positive magnitude weights of the perceptron.

7.1.6 Visualize BYU Fish Dataset

Figure 7.8 shows a visualization of four ECO features that have been trained on the BYU fish dataset using the species Whitefish as the positive examples and the other three species as the negative examples. The visualizations show the shape information that is being extracted that separates the fish species from each other. ECO feature A performs an adaptive threshold and then resizes the image to make it smaller, but the visualization of
the resize transform and the perceptron weights are shown resized back to the size before the resize transform so that it can be viewed easier. ECO feature A is looking at the shape of the head and some of the shape of the body as evidenced in the visualization of the positive perceptron weights.

Figure 7.9: Visualization of four ECO features trained on the BYU fish dataset. Transforms are as follows: A.1-adaptive threshold, A.2-resize, B.1-Gabor filter, B.2-erode, C.1-difference of Gaussians, C.2-census, C.3-Harris corner strength, D.1-Harris corner strength, and D.2-discrete Fourier transform. A.3, B.3, C.4 and D.3 are visualizations of the negative magnitude perceptron weights and A.4, B.4, C.5, and D.4 are the positive magnitude perceptron weights.

ECO feature B performs a Gabor filter followed by a single erode transform along a long strip across the spine of the fish. The result of the Gabor filter and erode transform is a silhouette of that part of the fish. From the silhouette the positive perceptron weights are strongest when the dorsal fin is a particular position and the negative weights are strongest when the adipose fin is in a particular position.

ECO feature C takes a fairly large subregion of the fish that includes basically all of the fish except the head. It first performs a difference of Gaussians that really emphasizes the shape of the fish along the spine. It then performs a census transform and then a Harris corner strength transform. The positive perceptron weights show a bright spot at the point
where the tail connects to the rest of the body indicating that its position is important. The negative weights show if an anal fin shows up at a certain location in the image that it is not a Whitefish.

ECO feature D performs a Harris corner strength transform with a large window size followed by a discrete Fourier transform. Because of the Fourier transform the visualization is hard to understand. The dimensionality of the output image and of the perceptron is high but very little of it contains useful information. ECO features do not attempt to reduce the dimensionality of its output.

7.1.7 Visualize Volcanoes on Venus

Figure 7.9 shows four ECO features trained on the volcanoes on Venus dataset. Not surprisingly all of the ECO features that were trained on this dataset focus primarily on the vent of the volcano that appears as a bright dot in the center of some of the volcano images. ECO feature A looks at part of the vent of the volcano but also appears to be looking at the gradient on one side of the volcano. ECO feature A performs an adaptive threshold with a window large enough to include the entire subregion, a median blur again with a window large enough to include the entire subregion, and then a resize transform. The perceptron weights show that the area on the right side of the subregion are dark for a volcano and that the region around the neck of the volcano is a bright region.

The same observation for ECO feature A can be made about ECO feature B except on a smaller subregion of the image. ECO feature B performs a difference of Gaussians followed by a Gaussian blur.

ECO feature C performs a difference of Gaussians, a convert transform that does not actually change the type, and then a gradient transform using a Kirsch compass kernel in the x direction. ECO feature C is primary looking at the neck of the volcano but apparently also at the texture of the volcano. The visualization on the negative examples after transform C.1, C.2, and C.3 are more textured than the visualization of the positive examples.

The same observations for ECO feature C can be made about ECO feature D except on a smaller subregion of the image. ECO feature C performs a difference of Gaussians followed by a normalize transform.
Figure 7.10: Visualization of four ECO features trained on the volcanoes on Venus dataset. Transforms are as follows: A.1-adaptive threshold, A.2-median blur, A.3-resize, B.1-difference of Gaussians, B.2-Gaussian blur, C.1-difference of Gaussians, C.2-convert type, C.3-gradient, D.1-difference of Gaussians, and D.2-normalize. A.1, B.3, C.4, and D.3 are visualizations of the negative magnitude perceptron weights and A.5, B.4, C.5, and D.4 are the positive magnitude perceptron weights.

7.2 Speciation

Genetic algorithms discard creatures with the lowest fitness scores in each generation, which eventually leads to the convergence of the entire population. In general, if the fitness truly reflects the strength of the solution, this is the desired behavior. There are situations, however, where certain types of solutions evolve slower than other types of solutions, but if given the opportunity could eventually evolve to have a similar or even better fitness score. For example, Figure 7.10 shows an example search space. Searches starting at randomly
chosen locations will most likely converge on solution A because of the gradient and the magnitude of the fitness score the hill around solution A provides. Searches that start off of the hill around point A could find point B, which has a slightly higher fitness score, but it is going to take longer and cannot be discarded too soon because of its low fitness score. Speciation preserves diversity and innovation by allowing creatures to compete in niches rather than competing against the entire population [108] [109] [110].

![Figure 7.11: A search algorithm to find the maximum fitness score is much more likely to find point A before point B.](image)

In order for speciation to occur, ECO features must be allowed to compete in niches, rather than against a large population. One way to allow speciation to occur would be to define species and only allow ECO features to compete if they were from the same species. With 27 possible transforms and millions of possible combinations of those transforms, defining species is very difficult. It is hard to determine a distance measure between sequences of transforms that would not divide the ECO features arbitrarily.

A simpler method to provide speciation, that does not alter the genetic algorithm, is to train ECO features in smaller populations. In large populations certain combinations of transforms were observed to appear frequently. If, however, those combinations of transforms were not present, other transform combinations over time could mature into solutions with equally high fitness scores. This observation shows that some combinations of transforms
Figure 7.12: The two figures above show the count for each transform, the circles, that appear in an AdaBoost model. The larger the circle the more frequently it appears in the AdaBoost model. Each transform is then connected to other transforms that follow it in the ECO feature. The wider the line connecting two transforms the more frequently that transform pair appears in the AdaBoost model. Figure (a) visualizes an AdaBoost model trained from ECO features that were found without speciation, while (b) used speciation.

mature much faster within the genetic algorithm but do not necessarily perform any better than other combinations of transforms that mature more slowly.

In order to show the advantages of speciation, 500 ECO features were trained on the INRIA person dataset using a population size of 1000, and 500 ECO features were trained
using a population size of 50. Figure 7.11 shows the diversity of transforms and combinations of transforms between an AdaBoost model trained using speciation and without speciation. The circles in the figure represent the various possible transforms. The larger the circle, the more frequently that transform appears in the AdaBoost model. Each transform has lines that connect it to transforms that follow it in the transform sequence, with the thickness of the line indicating how often that combination of transforms appeared in the AdaBoost model. This represents in visual form the diversity that speciation allows. The model with speciation has many more combinations of transforms that appear and the counts for each transform are more uniformly distributed.

Figure 7.12 compares the accuracy of the two AdaBoost models mentioned above on the INRIA person dataset [76]. The model that was trained using speciation performs much better than the model that does not use speciation. The diversity of the ECO features when using speciation allows better generalization.

**Figure 7.13:** Comparison between ECO features with and without speciation on the INRIA person dataset.
Figure 7.13 shows the comparison between the HOG method alongside our ECO features without and with speciation. Also, Table 7.0 shows comparisons to other state-of-the-art object recognition methods on the INRIA person dataset at a $10^{-4}$ false positive rate. ECO features with speciation is as good as or better than other state-of-the-art methods on the INRIA person dataset. Speciation allows ECO features to better compete with other state-of-the-art methods while still maintaining the ability of ECO features to adapt themselves to different target objects.

Figure 7.14: Comparison between Histogram of Oriented Gradients, ECO features, and ECO features with speciation on the INRIA person dataset.
Table 7.1: Comparison to state-of-the-art methods on the INRIA person dataset.

<table>
<thead>
<tr>
<th>Method</th>
<th>Miss rate at $10^{-4}$ false positive rate</th>
</tr>
</thead>
<tbody>
<tr>
<td>HOG</td>
<td>12%</td>
</tr>
<tr>
<td>ECO Features [26]</td>
<td>8%</td>
</tr>
<tr>
<td>Dollár [37]</td>
<td>7%</td>
</tr>
<tr>
<td>Tuzel [91]</td>
<td>7%</td>
</tr>
<tr>
<td>Dollár [94]</td>
<td>4%</td>
</tr>
<tr>
<td>Schwartz [90]</td>
<td>3%</td>
</tr>
<tr>
<td>ECO w/Speciation</td>
<td>3%</td>
</tr>
</tbody>
</table>

7.3 ECO Feature Length

The number of transforms used to create an ECO feature, $n$, varies from 2 to 8 transforms. This decision was made to allow ECO features to be long and complicated if necessary but not so long that so much time was spent on complicated features that are unlikely to yield good results. To test the validity of this decision several Adaboost models were trained on the INRIA person dataset where the range of allowable ECO feature lengths were varied. A comparison of the accuracy of these models is given in Figure 7.14. Using only two transforms has a higher miss rate at all false positive rates and a significantly higher miss rate at low false positive rates. Including ECO features with three transforms gives a significant lower miss rate at low false positive rates. Adding more ECO features that are allowed to be longer helps a little until accuracy start to decrease slightly after allowing ECO features with more than seven transforms. An obvious advantage of using shorter ECO features is computation times. The testing times on the INRIA person dataset ranged from five minutes to nine minutes, on our workstation using an AMD Phenom 1055T, depending on the range of lengths included in the model.

7.4 Subregions

ECO features were designed to use subregions to allow each ECO feature to specialize in identifying a different aspect of the target object. Further testing has shown that the use of subregions does not improve nor degrade the accuracy of ECO features. The use of subregions does, however, provide a 1.75 speedup testing on the INRIA Person dataset. Subregions tend to be fairly large with an average area approximately equal to 35% of the
Figure 7.15: Accuracy of several Adaboost models where the range of the length of the ECO features used by the models is increased.

entire image area. For example, over the INRIA person dataset where the full size images are $64 \times 128$, the average subregion size is $37 \times 79$. Figure 7.15 shows the comparison on the INRIA person dataset with subregions turned on and off.

7.5 Using a Support Vector Machine

A support vector machine [111] (SVM) is a popular machine learning algorithm that could be used to classify the outputs of the perceptrons associated with each ECO feature. AdaBoost was chosen to combine the perceptrons after the ECO features were trained because AdaBoost is typically used to combine weak classifiers. Also built into the AdaBoost algorithm is the ability to choose a subset of weak classifiers. An SVM was used to see if using AdaBoost was a good design decision.

SVMs construct a hyperplane or a set of hyperplanes, possibly in a higher dimensional space, to make the data more linearly separable. The more linearly separable the data is, the easier it is to classify. SVMs have several important parameters that affect their classification accuracy that are data specific. One of the most important choices is the kernel used to make
Figure 7.16: Accuracy of two Adaboost models where subregions were used and where they were not used on the INRIA person dataset.

the data linearly separable. The success of a kernel is data dependent and sometimes kernels are developed for specific applications [112, 113]. There are several kernels included in the LIBSVM [114] library that was used in our experiments and each kernel was tested. A reasonable effort was also made to find the other SVM parameters that worked best for our experiments.

No SVM model, however, was able to achieve better than a 12% miss rate at $10^{-4}$ false positive rate. In comparison, AdaBoost was able to achieve a 3% miss rate.

7.6 Important Transforms

To test the importance of any given transform, AdaBoost models were made for the various datasets where each transform was excluded in turn. While a very time consuming process, it was hoped that certain transforms could be identified as being important for certain datasets. However, no individual transforms were identified as being particularly
important. The accuracy would vary to a small degree because a new model was trained and the random nature of how models are built, but no appreciable decrease in accuracy was observed by removing any transform. This indicates that the transforms provide redundant information or, at least, equally significant information. The histogram of oriented gradients transform and the color transform were removed from all the tests performed in this work because they did not improve results and they significantly slowed the computation time of the ECO features algorithm.

7.7 Histograms of Oriented Gradients

In the introduction in Chapter 1 a discussion is given about the Histograms of Oriented Gradients (HOG) transform. The HOG transform is a very popular transform used for object recognition. Due to its popularity in object recognition it was hoped that the addition of the HOG transform would improve the accuracy of ECO features on the datasets used for testing. The other transforms in the ECO feature algorithm are considered more basic transforms and the addition of HOG would help determine how more high level transforms could perform. For information about the implementation of HOG in the ECO features algorithm see Section A.28 in Appendix A. The addition did not give better results and slowed the method considerably. In none of the tests reported in this work was the HOG transform used because of this.

7.8 Commentary on Performance

When one stops to think about the number of image transform combinations, their parameters, and the possible locations on the image to apply the transforms — the search space seems dizzyingly large. Surprisingly, however, for all experiments run so far, good ECO features have been found very quickly. Even after a few generations of the genetic algorithm, weak classifiers with good discriminative power are found. To help with run times, the image transform functions from OpenCV are used, which are optimized for fast performance.

Most of the training was done on an AMD Phenom II 920 quad-core running at 2.8 GHz, with four gigabytes of RAM. The Caltech datasets took approximately 20 minutes each to train. The volcanoes on Venus dataset, which consists of 20 different training sets,
took approximately two hours to train altogether. The INRIA Person dataset consisted of a far greater number of images and trained in about two hours using a cluster of three Dell PowerEdge M610 with two Xeon Quad-core X5560 processors. Although these training times are not critical, as training is intended to be an off-line process, they demonstrate that training time is not exorbitant nor impractical. It should be noted that as training images are added, or as the number of generations or size of population in the genetic algorithm increases, the training times scale linearly.

```c
int main(int argc, char *argv[]) {
    const int N = 100000;
    int i, a[N];

    #pragma omp parallel for
    for (i = 0; i < N; i++)
        a[i] = 2 * i;

    return 0;
}
```

**Figure 7.17:** A simple example of parallelizing a `for` loop using OpenMP.

To take advantage of multiple cores, the inner loop of Algorithm 1, where over 99% of the computation is done, was parallelized using OpenMP. OpenMP is a library that provides multi-threading with the use of compiler directives and library routines. OpenMP has a construct for parallelizing for loops, and in this situation, one simple compiler directive was all that was needed to take advantage of all the processors and cores on our machines. Figure 7.16 shows a simple example of using OpenMP to parallelize a `for` loop. The variable `a` is shared among all threads while each thread gets a private copy of the `for` loop variable `i`. 
Chapter 8

Conclusion

8.1 Discussion

A presentation of the Evolution-COnstructed (ECO) features algorithm and implementation details was given. ECO features were shown to be very effective at general object recognition. On all the datasets that were used, ECO feature either were better than the state-of-the-art or competed very well. The ECO features method is the first feature construction method where the features were completely constructed from scratch and that competes well with features constructed by human experts at general object recognition.

A closer look at ECO features shows that ECO features found shape and texture information in the training examples of the object. The information was specific to the target object and in some instances the context of the target object. For example, planes are generally found in the sky and motorbikes on the road. The ECO features for each dataset were varied and tuned to different aspects of the target object except on the volcanoes on Venus dataset where the amount of information to exploit is very limited and many of the ECO features were very similar.

Despite the large search space the genetic algorithm is able to quickly find many solutions that work quite well. Speciation allowed ECO features to compete within niches rather than against a large population improving the diversity and results of the ECO features. No ECO feature transform was found to be vital to accuracy on any of the datasets, indicating that the set of transforms provide redundant information or, at least, equally significant information.

ECO features provide many benefits for object recognition and compete well with state-of-the-art methods. We believe, however, that future work in automated feature construction will lead to better results in object recognition and that one day, features found
using feature construction methods will be far better suited to the task of object recognition than those created by human experts.

8.2 Future Work

While the ECO features algorithm currently is the best feature construction method for general object recognition, there are many improvements that could be made. Beyond what is listed here it is hoped that, based on this work, other intelligent researchers will expand the field of feature construction with their own ideas.

The use of a genetic algorithm has a few disadvantages and speciation was added to help deal with this. There are several alternatives that could be used, including evolution strategies, evolutionary programming, simulated annealing, Gaussian adaptation, hill climbing, and particle swarm optimization. It is unclear whether any one of these methods would improve the results, since each method has its advantages, but it would be interesting to look at.

The number of inputs to the perceptrons is fairly large. In many cases the number of inputs to the perceptrons is equal to the number of pixels in the subregion that was selected. With hundreds or thousands of inputs, the perceptron can suffer from Hughes phenomenon, where there are not enough training samples to ensure that there are several examples for each combination of inputs. A higher dimensional feature space will lead to reduced predictive abilities given the same training set. A feature selection or feature extraction step could be added to reduce the dimensionality of the input space.

There are several things that could be done to improve the runtime performance of ECO features. OpenCV was used extensively throughout this work and many OpenCV functions now also have a GPU implementation. In the same way that we saw a significant improvement in run times by implementing the HOG method on a GPU, a GPU implementation of ECO features could help dramatically. Avoiding a sliding window approach as discussed in Section 5.5 could greatly reduce the number of windows tested on full images.
8.3 Summary of Contributions

1. The ECO features algorithm is the state-of-the-art in feature construction. It is the first algorithm that competes with features sets designed by human experts at general object recognition.

2. ECO features can operate on a large variety of image types and could even operate on non-image signals given additional transform types.

3. ECO features allows non-expert researchers to create accurate classifiers for objects of interest to them.

4. ECO features were tested on a large number of datasets to show their abilities on a wide range of problems.

5. Created visualization of ECO features that show what ECO features learned on specific datasets.

6. Made an attempt to find the most important transforms for different datasets, but found that there were no specific transforms that were needed for any given dataset.

7. Analyzed the use of subregions.

8. Analyzed the number of transforms allowed in a single ECO feature.

9. Added other higher level features that did not improve the accuracy of ECO features.

10. The ECO feature algorithm was improved by adding speciation.

11. A support vector machine was used in place of AdaBoost as a classifier and found not to be as accurate in this application.
Bibliography


88


[71] F. Suard, A. Rakotomamonjy, A. Bensrhair, and A. Broggi, “Pedestrian detection using infrared images and histograms of oriented gradients.” Intelligent Vehicles Symposium, vol. 9, no. 5, p. 12, June 2006. 9


Appendix A

ECO Feature Image Transforms

A.1 Gabor Transform

The Gabor Transform is a linear filter named after Dennis Gabor. Jones and Palmer concluded that the Gabor filter provides a reasonably accurate description of most spatial aspects of simple receptive fields in the visual cortex [80]. The Gabor filter is applied by convolving the image with the filter. The filter is created using the following equations:

\[
g(x, y; \lambda, \theta, \psi, \sigma, \gamma) = e^{-\left(\frac{x'^2 + y'^2}{2\sigma^2}\right)} \cos \left(2\pi \frac{x'}{\lambda} + \psi\right),
\]

\[
x' = x \cos(\theta) + y \sin(\theta),
\]

\[
y' = -x \sin(\theta) + y \cos(\theta).
\]

The Gabor transform has six parameters that can be tuned by the genetic algorithm.

1. \(\lambda\) is the wavelength of the cosine factor of the Gabor filter.
2. \(\theta\) is the orientation of the Gabor filter.
3. \(\psi\) is the phase offset of the Gabor filter.
4. \(\sigma\) is the sigma of the Gaussian envelope of the Gabor filter.
5. \(\gamma\) is the spatial aspect ratio of the Gabor filter.
6. The size of the filter is another parameter. The filter is always square. Parameters \(x\) and \(y\) range from 0 to the size of the filter.

A.2 Sobel Transform

The Sobel transform, normally referred to as the Sobel operator, is generally used for edge detection. The Sobel transform is applied by convolving the filter with the image. The Sobel transform is a separable filter allowing a 2-D convolution to be done as two 1-D convolutions which reduces computations.

The Sobel transform has four parameters, three of which can be tuned by the genetic algorithm.

1. The depth of the destination image, meaning the destination image can have 8/16/32 bits per pixel or a single/double precision floating point value. Once chosen at creation
time this value cannot be changed by the genetic algorithm as it could cause future transforms to fail.

2. The size of the Sobel kernel. OpenCV allows for kernel sizes of 1, 2, 5, or 7.

3. The order of the derivative in the x direction.

4. The order of the derivative in the y direction.

A.3 Erode Transform

The erode transform is a mathematical morphological operation that erodes the boundaries of foreground pixels of a binary image. Erosion can also be applied to greyscale images where small bright spots are darkened and very small bright areas might be completely removed. As is generally done, a $3 \times 3$ structuring element was always used and could not be modified by the genetic algorithm.

The erode transform has one parameter that can be tuned by the genetic algorithm.

1. The number of iterations the image should be eroded.

A.4 Dilate Transform

The dilate transform is a mathematical morphological operation that expands the boundaries of foreground pixel of a binary image and is the exact opposite of the erode transform. Again, a $3 \times 3$ structuring element was always used and could not be modified by the genetic algorithm.

The dilate transform has one parameter that can be tuned by the genetic algorithm.

1. The number of iterations the image should be dilated.

A.5 Gaussian Transform

A Gaussian transform consists of convolving the input image with a truncated Gaussian function. The Gaussian function is truncated since it technically requires an infinite window length, but since the function decays rapidly, truncation is a good approximation of the entire Gaussian function. The Gaussian transform is also a separable transform so it can be processed as the sequence of two 1-D filters.

The Gaussian transform has one parameter that can be tuned by the genetic algorithm.

1. The size of the Gaussian filter. The filter is always square.

A.6 Adaptive Threshold Transform

The adaptive threshold transform converts an 8 bit greyscale image into a binary image. The destination image is thresholded according to the following equations depending on which type of thresholding is done. The threshold $T(x, y)$ is either the mean of the pixels in a window around the current pixel or a weighted sum of the window by using a cross-correlation of a Gaussian and the window.
\[ dst(x, y) = \begin{cases} \text{maxValue} & : \text{if } src(x, y) > T(x, y) \\ 0 & : \text{otherwise.} \end{cases} \] \tag{A.2}

\[ dst(x, y) = \begin{cases} 0 & : \text{if } src(x, y) > T(x, y) \\ \text{maxValue} & : \text{otherwise.} \end{cases} \] \tag{A.3}

The adaptive threshold transform has three parameters that can be tuned by the genetic algorithm.

1. The type of thresholding to perform, either setting the pixel to \text{maxValue} or 0 if it above the threshold value.

2. The type of threshold to use, either the mean value in the window around the pixel, or the weighted sum of the window.

3. The size of the window to use. The window is always square and the width is an odd number.

### A.7 Hough Line Transform

A Hough line transform is used to identify lines in an image. Generally a Hough line transform is performed after a edge detection transform, but when using ECO features this may not have been performed in a previous transform. The Hough line transform builds an accumulator array where each cell is a pixel representing a line defined by an angle and radius. Every cell/line is incremented by one for every non-zero pixel in the image that it intersects with. The higher the count in a cell, the more probable it is that the line exists with those parameters in the image. The OpenCV library code had to be modified so that it stopped once all the counts in the accumulator array were done. The accumulator array can either be the input to another transform or the inputs to the perceptron.

The Hough line transform has two parameters that can be tuned by the genetic algorithm, both of which define what resolution should be used to look for lines.

1. Number of angles to use.

2. Number of distinct radii.

### A.8 Histogram Transform

The histogram transform creates a 1-D histogram of the input which may or may not be an image. The result has as many channels as the input image with every channel being treated separately.

The histogram transform has one parameter that can be tuned by the genetic algorithm.

1. Number of bin to use.
A.9 Harris Corner Strength Transform

The Harris corner strength transform converts every pixel of the image to be the strength of a corner at that location. Corners can be viewed as the local maximums. It uses the Sobel operator first for edge detection, computes a $2 \times 2$ gradient covariation matrix ($M^{(x,y)}$) and then uses Equation A.4 to compute the corner strength.

$$dst(x,y) = |M^{(x,y)}| - k(trM^{(x,y)})^2.$$ (A.4)

The Harris corner strength transform has three parameters that can be tuned by the genetic algorithm.

1. The Harris detector free parameter from Equation A.4.
2. The size of the aperture for the Sobel operator.
3. The neighborhood block size to use around each pixel.

A.10 Normalize Transform

The normalize transform normalizes the input so that Equation A.5 is satisfied. The parameter $\alpha$ can be either 1 or 256 and $p$ can be $\infty$, 1, or 2.

$$||dst||_{L_p} = \alpha.$$ (A.5)

The normalize transform has three parameters that can be tuned by the genetic algorithm.

1. What kind of norm to perform; $L_\infty, L_1, L_2$.
2. The value of $\alpha$; 256 or 1.
3. The depth of the output image. This parameter changes with $\alpha$ and will produce a single precision floating point output when $\alpha = 1$ or 256 when $\alpha = 256$.

A.11 Histogram Equalization Transform

The histogram equalization transform alters the histogram of an image in an attempt to increase the contrast of the image. The histogram of the image is computed, the histogram is normalized so that the sum of all histogram bins is 255, and the integral of the histogram is computed as in Equation A.6. The image is then transformed using $H'$ as a look-up table shown in Equation A.7.

$$H'_i = \sum_{0 \leq j < i} H(j).$$ (A.6)

$$dst(x, y) = H'(src(x, y)).$$ (A.7)

The histogram equalization transform has no parameters.
A.12 Convert Type Transform

The convert transform is a transform for convenience in the ECO feature algorithm. Many transforms expect the input to be of a certain data type. The convert transform converts from one data type to another. The convert transform does not take into consideration the data type of the input and in some instances does no work when the input and output data types are the same.

The convert transform has one parameter that can be tuned by the genetic algorithm.

1. Data type to convert to.

A.13 Discrete Fourier Transform

The discrete Fourier transform can perform a forward or inverse Fourier transform. The discrete Fourier transform converts a signal, the input image, to the frequency domain. The input is treated as $M \times N$ elements and a 2D Fourier transform is performed according to Equation A.8.

$$dst = e^{-\frac{2\pi ijk}{N}} \cdot src \cdot e^{-\frac{2\pi ijk}{M}}.$$  (A.8)

The discrete Fourier transform has one parameter that can be tuned by the genetic algorithm.

1. Flags that indicate whether to do a forward or inverse transform, whether to scale the results by dividing by the number of array elements, and then whether to perform the transform on every individual row of the input or not.

A.14 Log Transform

The log transform is a very simple transform that takes the natural log of every element of the input.

There are no parameters for the log transform.

A.15 Square Root Transform

The square root transform is another very simple transform that takes the square root of every element of the input.

There are no parameters for the square root transform.

A.16 Median Blur Transform

The median blur transform smooths an image using a median filter. The filter takes a neighborhood around a pixel and sets the pixel to the median of the neighborhood.

There is one parameter for the median blur transform that can be tuned by the genetic algorithm.

1. The size of filter to use. The filter size is always odd and square.
A.17 Canny Transform

The Canny transform is used for edge detection and is able to find a wide range of edges by using a multi-stage algorithm. The Canny transform uses the Sobel and the Gaussian transform as stages of the transform.

The Canny transform has four parameters that can be tuned by the genetic algorithm.

1. The first threshold that is used for linking edges.
2. The second threshold that is used for linking edges.
3. Size of Sobel transform to use.
4. Whether to use the $L_2$ or $L_1$ norm.

A.18 Distance Transform

The distance transform finds the distance to the closest black pixel for each pixel and is computed according to the algorithm by Borgefors [115]. The output has a floating point data type.

The distance transform has two parameters that can be tuned by the genetic algorithm.

1. Distances are computed as a sum of basic distances that can be horizontal, vertical, and diagonal shifts. There is a cost $a$ for horizontal and vertical shifts and another cost $b$ for diagonal shifts. Three distance types can be chosen that modify the costs $a$ and $b$.
2. The mask size used for computing distances.

A.19 Integral Image Transform

An integral image is an image where each pixel is the sum of all pixels above and to the left of it as in Equation A.9.

$$dst(X,Y) = \sum_{x<X,y<Y} src(x,y).$$

(A.9)

The integral image transform has one parameter that can be tuned by the genetic algorithm.

1. The data type of the output.

A.20 Laplacian Transform

The Laplacian transform is the addition of the second x and y gradient calculated by the Sobel transform as in equation A.10.

$$dst = \frac{\delta^2 src}{\delta x^2} + \frac{\delta^2 src}{\delta y^2}.$$  

(A.10)

101
The Laplacian transform has one parameter that can be tuned by the genetic algorithm.

1. The kernel size to use when computing the second derivatives. It is always odd and square.

A.21 Hough Circles Transform

The Hough circle transform is very similar to the Hough lines transform explained in A.7. A generalized Hough transform also exists [116] to detect arbitrary shapes. The Hough circle transform builds an accumulator array where every cell represents a circle defined by a given radius and angle. Every cell/circle is incremented by one for every non-zero pixel in the image that it intersects with. The higher the count of a cell, the more probable it is that a circle exists with those parameters in the image. The OpenCV library code had to be modified so it stopped when all the counts in the accumulator array were done. The accumulator array can either be the input to another transform or the inputs to the perceptron.

The Hough circle transform has two parameters that can be tuned by the genetic algorithm both of which define what resolution should be used to look for circles.

1. Number of angles to use.
2. Number of distinct radii.

A.22 Difference of Gaussians Transform

The difference of Gaussians transform is a method to discard all but a small range of spatial frequencies that appear in a greyscale image. The method works by subtracting a Gaussian blurred image from another Gaussian blurred image. A Gaussian blur suppresses high-frequency spatial information, and by subtracting two Gaussian blurred images only the spatial information that lies between the frequencies is preserved. The amount of blur is determined by the size of the Gaussian kernel.

The difference of Gaussians transform has two parameters that can be tuned by the genetic algorithm.

1. The kernel size for the first Gaussian blur.
2. The kernel size for the second Gaussian blur.

A.23 Rank Transform

The rank transform is a form of non-parametric local transform introduced by Zabih and Woodfill [117]. It ranks a pixel's intensity value against the neighboring pixels. For example, if a pixel has the lowest pixel intensity in a neighborhood, it would have a value of 0, and if it was the highest pixel intensity it would have a value equal to the number of pixels in the neighborhood. The method is robust to variations caused by the camera gain and some lighting conditions. Downsides of the rank transform include the loss of intensity information and the fact that the same rank can be produced from a variety of patterns.
The rank transform is not found in the OpenCV library and had to be coded by hand. It is coded to only do a $3 \times 3$ neighborhood.

The rank transform has no parameters.

### A.24 Census Transform

The census transform is also a non-parametric local transform introduced by Zabih and Woodfill [117]. The census transform records as a bitstring the boolean comparison of a pixel's intensity to each pixel in the neighborhood. For example, if a pixel has a lower intensity than any pixel in a $3 \times 3$ neighborhood then its binary value becomes 00000000 but if it has the highest intensity value in the neighborhood its binary value becomes 11111111. This method is also robust to variations caused by camera gain and lighting conditions and unlike the rank transform encodes information about the local spatial structure. This transform is also not found in the OpenCV library and was hand coded using a $3 \times 3$ neighborhood.

The census transform has no parameters.

### A.25 Gradient Transform

The gradient transform can perform various types of gradients: Prewitt in the x direction given in Equation A.11, Prewitt in the y direction given in Equation A.12, Kirsch compass in the x direction given in Equation A.13, Kirsch compass in the y direction given in Equation A.14, Kirsch compass diagonal given in Equation A.15, a simple x direction gradient given in Equation A.16, and a simple y direction gradient given in Equation A.17.

\[
\begin{bmatrix}
-1 & 0 & 1 \\
-1 & 0 & 1 \\
-1 & 0 & 1 \\
\end{bmatrix}.
\] (A.11)

\[
\begin{bmatrix}
-1 & -1 & -1 \\
0 & 0 & 0 \\
1 & 1 & 1 \\
\end{bmatrix}.
\] (A.12)

\[
\begin{bmatrix}
-5 & 3 & 3 \\
-5 & 0 & 3 \\
-5 & 3 & 3 \\
\end{bmatrix}.
\] (A.13)

\[
\begin{bmatrix}
3 & 3 & 3 \\
3 & 0 & 3 \\
-5 & -5 & -5 \\
\end{bmatrix}.
\] (A.14)

\[
\begin{bmatrix}
3 & 3 & 3 \\
-5 & 0 & 3 \\
-5 & -5 & 3 \\
\end{bmatrix}.
\] (A.15)

\[
\begin{bmatrix}
-1 & 0 & 1 \\
\end{bmatrix}.
\] (A.16)
\[
\begin{bmatrix}
-1 \\
0 \\
1
\end{bmatrix}.
\] (A.17)

The gradient transform has one parameter that can be tuned by the genetic algorithm.
1. Type of gradient to use.

A.26 Resize Transform

The resize transform scales the input using bilinear interpolation.
The resize transform has one parameter that can be tuned by the genetic algorithm.
1. A scale factor between .1 and .9.

A.27 Pixel Statistics

The pixel statistics transform collects statistics about the image and stores the statistics in a \(11 \times 1\) matrix of floats. The statistics collected are the mean intensity value, the standard deviation of intensity values, minimum intensity value, maximum intensity value, and seven Hu moments [118]. The statistics are collected for each cell in a grid over the subregion that ranges from a \(1 \times 1\) grid to a \(4 \times 4\).

The pixel statistics transform has two parameters that can be tuned by the genetic algorithm.
1. Number of rows in the grid.
2. Number of columns in the grid.

A.28 Histograms of Oriented Gradients Transform

The Histograms of Oriented Gradients (HOG) transform was designed by Dalal and Triggs [59]. It creates a histogram of the orientations of gradients in local spatial regions. The subregion is divided into blocks which is then divided into cells. Each cell has a histogram of the orientation of gradients of pixels covered by that cell. A block stride defines how many blocks are in the subregion and blocks can overlap. OpenCV does implement the HOG transform. This transform was not used for any of the results presented in this paper because of the computation time without improving results.

The HOG transform has six parameters that can be tuned by the genetic algorithm. The parameters are not independent of each other, and checking is done to make sure that parameters work together.
1. The window size is the number of blocks across and down the subregion.
2. The size of a block.
3. The size of a cell.
4. The stride of a block.
5. The number of bins to use in the histogram.
A.29 Color Transform

The color transform attempts to capture the color information contained in the subregion. The transform converts the input image from RGB to another color space which could be YUV, YCrCb, XYZ, Luv, Lab, HSV, or HLS color space. The transform then either returns one of the three channels or resizes the image and then concatenates the channels. Most of the transforms used by OpenCV do not operate on three channels, so the concatenation allows the result to be one channel. The color transform is always the first transform in the ECO feature. This transform was not used on any of the datasets because the increased computation time from having to operate in some cases on three times the amount of data. Although no improvement in results was seen on the datasets presented here it is anticipated that the transform would be useful for some datasets.

The color transform has four parameters that can be tuned by the genetic algorithm.

1. The color space to convert to.
2. The method to use, either return one channel or the three channels scaled and concatenated.
3. If returning just one channel, which channel to use
4. The scale to use if returning 3 channels concatenated in the range .1 to .9.