2015

Novel Texture-based Probabilistic Object Recognition and Tracking Techniques for Food Intake Analysis and Traffic Monitoring

Robert Jacob DiBiano
Louisiana State University and Agricultural and Mechanical College, robertdibiano@gmail.com

NOVEL TEXTURE-BASED PROBABILISTIC OBJECT RECOGNITION AND TRACKING TECHNIQUES FOR FOOD INTAKE ANALYSIS AND TRAFFIC MONITORING

A Dissertation

Submitted to the Graduate Faculty of the
Louisiana State University and
Agricultural and Mechanical College
in partial fulfillment of the
requirements for the degree of
Doctor of Philosophy

in

The Department of Electrical & Computer Engineering

by
Robert J. DiBiano
B.S., Lamar University, 2004
M.S., Lamar University, 2008
December 2015

Acknowledgments

This research was supported by National Institute of Health (NIH) grants R21AG032231 and R01DK089051. The use of their extensive and detailed food image and nutritional database was invaluable to the completion of this work.

I would like to acknowledge the staff at Pennington Biomedical Research Laboratory for their advice and expert knowledge in the area of food nutritional information and also for providing their "free living" food data. The content is solely the responsibility of the authors and does not necessarily represent the official views of the National Institute of Health.

This project is supported by Army Research Office (ARO) under Grant #W911-NF1010495. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the ARO or the United States Government.

I'd also like to thank all the members of my committee, Supratik Mukhopadhyay, Xin Li, Jerry L. Trahan, Corby Martin, Bahadir Gunturk, and Gerald M. Knapp, as well as my sister and the other members of my research group: Jerry Weltman, Malcolm Stagg, Saikat Basu, and Manohar Karki for their time, encouragement, advice, and support.

Table of Contents

ACKNOWLEDGMENTS ii

LIST OF TABLES vi

LIST OF FIGURES vii

ABSTRACT x

CHAPTER

1 INTRODUCTION 1
- 1.1 Motivation 1
  - 1.1.1 Food Recognition and Portion Size Estimation 1
  - 1.1.2 Car and Human Tracking 2
- 1.2 Contributions to the Dissertation 3
- 1.3 Overview 5

2 RELATED WORK 7
- 2.1 Food Analysis 7
  - 2.1.1 Agricultural Analysis 7
  - 2.1.2 Meal Analysis 9
  - 2.1.3 Discussion 14
- 2.2 Car and Human Tracking 14
  - 2.2.1 Optical Flow 15
  - 2.2.2 The Drift Problem 15
  - 2.2.3 Early Tracking, the Kalman Filter 16
  - 2.2.4 Tracking vs. Detection and Tracking by Detection 16
  - 2.2.5 Enhanced Template Matching 16
  - 2.2.6 Enhanced Optical Flow 17
  - 2.2.7 Enhanced Difference Images 18
  - 2.2.8 Layer Representations 18
  - 2.2.9 Advanced Filtering 19
  - 2.2.10 Machine Learning Based Tracking 19
  - 2.2.11 Part Based Models 22
  - 2.2.12 Non-Rigid Objects 22
  - 2.2.13 Limb Tracking 23
  - 2.2.14 Surveillance 23
  - 2.2.15 Discussion 24

3 PRELIMINARIES 26
- 3.1 Graph cuts segmentation and max-flow min-cut theorem 26
  - 3.1.1 Segmentation as Energy Minimization [28] 26
  - 3.1.2 Energy minimization as Known Graph Problem [28] 28

3.1.3 Graph Cuts 29
3.1.4 Muticlass Graph Cuts and Efficient Approximations 30
3.1.5 GrabCuts [149] 31
3.1.6 Discussion 34
3.2 Texture Analysis 34
3.2.1 Statistical 35
3.2.2 Structural 38
3.2.3 Transform Based 39
3.2.4 Model Based Approaches 43

4 FOOD IMAGE ANALYSIS FOR MEASURING FOOD INTAKE IN FREE LIVING CONDITIONS 46
4.1 Introduction 46
4.2 Contributions 47
4.3 Methods and Algorithms 47
4.3.1 Preprocessing 48
4.3.2 Segmentation 51
4.3.3 Classification 52
4.3.4 Volume Estimation 59
4.4 Results 64
4.5 Discussion 66
4.5.1 Main sources of error 66
4.5.2 Interesting Cases 66
4.5.3 "One versus the rest" advantages and features 67

5 AN AGILE FRAMEWORK FOR REAL-TIME VISUAL TRACKING IN VIDEOS 69
5.1 Introduction 69
5.2 Contributions 69
5.3 Related Work 70
5.4 The Proposed Approach 70
5.4.1 Image Stabilization 70
5.4.2 Track Starting 71
5.4.3 The Agile Tracking Framework 72
5.5 Implementation of our Approach 79
5.6 Results and Comparative Studies 79
5.7 Conclusions 79

6 MAPTRACK: A PROBABILISTIC REAL TIME TRACKING FRAMEWORK BY INTEGRATING MOTION, APPEARANCE AND POSITION MODELS 81
6.1 Introduction 81
6.2 Contributions 81
6.3 Related Work 82
6.4 The Proposed Approach 82

6.4.1 Image Stabilization 83
6.4.2 Automated Track Initialization 84
6.4.3 The MAPTrack Framework 84
6.5 Implementation Details 88
6.6 Results and Comparative Studies 88
6.7 Conclusions 89

7 CONCLUSIONS AND FUTURE WORK 92
7.1 Food Analysis 92
7.2 Tracking 92
7.3 Future Work - Food Recognition 94
7.4 Future Work - Tracking 95

REFERENCES 97

VITA 114

List of Tables

3.1 Swap-move edge weights 31

3.2 Expansion-move edge weights 33

4.1 Final feature list. 59

4.2 Volume estimation results. 64

4.3 Classification results. 64

5.1 Comparison of the various trackers - number of frames after which the trackers lost track for the first time 79

5.2 Comparison of the various trackers - number of frames after which the trackers lost track for the first time(contd.) 79

6.1 The different states of the tracked object. 89

6.2 Comparison of single-object trackers in (Kalal et al., 2010) with MAPTrack. Shows the number of frames after which the trackers lost track for the first time 90

6.3 Tracker results for TUD (Andriluka et al., 2008). 91

6.4 Results from the tracker (Metric used as in (Smith et al., 2005) [160]). Scores for configuration distance, multiple objects, multiple tracks, false positives, and tracker purity are defined. 91

List of Figures

1.1 Apps like myfitnesspal require manual data entry. 2

1.2 Volume can be estimated from 2d digital images. 2

1.3 Low quality aerial car and human videos. 3

3.1 E_{data} measures similarity of each pixel to some model; more similar costs less energy to label. 27

3.2 E_{smooth} measures similarity of each pixel to is neighbors; each differently labeled neighbor costs energy. 28

3.3 Node for each class and pixel, higher energy costs convert to lower edge weights. 29

3.4 Find the minimum cost cut to divide the graph into 2 subgraphs, each containing one class node. 30

3.5 Swap-move 31

3.6 Expansion-Move 32

3.7 Small number of user provided seed pixels for initial conditions. 32

3.8 left:very rough user provided input, right: accurate results 33

3.9 First order statistics can be derived from a histogram. 36

3.10 Gray Level Run Length Matrix (right). 36

3.11 Gray Level Run Length Vector. 36

3.12 Gray Level Co-Occurrence Matrix (right). 37

3.13 Neighboring Gray Level Dependence Matrix. 37

3.14 Angle Measurement Technique. 38

3.15 Extracting textural primitives from an image. Primitive frequency and clustering characterize texture. 38

3.16 Convolution of image “object” by Gaussian mask “psf”. 39

3.17 lena.png (left), horizontal Sobel filtered (middle), and vertical Sobel filtered (right). 40

3.18 Eigenfilters computed from the ORL Face database(Eigenfaces.png, Source: Ylebru - Wikimedia Commons). 40

3.19 All periodic functions can be approximated by the sum of a series of sine and cosine functions. 41

3.20 Frequency Spectrum. 41

3.21 Discrete Cosine Transform Filters. 42

3.22 Fast Fourier Transform Results. 42

3.23 Gabor Filters. 43

3.24 A Meyr Wavelet. 43

3.25 A simple Markov chain. 44

3.26 A Markov Mesh. 45

4.1 High level block diagram of automated food photo analysis. 47

4.2 Feature rankings. 49

4.3 Yellow fluorescent light spectrum.png(left), Spectrum of blue sky.png(right) (Source: Deglr6328 - Wikimedia Commons) 50

4.4 Segmentation. 53

4.5 Feature vector generation. 55

4.6 Classification results. 55

4.7 OVTR Classification. 56

4.8 OVTR Training. 56

4.9 Feature rankings. 58

4.10 left: raw image, middle: perspective transform, right: standard with known weight 61

4.11 left: raw image, middle: affine transform, right: standard with known weight 62

4.12 Volume estimation on solid foods. 62

4.13 Best linearity. 63

4.14 Worst linearity. 63

4.15 Volume estimation on liquid foods. The two different curves for bowl A and B are clearly visible 63

4.16 Classification results. 65

4.17 Detailed testing results. 65

4.18 Lighting differences. 66

4.19 Multimode foods. 67

4.20 Multiple types/textures (top left: oatmeal, top right: garlic toast, bottom left: cream of wheat.) 67

5.1 Schematic representation of our approach. 70

5.2 Confidence value update for the frames (for increasing confidence) 72

5.3 Results from the agile tracker. 80

5.4 The left one represents the output from the agile tracker and the right one represents that from TLD, which has trouble with nearly identical objects. 80

6.1 Schematic representation of our approach. 83

6.2 a) Image b) Motion Pixels c) Appearance Pixels d) Projected Position Pixels 86

6.3 Foreground and Background Color Histograms of the two cars. 87

6.4 Output from MAPTrack (Left) and TLD (right). TLD switches randomly between similar objects in noisy videos. 89

6.5 MAPTrack results for TUD videos. 89

6.6 Image of people and cars, the images are the ROI images, followed by MCP, CP, Velocity Image and the Weighted Composite Image from top to bottom. 90

6.7 ROC curve for the tracker. 91

6.8 Results from MAPTrack. 91

7.1 Confusion matrix. 93

Abstract

More complex image understanding algorithms are increasingly practical in a host of emerging applications. Object tracking has value in surveillance and data farming; and object recognition has applications in surveillance, data management, and industrial automation. In this work we introduce an object recognition application in automated nutritional intake analysis and a tracking application intended for surveillance in low quality videos. Automated food recognition is useful for personal health applications as well as nutritional studies used to improve public health or inform lawmakers. We introduce a complete, end-to-end system for automated food intake measurement. Images taken by a digital camera are analyzed, plates and food are located, food type is determined by neural network, distance and angle of food is determined and 3D volume estimated, the results are cross referenced with a nutritional database, and before and after meal photos are compared to determine nutritional intake. We compare against contemporary systems and provide detailed experimental results of our system's performance. Our tracking systems consider the problem of car and human tracking on potentially very low quality surveillance videos, from fixed camera or high flying Unmanned Aerial Vehicle (UAV). Our agile framework switches among different simple trackers to find the most applicable tracker based on the object and video properties. Our MAPTrack is an evolution of the agile tracker that uses soft switching to optimize between multiple pertinent trackers, and tracks objects based on motion, appearance, and positional data. In both cases we provide comparisons against trackers intended for similar applications i.e., trackers that stress robustness in bad conditions, with competitive results.

Chapter 1
Introduction

1.1 Motivation

Computers are smaller, more powerful, and cheaper than ever before, making them practical for an increasing number of applications. So more complex image understanding algorithms are increasingly practical in a host of emerging applications. Computer vision is used in manufacturing and food processing to differentiate and characterize objects. Vision systems also have tracking and security applications. And most significantly, databases accessible from the Internet contain massive amounts of image and video data, and are being continually added to. Hence better automated analysis of images and video is newly possible, and has nearly limitless applications. In this work we introduce an object recognition application in automated nutritional intake analysis and a tracking application intended for surveillance in low quality videos.

While the 2 problems may seem different at first glance, diverse machine vision problems utilize similar knowledge and algorithms. Background noise and predictable errors must be accounted for in preprocessing. This may involve accounting for lighting differences or blurring in a single image, or camera movements in a video. Objects of interest must be segmented apart from the background in both cases to enable analysis. This may be done by color, texture, edge, or motion data. And finally objects of interest must be analyzed. In the case of food recognition this means identification, and estimating the position and volume of the object. For video tracking this means identification and estimating the current and future position of the object. The domain specific constraints and best features to use vary by application, but these ideas of image restoration, segmentation, feature extraction, and feature matching apply almost universally in machine vision.

1.1.1 Food Recognition and Portion Size Estimation

Food recognition has been applied in the past in an agricultural setting to assist robotic picking, grade by ripeness/size/color, and locate surface defects/bruising/mold [48]. There have also been applications in grain sorting [7], and weed control [32]. Research on food recognition as meal recognition is a very recent evolvement.

An AI-hard or AI-complete problem is one that requires complex and nuanced judgments about the world, such that solving it would be synonymous with solving the general problem of AI. General purpose food recognition is an AI-hard problem, which can never be completely solved from image data alone. Most glaringly, the composition of a food cannot always be accurately determined by looking at its surface. For example breaded and fried foods all look similar. Even with context taken from nearby foods and the image background, and intuitive guesses taken from the general shape of the breaded blobs, human observers can generally not distinguish between different breaded foods. Composition may vary widely within a single food type, or food with a similar composition may have widely variable appearances. An actual instance of food may legitimately belong to multiple types, and one type may overlap with or be a subset of another.

Automated food recognition systems have practical use for personal health applications, to monitor food intake and provide dietary reports and statistics. Automatic food

intake estimation cross referenced with nutritional database info can provide immediate feedback on dietary deficiencies, and the information could be used to automatically make suggestions on how to correct the imbalance. Manual dietary monitoring apps (ex. myfitnesspal.com in Figure 1.1) require detailed data entry after every meal; and even with these limitations such applications are widely used. The process is streamlined as much as possible; but the user must manually pick every food type off a list and do portion size estimation at each and every meal.

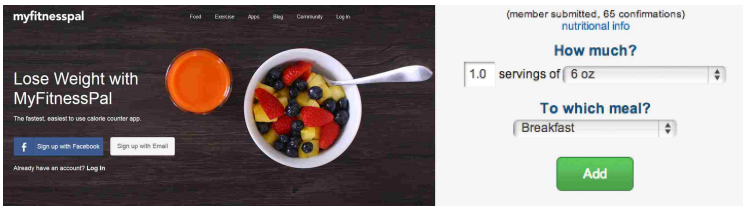

Figure 1.1: Apps like myfitnesspal require manual data entry.

Automated food recognition and portion size estimation is also useful for data collection for health studies such as [122]. Indeed this data collection is already being done manually [190], requiring training and many man-hours spent by human graders on grinding, repetitive work. Nutritional studies are useful to provide statistics that help improve public health, help the military efficiently design nutritious meals, and impact governmental laws and regulations applying to food.

The majority of existing food recognition schemes are limited to distinguishing between a handful of foods, and therefore of little practical value for these applications. A good system should work on large if not open ended numbers of foods and provide a sorted list of most likely candidates, similar to how spelling correction algorithms work. Because of the AI-hard nature of the problem a food recognition system should be supplemented with inferences, expert knowledge, and metadata wherever possible.

Portion size estimation is also a difficult problem. Human experts need practice to become accurate, and even then the error is around 10 percent or more for some food types [190]. This is not an AI-hard problem though, and computers are better suited for this type of calculation than humans. Still, with only single 2d images as input(Figure 1.2), the problem is nontrivial. Scale and positioning are not immediately apparent. Some foods have a strong relation between visible surface area an volume and some do not. Liquid foods have none at all.

Figure 1.2: Volume can be estimated from 2d digital images.

1.1.2 Car and Human Tracking

Tracking is a challenging problem; input videos are often blurry and or low resolution due to distance, hardware limitations, or compression (Figure 1.3). There is inherent difficulty in recognizing the shape of 3 dimensional objects from 2d data after rotations.

Moving through shadows or variable light level may change the appearance of an object, as could being lit by artificial lighting with various different spectra. The sensors used for most computer vision systems have a very low dynamic range, often reducing image quality. Fluid or articulated objects are often hard to characterize with a template based model. There is the problem of how and when to start tracking an object, and the problem of detecting tracking failures and ending a track. Objects that are partially occluded will change in appearance, and if fully occluded, this should be detected separately from a normal track loss if possible. Lost or occluded tracks need to be linked with later tracks when the object shows up again. Finally, there is the "drift problem": if an object changes in appearance slightly, the tracker needs to adjust its internal model of the object to compensate; over time this can lead to modeling and therefore tracking the wrong object. So there is always a trade off between stability and adaptability.

Figure 1.3: Low quality aerial car and human videos.

Tracking has useful applications in robotics, navigation (by tracking the position of the terrain relative to the observer), Human-machine interface (control a computer with gestures), as well as law enforcement and national security. Being able to automatically track a car or human automatically over many miles and hours from satellite data would be useful for security applications. One of the holy grails of video processing is to be able to characterize and search video content automatically, without relying on nearby text or user generated tags. Automatic tracking information is a necessary input to later layers for activity recognition on track-based activities, and the activities would be amalgamated into searchable attributes, actions, and behaviors.

There are any number of tracking algorithms, and many of them are reasonably good; but tracking is a complex problem and there's still plenty of room for improvement. Each existing tracking method makes implicit assumptions about its application, and has individual strengths and weaknesses, so domain specific trackers are useful. Lately, tracking by recognition has become a dominant trend, but it has weaknesses. Track starting by recognition will only work if you have a representative dataset of targets; so it can't be directly used for general purpose tracking. Many trackers work poorly on fluid or articulated objects, since the size, shape, and appearance of these objects can alter suddenly and drastically. Trackers that fall back heavily on object recognition (to recover lost tracks for example) will have trouble tracking multiple similar objects; a standard situation in car/human tracking.

1.2 Contributions to the Dissertation

This dissertation addresses the problem of identifying and estimating the nutritional value of food via textural analysis on still images, and tracking humans and cars, via

motion, appearance, and positional information in videos. Although the problems are different, some of the methods used overlap, and textural analysis of the background can provide useful context clues for video processing. Many of the methods used can be adapted to other applications.

Measuring the type and amount of food intake of free-living (outside controlled clinical research centers) people is an important task in nutrition research. One practical method, called the Remote Food Photography Method (RFPM) [119], is to provide camera-equipped smartphones to participants, who are trained to take pictures of their foods and send these pictures to the researchers over a wireless network. These pictures can then be analyzed by trained raters to accurately estimate food intake, though the process can be labor intensive. In this dissertation, we describe a computer vision application to estimate food intake from the pictures captured and sent by participants. We describe the application in detail, including segmentation, pattern classification, volume estimation modules, and provide comprehensive experimental results to evaluate its performance.

The food recognition problem is intended to be a complete system, potentially fully automated from beginning to end. i.e., food snapshot in, nutritional intake out. Because of the AI completeness of the problem, it also necessarily allows for manual intervention at any point along the process. Food images are taken before and after meals. The images are scanned and foods are located and identified. Scale and rotational information is used along with information about the foods to estimate volume from visible surface area. The food intake information is then ready to be cross referenced with a nutritional database.

This system makes several contributions. It is the most advanced published complete system of its type. It has significant and more extensive data on 2d food volume estimation than any study of its type, and compares the results against those of trained human analysts. It presents a novel food recognition scheme, which provides multilabel classification on a large number of classes parallelizably, in linear time with number of classes, and generates meaningful runners up. Much of research [118, 125, 144, 145, 205, 207] in the area does not meet these requirements, which are absolutely necessary to solve the problem. To the best of our knowledge, at the time it was published our food recognition scheme was the most advanced and accurate system of its kind that fulfilled these requirements. Our system was published in the Proceedings of the SPIE on Medical Imaging [51].

Rather than being a complete system, the tracking systems (Agile Tracker and MAP-Track) are intended to be part of a larger system; presumably with image understanding steps later in the process; they are specifically intended for car and human tracking in that they make certain assumptions - such as the objects they track can move, are all around the same size, their color is significant but multiple different objects may also look nearly identical, etc. They are intended for automated car and human tracking, and use an automated track starting algorithm to decide what and when to track. The videos to be intended for analysis may be grayscale or color, and will be low very quality with parts of objects missing even in absence of occlusions, can interrupted by static or missing frames, and contain rotations and changes in viewpoint and scale. Specifically, the videos we used to define the problem were drawn from the /acrshortvirat Aerial Dataset and /acrshortvirat Ground Dataset [178].

After track starting and image stabilization, our agile tracking framework switches among several simpler trackers depending on the video's and target's states. If the current

tracker's performance metric falls below some threshold, a new tracker is chosen based on the properties of the video and object being tracked. It can handle near passes of similar objects and distinguish between humans and cars.

The agile tracker makes several contributions. Firstly, our agile tracker provides a domain specific solution applicable to car and human tracking both in poor quality aerial videos and good quality surveillance videos. Although intended to be part of a larger system, it is designed to be a complete tracking system suitable for practical stand alone application in the specified domain. In contrast to similar surveillance trackers, our tracker uses multiple main tracking engine types and can switch between them when performance drops. Secondly, the agile framework provides a novel method of merging multiple trackers to produce better results than any individual in the ensemble, by allowing each tracker to run in the situations it is most applicable to. This relies on an estimate of tracker applicability that the agile framework generates from various video and object metrics. Our agile tracker was published as a workshop in the 39th Annual International Computers, Software & Applications Conference (COMPSAC) [20].

Our improved tracker, MAPTrack, uses a probabilistic scheme to merge motion, appearance, and positional models instead of switching between tracking schemes. The contribution each model makes to the amalgamated model is determined by object and video properties. The motion model distinguishes objects of interest from the background based on their relative motion. The appearance model characterizes an object based on its color composition, or its color composition versus that of the environment. The position model estimates an object's location based on its previous positions, velocity, and acceleration. Because it combines the strengths of multiple models it is robust to abrupt changes in lighting, can follow an object through occlusions, and can track multiple closely spaced objects.

MAPTrack improves on the agile tracker by using mean shift filtering to integrate motion, appearance, shape, and expected position into one hypothesis, providing "soft switching" that is less threshold sensitive and accounts for multiple models simultaneously. Rather than handle occlusion/track loss and becoming stationary in separate modules like previous approaches, they are integrated into the soft switching equations, providing better robustness in poor conditions. This system was published in the International Conference on Computer Vision Theory and Applications (VISAPP) [21] and has a patent pending [129].

1.3 Overview

Chapter 2 is a review of the literature on similar systems, and systems that have been applied to similar problems. Chapter 3 defines the problems of object recognition and tracking, and explains the various algorithms and processes used to accomplish our objectives. It covers the background of neural networks and the backpropagation algorithm. It defines mixture models and explains the usage of Gaussian mixture models in foreground/background segmentation. It discusses the color appearance model used and the justification for using the pseudo-probabilities it generates. It discusses graph cuts in detail, and explains its application to image segmentation, and reviews and explains all major textural analysis methods. Chapter 4 details the design of the complete food recognition and portion size estimation system and discusses the results. Chapter 5 discusses an

automatic track starter and the agile tracking framework. Chapter 6 details MAPTrack, a human/car tracking framework based on motion, appearance, and position. Chapter 7 is conclusions and future work.

Chapter 2
Related Work

2.1 Food Analysis

Food analysis can be subdivided into 2 categories; agriculture analysis and meal analysis. Agricultural analysis involves applications like grading and sorting raw foods or weed identification, and has simple, predictable constraints. Meal analysis involves analyzing prepared food in a meal on someone's plate. Originally, nearly all food analysis research was for industrial applications. As technology advanced meal analysis became feasible, and split off as a separate area.

2.1.1 Agricultural Analysis

There have been a number of automated food detection and classification methods used for agricultural applications, and to a lesser extent health and medical applications. Throughout the 90's researchers were starting to explore using machine vision for fruit, nut, grain, and meat grading [31]. These techniques mostly consisted of analyzing the silhouettes of food on a flat conveyor belt, or examining the color. These systems used whole fruit or vegetables in an industrial setting, so analyzing silhouettes was often a big part of the process. They were generally intended to separate one specific food type into grades or classes, or at most distinguish between a handful of food types. These methods usually involve machine vision and machine learning, and have preprocessing, segmentation, and classification steps [53]. In [31], early food analysis techniques are surveyed. Another good survey of pre-meal analysis food analysis is given in [202, 203]. More recent developments in industrial food analysis are reviewed in [48] and later in [153].

Preprocessing

Perceived colors vary depending on the type and amount of lighting, and segmentation/classification algorithms may work better on smoothed or cleaned up data - so some kind of preprocessing was usually necessary. In a review of image processing used in food evaluation prior to 2004, [53] notes that the Hue Saturation Intensity (HSI) color space was generally preferred for food analysis applications, or sometimes Lightness A B (LAB). Preprocessing consisted of either simple color space transformations, or filtering operations. They note that low pass type filters can be used to smooth out noise, or high pass types can enhance defects/edges. In practice, median filtering was often used for edge-preserving smoothing on low quality images, but is a nonlinear filtering operation, and therefore slower.

LAB features have been found to be best for measuring color across the curved surfaces of fruit with the A and B channels, maintaining more constant values across curvatures, glossy, and shadowed areas [127]. In [202], the author categorizes HSI as a human oriented color space (because it tries to create channels meaningful to human perception) and LAB as an instrumental one, which may explain this result.

Segmentation

Segmentation involves separating out the foods to be classified, labeling each pixel as "food" or "background". Most segmentation in the earliest period was threshold based or

region(growing) based [53]. The authors of [179] developed a way to segment grain kernels by modeling them as ellipses.

Classification

Classification is the process of automatically categorizing unknown data samples into groups based on similar properties. Normally, examples of each class have been previously observed, and used to make generalizations about that class. The most common classification methods before 2004 were fuzzy classification and Artificial Neural Networks (ANN) [53]. A few years later, ANN and Statistical Learning (SL) were the standard machine learning techniques for food quality evaluation [54]. In [116, 117], Majumdar et al. experimented with classifying bulk grain samples by color and texture analysis, using Color Co-occurrence Matrix (CCM) and Run Length Matrix (RLM) features. CCM features were later used to distinguish between 6 weed classes [32], using several different types of neural networks, including counterpropogation, backpropagation, and radial basis function. Backpropagation showed the best speed and results.

Color and Texture Analysis

All Classification schemes classify food based on some features, so choosing descriptive features is just as important as the classification algorithm. Average color is the most obvious and descriptive feature; but doesn't give enough information alone to classify foods. Textural features are any features that relate to multiple pixels at different spatial locations simultaneously; they can involve information about color differences as well as edges. Textural features can be categorized into 4 groups [22]: statistical, structural, model based and transform based. In [203] the author noted that of these four groups, statistical textures are most often used for food analysis, and structural are not very applicable to foods, and so aren't used. A review of classical texture exaction algorithms can be found in [124].

Many food types have nonhomogeneous colors - i.e., alternating patches or large areas of different colors [15, 16], so analysis must be developed to account for this. The authors of [15] suggests several methods for measuring the amount of color nonhomogeneity in a region, and [16] tested them and compared the results with human perception. In 2009 [93] developed a simple algorithm to segment bicolor foods, which was a start, but the problem was still very much open.

Onward to Complete Systems

The early research on the subproblems involved, and simple industrial systems, slowly but steadily improved the state of the art in complete food analysis systems. In 2010, [133] developed a method for grading baked biscuits. It uses a watershed(gradient based) segmentation scheme and Support Vector Machines (SVM) for classification; both state of the art at the time. The authors suggested and experimented with several different architectures for the SVM, their One-Versus-All (OVA) uses the same basic idea as our One Versus The Rest (OVTR) architecture, but it wasn't particularly well suited to their problem, and wasn't fully developed/implemented well, and gave poor results. The authors of [56] developed a system to grade oil palm fruit, using domain specific knowledge to segment, and ANN to classify. They saw a 1.66% classification rate improvement from dimensionality reduction by Principal Component Analysis (PCA). Although the state of

the art is to the point where complete systems are functional in the field, the area of food quality evaluation is still active and seeing innovations [79,153,193,196]. But recently meal analysis has split off and become a separate category, similar in many ways, but often requiring a different approach.

2.1.2 Meal Analysis

All the agricultural classification systems discussed above were limited to extremely narrow domains, and used specific knowledge of those domains to aid in segmentation and classification. Until around 2009, there was little or no existing work on food recognition and volume estimation applied to meal analysis; for the most part the technology(size, price, resolution, processing power) had not progressed far enough before this. As technology advanced, people proposed new applications, and studies were done on their feasibility. In [165], the authors reviewed several aspects of complete meal recognition systems.

Feasibility Testing

Studies on the viability using digital photography for manual data collection had been in progress for some time. In 2003, [190] studied the validity of using digital photography to manually estimate food volume, by comparing in-person estimates against estimates from digital photography. They found that human analysts have error rates on digital images of anywhere from 1% to 14% depending on the food type; and that using digital images only reduces the accuracy by a few percent from in-person estimation. More complex follow up studies in 2004 [188], 2009 [120], and 2012 [119] gave comparable results and continued to support the idea that accurate volume estimation from 2d images was viable for human analysts, both in cafeteria type setting and in "free living" real-world conditions. A similar study was described in [45], finding that the average of 2 human analysts gave between 9% and 16% estimation error. The authors of [168] also evaluated digital photography as a tool for food intake estimation, without regard to automation, and determined that it is a low cost solution, and provides enough information for a trained human to solve the problem in a repeatable(therefore potentially accurate) way. In 2011, [17] studied portion size estimation in children, finding a 60.3% error rate. The authors of [155] did a similar study among adolescents with errors over 10%, suggesting that error rates below that range on images are either difficult for humans to reach, or that they require trained analysts to achieve. Overall, research clearly shows that digital images carry enough information to estimate volume in a way on-par with direct observation, and that in either case, humans often have estimation error rates over 10%, depending on food type.

Early results in viability studies naturally led to the idea of using automated systems for assistance as these systems became available. In 2008, [208] proposed the idea of a mobile phone based system for food intake estimation such as the one presented in chapter 4. They suggested it be comprised of calibration, segmentation, feature extraction, classification, and volume estimation steps, and comparing before and after meal photos to estimate nutritional intake. In 2009, [27] did a study of the technical viability, challenges involved, and social impact of using mobile devices to measure food intake and laid out a plan for an implementation of [208].

With a proposed problem and rough solution in place, more complex studies were done on manual, semiautomated, and automated food analysis. In [9] the authors did a study

on the viability of automatic collection of daily meal images from cell phones worn hung around the necks of participants. Ten second intervals were found to be sufficient to fully capture food from regular meals, such that a human could identify the type and amount. In [47] the authors studied the viability of actually requiring adolescents and adults to take before and after images at each meal and concluded it was viable. The authors of [159] also studied the feasibility of teaching ordinary people how to use meal capture software and getting them to use it. The authors of [166, 173] reviewed the details of state of the art in manual food intake estimation to encourage/empower researches to try to solve the automation problem.

From Feasibility to Proposed Solutions

Feasibility studies with good results and the earliest proposal of [27, 208] led to more detailed and discrete ideas for how to approach the problem. Machine learning requires a representative training set, and often a very large one. In [186] the authors described a project to create a large database of free-living type foods using digital photography from cell phones.

Of course since the pictures were being taken by cell phone the idea of an app - either on board or communicating with some remote processing server was obvious. In [98] the authors outlined a mobile food recording app to test if applications running on mobile devices have the potential to assess diet.

Normalizing Lighting

In laboratory conditions lighting is standardized, whereas in free living conditions the light spectra are completely unknown, with cafeterias somewhere in between. RGB values do not characterize color well if there can be different lightings [63]. Using ANN trained with each lighting type and a full gamut of colors to convert from RGB to a normalized LAB was proposed to address this problem. However, finding a training sample of every possible lighting doesn't seem very feasible in practice. With a just a baseline black and white level, contrast normalization and a rough color normalization can be done. These black and white levels can be drawn from a reference object in the image, or black/white reference pixels can be found by algorithm and used to normalize. Multiple calibration colors can give a better estimate. Normalizing color makes the task of classification in different lightings much easier.

Normalizing Scale/Orientation

The main early approach to finding orientation/scale involved using a reference card or patterned tablecloth [118]. The card could be the Bullseye patterned black and white card of [121], which can be detected robustly and give black/white levels to simultaneously normalize color. Or it can be a multicolored card like [145, 205, 207], which is hard to automatically detect (none of them did) but should give a better color calibration.

In 2012, [4] suggested using the patient's thumb for reference instead of a calibration card. The authors of [80] pointed out that plates are both common in food scenes and circular, and the ellipses can easily be detected and used for scale/orientation detection, assuming one knows the exact size of the plate. This actually applies fairly well in cafeteria settings, where many food studies are done.

Segmentation

In actuality, segmentation is inextricably linked with classification; if a single food has areas with different colors/textures (which many do [16]), then segmentation by classification must be used. Unfortunately, it's also much easier to classify given an entire region rather than only a small patch of pixels. This creates a chicken and egg problem; classification and segmentation can only be solved optimally by solving the other first. This means either coming up with a sub-optimal solution for one of the 2 problems, creating an iterative approach that converges to optimal, or using domain specific knowledge/human assistance. The majority of researchers assume segmentation or use a simple sub-optimal segmentation. This is probably because segmentation by classification is difficult and impractical with a large number of food types. However some headway has been made on the iterative approach as well.

Since the problem is hard to approach, many researchers simply assume segmentation, or segment manually. Some use threshold based, region growing, or contour based methods; most assume one food type. The authors of [125] used the more modern J measure based SEGmentation (JSEG), allowing them to estimate the number of foods. Our system, [51], and [95] use a graph cuts implementation, GrabCut, that creates a very good segmentation from a few initial guessed seed points. It doesn't inherently estimate the number of food classes, but can often create a full segmentation from simple thresholding (as in our research) or a single manual mouse click per region.

Some systems use domain specific knowledge like [207]. Plates can be detected in an image by a simple ellipse finding algorithm. Plate detection [80] can be a great aid to segmentation, and a number of systems, including ours, use it [51, 125, 205, 207]. In a cafeteria setting, tray detection might be more appropriate.

A few authors are more ambitious and try the iterative approach. In [206], the authors used feedback from the classification step to iteratively re-segment in an attempt to solve the chicken/egg problem of food segmentation vs classification. We did experiment with several segmentation by classification and iterative schemes, involving region merge after pixel-wise classifications; but found that the problem became less feasible the higher the number of food classes. In 2013, [8] proposed a method of segmenting multiple food items using mean shift followed by region growing/merge, which gave good segmentation results (88.5%) on test images. To approach the chicken/egg problem, it used 6 very general food classes for a region merge after the classification. This makes use the observation that most meals are likely to be a "balanced" mix of different types to simplify the problem.

Volume Estimation

Once a food is located and identified, the volume needs to be estimated in order to cross reference with a nutritional database and produce useful nutrient intake estimates.

The simplest models are 2d models. The apparent 2d area of food in a plate or in a clear cup seems (oddly) to be nearly linear function of mass; which varies depending on the size of the segments a food is cut into [180]. We also noted this linearity, and used it in our system; The error rate averages to 15%, varying between 5% and around 25% depending on food type. This is a bit worse than but comparable to human perception (around 10% also with high variance).

Some systems try to estimate 3d shapes from a single 2d image. The authors of [192] presented a method to approximate the area of spherical foods based on a single photo and a calibration card; they approximate prism shaped foods as well, but with manual assistance required. The majority of foods are not perfectly spherical or prism shaped, and for those that are manually fitting a curve to each food largely defeats the purpose of automatic estimation. So generally, single image 3d estimation seems to be hard to implement and more trouble than it's worth.

More sophisticated volume estimation methods use multiple images for 3d modeling; with results similar to or slightly better than human perception. An extensive study on 3d modeling methods is published in [108]. Most such methods use 3d scans, and/or keypoint matching between different viewpoints. In [35] the authors implemented single image volume estimation on cups and prism shaped foods using keypoint detection and active contours. The authors of [145] reported a system with 5% error. In 2013, [50] introduced a 2-image 3d volume estimation system with an error of around 10%. So 3d volume estimation is more accurate than 2d estimation, and may be more accurate than human perception. The downside of course is that the user has to take multiple images, although a well designed app could make this relatively painless. Additionally, these 3d methods still require a reference object to determine scale.

Food Classification Systems

The classifier is the main component of a food recognition system, and draws the most researchers. While there are only a few published, complete systems there are any number of classification schemes and partial systems involving one or more of modules discussed above in addition to the classifier. We feel it provides better insight to consider these modules together, as part of a system.

The first meal classification systems were generalized rather than food-specific. In 2007 [130] presented a machine vision system for sensory evaluation of meals, presumably to aid in automated arrangement and perceptual analysis of TV dinners. It did not use sophisticated segmentation or classification, but managed to perceive layout/arrangement features of the meals very similarly to a human panel. In 2008, [101] developed a web app that would classify a scene as food or not food in an arbitrary image, and estimate the "food balance", a rough measure of which food groups were present.

In 2009, [121] provided what is probably the first implementation of the system described in [208]. They use a small reference card with bullseye patterns on it to calibrate for range and viewpoint(fully automatic), and segment manually. They classify pixel-wise based on color, and follow up with morphological region growing to form contiguous regions.

The same year, [83] provided a more sophisticated classification method for the meal recognition problem without delving into segmentation or volume estimation. It uses a modified form of SVM, called Multiple Kernel Learning (MKL) for classification with color, histogram, Gabor, and Bag of Features (BOF)(of SIFT) features. They used a "one vs. rest" method to handle the large number of classes; but it is unclear how they got from individual classifier outputs to a sorted list of probable food types. They had an 80.05% rate of correct classifications within the top 3 candidates (i.e., the 96th percentile) of foods sampled. As noted in chapter 4, our system gave a 92% rate of ranking within the top 3 (i.e., 98th percentile; more food types), and 95% within the 96th percentile.

In [118] the authors described an in development system for meal analysis using threshold based segmentation plus SVM with color and Gabor features. It used a patterned tablecloth for range/angle calibration. However it was not complete, and it only recognized 11 food types, with classifier architecture not suitable to scale to large numbers of types.

The authors of [145] presented a meal assessment system with several novel features. They attempt to segment by classification (a.k.a. top down segmentation); which could theoretically provide viable results in the fairly common cases where fully automated segmentation before classification is fundamentally impossible. But there are many practical difficulties, particularly for large numbers of classes. They use an interesting classification scheme involving a large set of pairwise classifiers, which minimizes retaining as food types are added / removed. But in order to be viable for large numbers of classes, it requires an already small "candidate set" to start with that must be manually entered via speech recognition software. They calibrate color/scale with a multicolor checkerboard patterned card; but it is not located automatically, the user presumably must click each of the corners in each image. It merges multiple images to generate a 3d model for volume estimation with an error of only about 5%. They later patented the system [144].

In 2010 [198] explored using the spatial relationships between different parts of a multipart food to identify it. The method was tested on only a few food classes and was not generalizable to many food types; but showed interesting and promising results for classifying things like fast food sandwiches and salads. The authors of [205,207] propose a complete system for food intake estimation. They use a colored checkerboard card for scale/color calibration (located manually), and segment using assumptions about the color of the table, plate, and food. For the actual segmentation they tried multiple things: connected components, active contours, and normalized cuts. They used color and Gabor features with SVM for the classification. They classified using 19 food classes; their method doesn't scale to large numbers of food classes [111], or provide a sorted list of runners up. Their volume estimation model uses the prismatic model from [192], but it requires manual input to work. In [25] the authors found that fractal features work well for food texture analysis; outperforming classical texture and Gabor, and introduced a new fractal texture feature that showed good results in this domain.

In [123] the authors developed an app for supermarket food and produce recognition from image sequences using Speeded Up Robust Features (SURF) for shape matching and color histograms for color matching, with bag of features to form feature vectors. SVM was used for the classification, it was tested with 30 food classes and achieved 70% recognition rate within top 3 candidates (10 percentile). The authors of [125] used a combination of deformable part models, bag of features, SVM, plate (circle) detection, JSEG segmentation, SIFT descriptors, and Histogram of Oriented Gradients (HOG). It is significant because it addresses the problem of automatically segmenting more than one food type, which few previous papers had done. A novel automated system using the signal from an electronic ear to detect eating and classify foods was proposed by [139]. It achieved 83% detection and 79% classification accuracy for 8 food types, but proposed no method of estimating intake.

In 2013, we published a complete system [51] for food intake estimation based on [121]. It used plate (ellipse) detection and GrabCut for segmentation, and a One Versus The Rest (OVTR) strategy with ANN for classification. It calculates color, 1st order statistical,

and CCM features over multiple patches and uses a voting process to get a ranked list. It was tested on 110 food types, and its run time is linear with number of classes. Our system gave a 92% rate of ranking within the top 3 (i.e., 98th percentile). The volume estimation is done with an autodetected reference card to calibrate scale/orientation, and fitting a linear area vs. volume model to each food type. In [60] the authors analyzed pasta with several different texture extraction features, including GLCM, Angle Measurement Technique (AMT), and Haralick statistics. They conclude that 1st order statistics and GLCM hold up well against newer methods. In [95] the authors extended previous work in [83] to develop a food recognition system which achieved a recognition accuracy of 81.55% within top 5 candidates (92nd percentile) over 50 food classes. It segments with GrabCut. It uses color features plus bag of SURF to keep processing time down, and uses multiple linear SVMs to get a ranked list, presumably in an OVTR type approach.

In [87] (published in 2014) the authors used Convolutional Neural Network (CNN), which generates its own features internally for food recognition, with good results. They also included a successful food image recognition module using handcrafted features and SVM. In [96] the authors further extended their work in [95] to include 100 food types, and achieve a recognition rate of 92% within the top 5 (96th percentile). Their revised system made use of CNN.

2.1.3 Discussion

In summary, there are a number of challenges in creating a food intake estimator, but there are 3 primary problems that are unique to the problem - food volume estimation, food segmentation, and food classification. As our system and others [180] have shown, volume estimation is surprisingly accurate even by simple means, and more complex means can match a trained human; so it warrants the least attention. Food segmentation is the most difficult problem and has not really been solved satisfactorily, but the segmentation by simplified classification approach of [8] seems promising. It would be even better if there was a formalized way to algorithmically find N optimal simplified food classes for the segmentation. There are few approaches in food classification to the problem of very large number of classes with multiple memberships and meaningful runners up. The one we came up with is One Versus The Rest (OVTR), which seems to be borne out by our results and the results of others. An alternative might be a purely generative model, but in the presence of enough training data the performance probably wouldn't be as high [82].

Integrating user input for verifications and corrections is another significant part of the problem that few have addressed due to most systems being partial. Our system was designed with this in mind, allowing for simple manual alterations at any point in the process.

2.2 Car and Human Tracking

Object tracking is not an AI hard problem, but it is a difficult one. The method and even precise definition of 'object' and 'tracking' varies for different applications. There are numerous viable approaches each with strengths and weaknesses. A survey of various tracking methods, with a particular focus on human tracking is covered in [199].

RADAR was the first form of electronic tracking system, developed before and during World War II. Digital cameras were invented in the 70's, and by the late 80's handheld

digital cameras were publicly available, making computerized tracking possible. The most naive approach would be direct template matching by mean difference in pixel value; but there are two big problems with this. First, it is computationally expensive to compare an object at every possible xy offset against a significant section of the image. Second, if the lighting or object orientation changes even slightly, then the template won't match very well. The problem of tracking is a difficult one, and has be approached in many different ways. Each approach has inherent advantages and weaknesses.

2.2.1 Optical Flow

Optical flow is a movement metric measuring apparent local motions in small neighborhoods, and is based on the way humans and animals perceive motion. In 1981, Horn and Schunck [75] developed a method for globally estimating optical flow from pairs of digital images. The same year Lucas and Kanade [114] developed a local estimation method for doing the same. It creates a vector at every point proportional to how much it appeared to move between the before and after image. Optical flow had obvious and great utility for object tracking, and is still used today for some applications. In an overview of optical flow techniques in 1992, [18] noted that global methods are not suitable for precision applications, and that Lucas-Kanade flow was still on par with the most accurate methods. Even the most cutting edge methods up to the present day often integrate flow as a major component [12, 88, 134, 152, 171].

Optical flow has the advantage of being closely related to the human perception of motion, and being able to follow a specific point around indefinitely. It has the disadvantages of being slow, vulnerable to points appearing and disappearing in the scene, prone to drifting off target, and easily disrupted by "optical illusions" when an object moves over a patterned background.

2.2.2 The Drift Problem

Visual tracking usually includes an object model, estimated motion/position, and model updates. In the case of flow, the bounding box at the beginning of each frame can be interpreted as the model, the motion of the box due to flow as the estimated motion, and the new box as the updated model. For a method like direct pattern matching the model is the template being matched, the place where the template is matched is the estimated position, and a new template copied from that place is the new model.

The problem is that the new position is always an estimate - image noise, approximation errors, and algorithmic imperfections all cause small amounts of inaccuracy. If a new model is generated as a bounding box for example, it will no longer center exactly on the target object. Over time the model can come to represent more of the background and less of the object; eventually losing the correct object entirely [26, 194].

If we refuse to update the model to avoid this, we'll lose track of the object as soon as it no longer matches the model. In the case of flow this would happen immediately when we didn't move the bounding box. In the case of template matching it would happen once lighting/scale/orientation appearance changed slightly and the object no longer registered as a match [126].

So in most trackers, the bounding box can drift off of the target over time - or fall off more suddenly if the object model isn't updated well enough. Some kind of trade off

between stability and flexibility is present in every object tracker. This is known as the "drift problem" or the "model update problem", and is one of the fundamental problems of tracking [184].

2.2.3 Early Tracking, the Kalman Filter

Early alternatives to flow were correlation and/or feature tracking. Cross correlation is essentially a form of template matching, while feature tracking involves extracting some features from the image and matching them instead of the image itself. Of course, the quality of this approach depends on the quality of the features chosen. Shi and Thomasi [157] came up with a mathematically sound method to identify/quantify good features to track. They also proposed a way to quantify feature drift to determine when to stop tracking a feature.

In the late 80's, Kalman filtering and extended Kalman filtering [19, 92] started to be used in machine vision, and by the early 90's, for tracking. The Kalman filter is an algorithm to estimate variables by combining multiple noisy measurements. It assumes a linear signal plus Gaussian noise. In [174] the authors demonstrated their use for tracking large numbers of objects simultaneously on radar-like streams; the same principal is applicable to general tracking. The more general extended Kalman filter was difficult to implement in practice, often producing unstable results due to incorrect assumptions it makes about local linearity in order to handle nonlinear systems [84]. Authors such as [84, 181] (who proposed the "unscented Kalman filter", which modeled nonlinear systems much more accurately) made various improvements. The authors of [10] note that the unscented Kalman filter (which was state of the art at the time) still didn't perform well for bimodal hypotheses, and suggested the up and coming particle filters as an improved method. So while the Kalman filter was an elegant and powerful algorithm, the consensus was that it was not particularly well suited to visual tracking [61].

2.2.4 Tracking vs. Detection and Tracking by Detection

Object tracking implies following a moving object, using the previous physical location(s) of the object to simplify finding it again. Object detection implies an exhaustive search of a large area for one or more instances of a known, recognizable object. In practice, detection is often a subtask during tracking: to know to start tracking when an object has been found, to stop when an object has been lost, to help with the actual tracking, etc.

In the latter case; tracking by detection is a viable tracking method [68]; with built in resilience against large movements and full occlusion. So to a large extent object detection can be considered a subclass of object tracking. For example, template matching is a detection method, not a tracking method, but it is used for tracking. Since it uses more processing power to do an exhaustive search; this approach has become more feasible and more common in recent years [5, 194].

2.2.5 Enhanced Template Matching

Template matching is a too simplistic an approach as it is sensitive to lighting differences, scaling, rotation, and background included in the template. It is also prone to drift, and prohibitively slow to search large regions. But if some or all these weaknesses are handled, it can still be useful. Many machine learning based methods (discussed later), being

detection based, are essentially very advanced template matching, and can suffer from the same problems, and more importantly, benefit from the same advances.

One approach to template matching is to use some transformation to move the problem to a domain with fewer dimensions, or one where the matching problem is easier. One such was an eigenspace based approach that used a "multiple-views plus transformation" model of object recognition [23]. In this method, a large number of training samples are approximated by linear combination of a small number of basis images; and testing images can be approximated from the same basis set to determine their similarity. This "Eigentracking" operates by object recognition rather than looking for motion in the image like flow based techniques.

As with all trackers, researchers tried to take advantage of the strengths of the method while coming up with additions to compensate for the weaknesses. The authors of [69] proposed a framework for tracking large image regions based on pattern matching and minimizing the Sum-of-Squared Differences (SSD) between regions. It is significant because it compensates for lighting changes and transformations between frames, as well as detecting and compensating for occlusion. One solution to the problem of template drift in template-matching type tracking approaches was proposed by [126]. They suggested keeping a short and long term template in memory, loosely modeled after the human memory. This principle was later used to great effect in advanced trackers like the well known TLD tracker [91].

In 2005, [46] revolutionized template matching with a simple method that is indifferent to lighting differences and small positional/scaling differences, and much faster than template matching. Their Histogram of Oriented Gradients (HOG) features rely on finding the general edge directions in a block of space by taking their histogram. Since the edge directions are de-localized over a block, exact position within a block doesn't matter, and the template need not be matched pixel by pixel. Most importantly, HOG features were also well suited to the newly popular machine learning based approaches.

Template matching has the advantage of being tracking by detection, namely it can be used to recover from track loss and occlusion easily. The disadvantages are in speed, and the fact that it is prone to track loss and template drift depending on how often the models are updated. Template matching is also unsuited for fluid or articulated objects.

2.2.6 Enhanced Optical Flow

As noted earlier, optical flow is still used for many applications today, and is also a valuable subpart of many modern trackers. In 2000, [26] proposed a pyramidal implementation of the Lucas Kanade flow algorithm to extend it to track efficiently with large inter-frame motions. It's worth nothing that the concept of coarse to fine pyramidal processing has wider application in speeding up similar algorithms.

A novel tracking using optical flow to guide keypoint tracking was proposed by [152]. It is far more robust than flow however. With spatial dependence between keypoints it handles occlusions and new objects well, and like flow can be used to track an arbitrary point for a long period of time.

2.2.7 Enhanced Difference Images

Another naive approach to tracking on par with template matching is "image subtraction", where 2 images taken consecutively of the same scene are subtracted. Ignoring noise, the differences should be zero at every point in the image if nothing changed. So if an object moved, the differences will be nonzero at the place the object vacated, which is now background, and the space the object moved to, which is no longer background. The most obvious problem with this approach is that you can't tell which of the 2 clusters corresponds to the object without some sort of hypothesis about its heading. Swaying trees and whatnot in the background can also be a problem.

The idea of background subtraction extends simple image subtraction by explicitly modeling the background in order to distinguish when it has been covered by a moving object, and when it has been revealed. In 1999, [163] extended the idea of simple background subtraction by modeling each pixel as a mixture of Gaussians to segment out moving objects. Their method works in the presence of clutter, lighting changes, and repetitive background motions like swaying leaves. Later, [85] improved on their moving foreground segmentation to learn faster and ignore shadows, which the previous system had considered as objects.

Recently, another approach to the problem of separating moving objects from a background has been proposed. In [33] the authors showed how an extension of PCA called principal component pursuit can be used to separate foreground events from a video stream with a stationary background. Later, [204] extended their method to moving background.

For track starting, the state of the art is either object detection, which requires a representative object model, or background subtraction [161, 204].

Tracking based on image difference has the advantage of being applicable to automatic track starting and being able to handle blurry, fluid, and nonrigid objects seamlessly. It is also not prone to drift as such, since the model is reliably updated by image differences. Its weaknesses are that it requires either a stationary or simple background, and that it doesn't explicitly handle stationary or slow moving objects.

2.2.8 Layer Representations

Dynamic Layer Representations are a segmentation method that uses optical flow, but produces segments like background modeling methods discussed above in "Enhanced Difference Images". The Idea is to group pixels into clusters with similar motion properties, thus segmenting out objects moving relative to their surroundings. This is convenient because the clusters can also be tracked by their motion properties.

In 1993, [182] proposed the idea of using the discontinuities at object boundaries in a flow field to detect those boundaries. They refined their system in [183], and showed how it can be applied for motion segmentation, determining relative depth, and video compression.

Later, [170, 171] extended the idea into a more complete solution suitable for practical application. It merges motion, appearance, and shape models of detected layers to continue to track them. It attempts to detect and handle various object states such as occlusion, track loss, and target becoming stationary by a decision tree that transitions through various track states.

Tracking based on layer representations has many of the advantages and disadvantages of image difference, since the results look similar. Layer representations have trouble with non rigid objects though, since different parts don't have clear layer membership. However, moving backgrounds are far less of a problem, and are handled seamlessly.

2.2.9 Advanced Filtering

Kalman filtering did not work especially well for vision based tracking. Any methods that fuse noisy past data that give incomplete information about position individually in order to hypothesize current position are called "filters" in this context. This is because they essentially filter out the noise and find more reliable location information.

In 2000, [40] proposed the "mean-shift" algorithm that assigns a probability to each pixel to be part of the target object, then follows the center of mass of the cluster from frame to frame. It suggests basing the probability on similarity of color to colors near the center of the previous object windows. It also includes a scheme to compensate for changes in scale. This algorithm is well suited to non rigid objects because it doesn't rely on pattern matching. Mean-shift is very simple, yet produces good results, and is applicable to many different types of problems. Later, [41] proposed combining a histogram based appearance model with a shape based kernel to improve on their purely color based model, allowing tracking of more closely spaced and identically dressed humans.

The authors of [78] proposed a probabilistic algorithm based on factored sampling to estimate motion in clutter in 1998. Their "condensation" algorithm would later be known as particle filtering. This algorithm was so interesting because it could handle *multimodal hypotheses*; in the context of tracking that means if measurements indicate several probable headings that diverge in different directions, the filter will handle all of them at once and continue to support/undermine them for some time as more data comes in, rather than immediately forgetting all but the most probable heading. This allows the tracker to recover from temporary confusion or shortages of information. Improvements were made on the standard particle filter for tracking with Kernel Particle Filters (KPF) [36]. It uses kernels to weight the particle likelihood, and utilizes the mean shift procedure. The work of [115] sped up standard particle filtering based trackers by finding a fast approximation with mean-shift. Later [128] used particle filters to fuse shape and appearance cues.

Filtering has the advantage of being very flexible; depending on what data you use it to "filter", similarly its disadvantages depend on the usage. One universal disadvantage is that it is not tracking by detection, and provides no inherent way to recover from track loss.

2.2.10 Machine Learning Based Tracking

After 2000, approaches using machine learning have been increasingly popular, both for tracking by detection (classification), and elsewhere in the algorithms. Tracking by detection and detection by tracking both have inherent problems; so as tracking by detection came into use, researches needed to combine it with detection by tracking. Particle filtering had shown impressive promise, but had many problems that needed to be worked out; and the mean-shift procedure remained popular in many different types of tracking.

Early attempts at tracking by learning based detection suffered from most of the same problems as template matching, since machine learning is, at its core, a similar idea. With

enough training data and a homogeneous enough class of objects, it was possible. In one such application, [12] used the pyramidal flow of [26], to integrate learning based object classification with flow based tracking to track (the backs of) cars.

In 2005, the HOG features of [46] (discussed in enhanced template matching above) produced better results than any contemporary method for human detection in images. The resulting blocks and histograms their algorithm produces capture the general shape of an object and without requiring precise registration, and make excellent features for machine learning.

Boosting Based Detection

The boosting algorithm [156] is a general method for combining many weak classifiers into a stronger one. In 2001, [177] proposed a machine learning based object detection algorithm that can be used for tracking. It uses boosting to combine weak rectangle features into a stronger classifier, and cascades faster classifiers to reject easily ruled out regions with successively slower but more powerful classifiers to choose between likely candidates.

In [136] the authors used boosting based object detection for automatic track starting, and extended the particle filtering of [78] with boosting based detection to track hockey players. In [13] the authors combined boosting based detection with mean-shift tracking to track humans and cars. The background colors/textures are used to infer object colors/textures, and train the classifier online.

Online Learning

Rather than try to match the target object against any kind of template, an alternative is to build foreground and background models of some kind (discussed in the sections on Enhanced Difference Images and Layer representations), and use them to train a learner. Within a few frames it can produce very good segmentations with very little processing power required to get them. Such a classifier cannot be trained ahead of time; since they differentiate a specific object from a specific background, so some initial segmentation must be provided by some other method.

A classifier that is fully trained ahead of time is called offline, and generally is a template for a general object class. A classifier that is continually trained during runtime is called online, and generally discriminates a specific object from its current background. Offline classifiers are able to start tracks, and don't exhibit model drift, but cannot handle non rigid or multiple objects well. Online classifiers are more accurate and flexible, but cannot start tracks and experience model drift.

The authors of [64,65] proposed a method using boosting for feature selection in tracking. They do foreground/background segmentation through classification, using a mean shift like approach to do online, fully classification based tracking. The author of [135] also did classification based foreground/background detection.

To improve online tracking and mitigate its weaknesses, a number of improvements have been suggested. In [14] the author showed a way to use multiple instance learning to handle model drift in purely online based detection. The author of [151] proposed the use of random forests (an amalgamation learner similar to boosting) for online learning based tracking. Due to their extremely high speed and parallelizability, they are well suited for real-time applications. Kalal and Matas proposed a way of labeling unlabeled data online

using structural constraints in [89], and a method of detecting tracking failure by tracking both forwards and backwards in time in [90]. In [201] the author integrated semi-supervised and multiple instance learning into one tracker.

Multi Object Tracking

One problem with tracking by learning is that it does not handle multiple objects particularly well. Online learning differentiates objects from background rather than each other, nor do offline methods distinguish between objects. Common approaches to handling this are coupling detection with traditional tracking, or concentrating on differentiating between objects in a dedicated module.

In [105, 106] the authors address the problem of tracking multiple humans simultaneously in crowded scenes. They combine feature and silhouette based cues over multiple frames to handle large amounts of occlusion, coupling detection and trajectory estimation into an optimization problem. To get partial silhouettes in this situation, they use top down segmentation.

The authors of [135] improved on their work in [162] to better handle various problems. They note simple online training suffers from the drift problem; indeed all online tracking methods have a trade of between drift and adaptability. They also note that learning based tracking is sensitive to occlusions, and multiple similar objects. Their upgraded method uses separate classifiers for detection, tracking, and differentiating between similar objects, which simplifies the problem and accounts for drift, occlusion, and multi-object tracking. In [66] they demonstrate tracking fully occluded or out of frame objects by analyzing their statistical relation with the motion of visible points.

Integrating Offline Detection with Online Learning

Online and offline learning each complement some of the other's weaknesses, so methods like [135] that integrate both are common.

In [67] the authors handled the problem of integrating recognition and tracking elegantly by training a classifier offline, then introducing nearby background patches as additional training samples once the track had been started, and training online with the combined dataset. The authors of [107] describe the semi-supervised boosting they used in more detail. In [200] the authors described a similar approach, co-training generative (offline) and discriminative (online) models.

In [57], the authors used CNN for human tracking, taking into consideration the drift and multi-object problems.

Integrating Tracking by Detection and Detection by Tracking

Another way of handling the weaknesses of learning based tracking is to couple it with traditional tracking methods. A number of trackers have given good results coupling with simple optical flow.

The work of [109] focuses on the problem of tracking in low frame rate video. They hypothesize that tracking by detection is the best approach in such scenarios due to low motion continuity, but note that there is discontinuity in detection features as well. They model the tracker as a group of observers with different lifespans; short lifespans being more

classifier-like, long lifespans more tracker-like. The idea is based on and utilizes particle filtering.

In [88] the authors integrate tracking by detection with optical flow, and have a strategy to minimize drift while still allowing for large changes in object appearance over time.

in 2012, [91] extended their work of [88, 89] with the well known Tracking Learning Detection (TLD) algorithm, which breaks the problem of tracking into simpler sub-tasks in a manner similar to [162]. It is a learning based method that smoothly optimizes between tracking by detection and detection by tracking, and is commercially available under the name "Predator" tracker. It cannot handle multiple objects, but is very flexible and applicable to choppy and occlusion-heavy video due to fully online training, low drift, and a strong detection component to recover from errors.

In 2013, [134] came up with an algorithm that used optical flow to improve machine learning on keypoints in a manner based on the TLD of [91].

Learning Based Tracking - Advantages and Disadvantages

Machine learning based tracking has the advantage of being relatively flexible, powerful, and easy to implement. It is generally some type of tracking by detection, so it has the advantage of easy track recovery as well. If trained offline, it can be used for track starting and doesn't drift, but has the disadvantage of needing a huge, representative dataset, and losing adaptability. If trained online it has much more adaptability the disadvantage of the drift problem. Despite the power of the approach, neither method is well suited to lighting changes, quality changes, occlusions, or multiple objects, so each of these problems must be accounted for in practical trackers.

2.2.11 Part Based Models

Machine learning is not well suited to articulated objects, or objects that vary in appearance, or objects for which a large training set isn't available. One approach to these problems is considering an object to be a collection of recognizable parts, with their relative positions only loosely related.

In 2002, [2] proposed an object detection method that represents and object as a collection of high level parts. It automatically learns these parts from lower level interest points. It is applicable to objects with similar and similarly arranged parts, and was demonstrated for car detection (always in profile, from the same angle) with good results. These parts can be templates of physical parts of the object, or they can be extracted features. In [1], position and scale of an object are estimated from the histograms of multiple patches.

Part based models have the advantage of being able to generalize from few examples [58], match to examples they haven't been explicitly trained with, and are directly applicable to activity classification. Of course they are only as good as the combination of their feature choices and the algorithm to match groups of related features; practical implementation can be complex [81].

2.2.12 Non-Rigid Objects

Learning based tracking tends to have trouble with non-rigid objects, and fail on objects that undergo rapid appearance changes. Vehicles or animals that turn around suddenly

and show an unknown side, humans that bend into shapes not seen in training, and anything with joints that changes shape can cause problems. Many methods use patch-based classification to address this; extracting part-like features that don't vary rapidly over space from tiled regions over the object of interest. In 2011, [34] modeled objects as a collection of patches with a separate layer of global properties/motions shared among them. Local histogram features were an important part of the patch recognition. The authors of [62] used random forests (a machine learning algorithm) with generalized Hough transform to track non rigid objects. They use the GrabCuts algorithm to segment and select better training samples, all but eliminating drift as long as segmentation was possible. Others have modeled nonrigid objects as a geometrically related group of patches using simple color features, with competitive results [102].

2.2.13 Limb Tracking

Limb Tracking, the tracking of individual limbs on tracked humans, is a topic of particular interest in human tracking, because of the extra information we can infer from grasping objects, gestures, etc. It can be approached bottom up as a special case of part based models, or top down as a much more complex type of template matching [142].

In a top down approach, [103] tracked limb positions of humans, compensating for the problem of self occlusion. That is, the fact that limbs are regularly blocked by the body or other limbs, making their position difficult to calculate accurately by direct observation. They model a human as a 2 dimensional collection of rigid parts connected by springs to constrain their positions in space, and use hidden Markov temporal models to constrain their positions in time. Silhouette based trackers lend themselves to top down approaches just as part based lend themselves to bottom up. In this way [5] extended the silhouette based work of [106] by modeling limb positions, estimating pose over multiple frames using a dynamical walking model to form tracklets, and combining tracklets to form longer tracks. The authors of [146] combined top down human detection with bottom up part detection to model and track limbs on detected humans. In another approach, [59] used deformable part models in a human recognition system.

2.2.14 Surveillance

Surveillance is best viewed as a large set of different, difficult problems that share some approaches in common, but need to be approached individually. In 2008, [49] published a study on the state of automated visual surveillance. They state that there are many "sub systems and partial solutions which go some way to wards solving elements of" the "problem of surveillance". They conclude that the successfully deployed automatic surveillance systems to date used domain specific knowledge and constraints to reduce the problem complexity.

Thinking along these lines, [104] extended their multi object tracking [105] to use trajectories from previous humans to provide likelihood information about future ones, for use in a fixed surveillance application. The authors of [148] used a grid of simple learning based trackers to handle a fixed surveillance application, and found it to be competitive with more complex (but non domain specific) state of the art trackers. Building on their classifier grid for fixed surveillance cameras, they point out that a problem with all classification

based tracking is getting a representative training set. To alleviate this they provide a method of supplementing the data with unlabeled offline data in [161].

In the case of aerial surveillance, the camera is moving, and the scene is changing, so the simplification approaches mentioned above will not work. Furthermore the resolution is usually several orders of magnitude worse than for a non-moving ground-based camera much closer to the target. To approach this problem,the dynamic layer representation of [170, 171] was applied to tracking distant, low resolution vehicles from the air.

Many surveillance systems don't take so much a video as a sequence of images, once every few seconds, to conserve space. In [112] the authors study the problem of analyzing very low frame rate (a few frames per minute or less) image sequences. Under these conditions, an individual object is not trackable; but they note that learning based detection, and learning based on timestamp of periodic events and metadata are possible.

2.2.15 Discussion

The best approaches to surveillance are application specific [49], so in order to design a surveillance system we need to examine the intended application carefully. The example scenes we were interested in tracking were drawn from the /acrshortvirat Aerial Dataset and /acrshortvirat Ground Dataset [178]. The aerial photos are regularly blurry and low resolution, with camera motion and variance in viewpoint. The ground photos are better resolution, better quality, and have little camera motion. Both contain humans and cars, the objects we are interested in tracking. The objective was to design an algorithm suitable for automated surveillance on both datasets, with an eye towards being connected to an event detection layer as part of a larger system.

An automated system would need to be able to automatically start tracks, so it would need to either use offline detection based on prior class examples or some form of enhanced difference images [161, 204]. Because of the extreme variability in viewpoint, scale, object shape, and image quality, offline machine learning does not seem like a feasible approach. The most promising state of the art foreground segmentation approaches are Gaussian mixture like [85], or layer representations like [183]. Either would be fine for surveillance; the layer representations might be better for aerial videos because they don't have to explicitly account for camera motion. For general car/human tracking, Gaussian mixture models should be able to produce simple and complete silhouettes which can potentially be used by a top down body tracking algorithm [5]. Part based models would be a useful feature to enable bottom up body tracking, but as discussed above, pre-trained classifiers are not viable, and many of the videos we want to track are poor enough that parts would not be distinguishable.

Since Gaussian mixture models foreground segmentation doesn't inherently account for motion, a stabilization algorithm would be necessary; the aerial videos in question have many more background than foreground pixels, and usually have relatively slow, simple camera motions. In the cases where the camera moves suddenly, things are usually moving out of frame and are no longer trackable anyway. Because of this and the fact that optical flow often proves useful to bolster other tracking methods [12, 88, 134, 152, 171], we used pyramidal flow [26] as the basis for our stabilization.

For the tracking itself, either blobs picked up by the foreground segmentation or optical flow seem like convenient choices, since the data is already acquired for previous steps.

Unfortunately, optical flow by itself is unreliable and prone to drift, can be disrupted by near passes or a textured background. Foreground segmentation doesn't differentiate well between objects, and cannot track an object that stops moving briefly. So if these methods are used to track, near passes, similar objects, track loss, and occlusion need to be handled.

Every individual tracking method discussed above has strengths and weaknesses. Most robust trackers combine multiple strategies - such as tracking plus detection or flexible plus reliable in order to produce practically useful results [5, 67, 88, 91, 134, 135, 152, 171]. Also, domain specific knowledge has proven to be useful in surveillance applications [49, 148], meaning the best solution varies with the specific problem. So clearly some method to combine multiple trackers is in order.

The agile tracker switches between tracking methods to when a confidence measure associated with a given tracking measure starts to drop. For example if the object stops moving, it would switch off of Gaussian mixture foreground segmentation; and if it continues moving for some time or crosses a textured background it would switch off of flow to avoid drift or confusion. Gaussian mixture foreground segmentation inherently handles partial occlusions, and the confidence measures provide a natural way to detect track loss. It uses simple Gaussian kernels in a manner similar to [41] to handle near passes and similar objects.

Many good nonrigid object trackers such as [34] use tiled patches with color histogram features to model objects. Other such as [102] use color or texture features targeted freely around arbitrary points. The block based approach requires the object to be, at minimum, almost as big as a block; so may not be viable on very distant cars or humans from the air. Similarly, most feature extraction approaches require some neighborhood of pixels. Scale invariant features are an exception, but for distant, blurry objects, tracking by detection based on only scale invariant features might still be a problem; such objects seem unlikely to have an abundance of good features.

MAPTrack, with this in mind, integrates pixel-wise color model information into the tracking, and replaces the switching algorithm with a more stable "soft switching" equation, followed by mean shift [40] filtering in order to track. It also uses a more complex kernel built from previous observations for the shape model. Considering the recent progress in online discriminative tracking, a learning based appearance model might be a significant future improvement, even with small or single pixel features. Keeping in mind that being able to match by appearance model with an object in another scene would be useful in higher layers of the system.

Chapter 3
Preliminaries

3.1 Graph cuts segmentation and max-flow min-cut theorem

Graph cuts is a mathematical technique that has been successfully used to solve a number of low level computer vision problems, including segmentation. Our food recognition system uses graph cuts based segmentation, and the same ideas may be applicable to future versions of the tracker as well. The basic idea is to redefine the segmentation problem as a min-cut max-flow problem, a graph-based energy minimization problem with a known solution. However finding an exact solution is often computationally prohibitive. In order to solve the problem efficiently, an approximate solution can be found via graph cuts. By varying the details of the energy function being minimized, and how the graph problem is set up, the segment type, quality and segmentation speed can be varied. Graph cuts segmentation has been used to segment single images based on only a few pixels of the object to be segmented as initial information. It has been used in the same way on single frames of video sequences, using tracker output as the initial guess to prevent model drift. The graph cuts procedure can be applied more generally to a whole class of optimization problems associated with balancing well connected image segmentation against other constraints.

This section will explain how graph cuts sets up segmentation as an optimization problem, how that optimization problem can be solved by using the method of graph cuts. It will show examples of simplest cases, then discuss more complex details necessary for actual application, including extending graph cuts from the binary case (2 classes) to multiple classes, algorithms to efficiently get approximate solutions to the graph cuts problem, what energy functions to use, and discuss a practical implementation called GrabCuts [149].

3.1.1 Segmentation as Energy Minimization [28]

First, segmenting an image into foreground / background is treated as a binary classification problem, with each pixel labeled as foreground (the object of interest) or background. An energy cost is associated with each labeling, depending on how well the label satisfies some constraints. The problem can be solved by finding the combination of labels that gives the smallest total energy cost across the image [28]. A standard form for the energy term is

$$E(L) = E_{data}(L) + E_{smooth}(L) \tag{3.1}$$

for some labeling L. The goal is to find the labeling L that minimizes the global sum:

$$E(L) = \sum_{p \in P} E_{data}(L_p) + \sum_{\{p,q\} \in N} E_{smooth}(L_p, L_q) \tag{3.2}$$

where p and q are pixels, N is the set of pairs of pixels in the same neighborhood, and P is the entire image. L is the proposed labeling for the entire image, L_p is the proposed label for pixel p. E_{data} is based on the similarity of the current pixel to some model, and E_{smooth} is based on the similarity of the proposed label to the labels of nearby pixels (L_q).

In practice, the 2 terms can be weighted differently to give more precedence to one or the other. A good weight can be determined experimentally.

E_{data} measures the similarity of a pixel to some model, which is generated from observed data, as shown in Figure 3.1. The simplest form of E_{data} is

$$E_{data}(p, L_p) = (I_{L_p} - I_p)^2 \tag{3.3}$$

where I_{L_p} is the average intensity of the observed data for that class, and I_p is the intensity of the current pixel. A more sophisticated measure could be the histogram based probability introduced in [74]. In this method a normalized histogram of the observed data on the hue channel is taken, and a candidate pixel is assigned a probability based on the count in the corresponding bin:

$$E_{data}(p, L_p) = \begin{cases} 1 - \frac{hist_\alpha(H_p)}{\max(hist_\alpha)} & \text{if } L_p = \alpha \\ \frac{hist_\alpha(H_p)}{\max(hist_\alpha)} & \text{if } L_p = \beta \end{cases} \tag{3.4}$$

where H_p is the hue of the current pixel, $hist_\alpha$ is the initial foreground histogram, α is the foreground label, and β is the background label. With models of both foreground and background, probability could be determined by the relative height of the histogram bin in fg/bg models:

$$E_{data}(p, L_p) = \left[\frac{hist_{L_p}(H_p)}{\max(hist_{L_p})} - \frac{hist_{\tilde{L}_p}(H_p)}{\max(hist_{\tilde{L}_p})} + 1 \right] / 2 \tag{3.5}$$

Since lower energies correspond to a better match, the similarity measure must be manipulated to reflect this. For texture based segmentation, the similarity to a texture model can be used instead. The foreground/background models, whether they are average intensities, histograms, or something else, must be found from some initial guess or previous example.

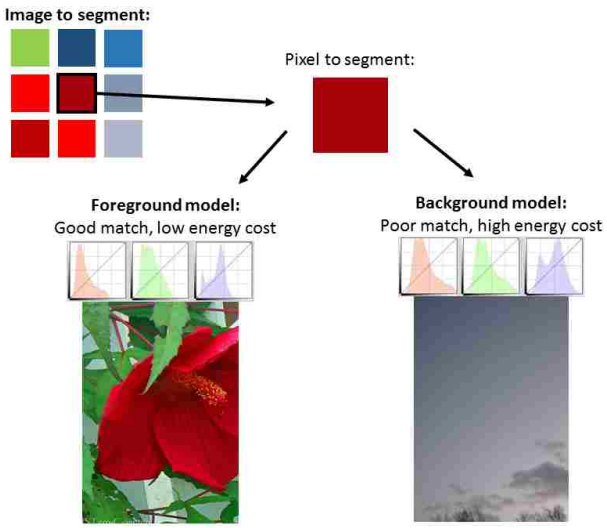

Figure 3.1: E_{data} measures similarity of each pixel to some model; more similar costs less energy to label.

E_{smooth} measures if a pixel has a similar label to each of its neighbors, as shown in Figure 3.2. The simplest version would be zero if neighbors share the same label, a constant otherwise:

$$E_{smooth}(L_p, L_q) = \sum_{\{p,q\} \in N} \begin{cases} 0 & \text{if } L_p = L_q \\ 1 & otherwise \end{cases} \tag{3.6}$$

where N is a the set of adjacent (8-connected) pixels. In practice it is recommended to weight the penalty for not matching a neighbor less if the appearance is very different, giving an effect similar to the bilateral filter:

$$E_{smooth}(L_p) = \sum_{\{p,q\} \in N} \begin{cases} 0 & \text{if } L_p = L_q \\ 1 - |I_q - I_p| & otherwise \end{cases} \tag{3.7}$$

where I_p and I_q are the intensities of the current and neighboring pixels, normalized to range from 0 to 1, so that the energy term is non-negative. After some experimentation, [29] suggested the following to give good results in practice:

$$E_{smooth}(L_p) \propto \sum_{\{p,q\} \in N} \begin{cases} 0 & \text{if } L_p = L_q \\ \exp\left(- \dfrac{(I_q - I_p)^2}{2\sigma^2} \right) \cdot \dfrac{1}{dist(p,q)} & otherwise \end{cases} \tag{3.8}$$

where σ is a penalty threshold representing the intensity level of camera noise, and $dist$ is the distance in space between a pair of pixels. Hue difference, edge strength, or gradient direction can be used in the same way as intensity difference. Alternatively, E_{data} may provide an already calculated measure of color/texture to use for comparison.

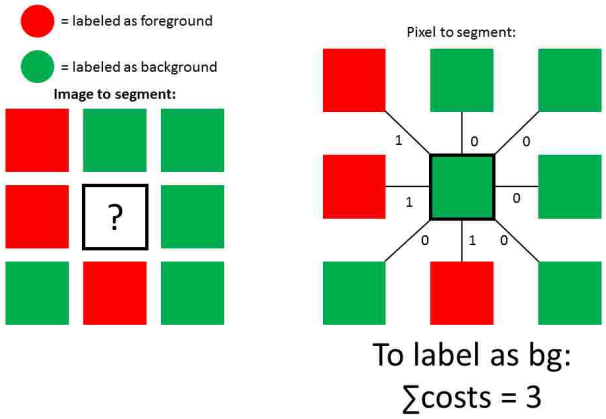

Figure 3.2: E_{smooth} measures similarity of each pixel to is neighbors; each differently labeled neighbor costs energy.

3.1.2 Energy minimization as Known Graph Problem [28]

The problem with energy minimization in the context of image (or video) processing is the high computational cost. Even a low resolution image has hundreds of thousands of pixels, most have millions. The dimensionality of the space we're looking for the minimum in is $|P|$ (number of pixels), and the space is full of local minima so a simple gradient

descent or the like will fail [28]. In the past, various methods were attempted to minimize energy efficiently, with limited success. Notably, simulated annealing was popular for a time, but it was very slow.

The minimization problem can be reformulated as a graph problem, where each pixel is a node, connected to its neighbors by edges weighted by a smoothness term W_{smooth}. There are 2 additional nodes, one for each class, and each pixel is connected to both of these by edges weighted by the a data term W_{data}, as shown in Figure 3.3. Note that these terms are different than the ones for the non-graph-problem formulation. The data term is minus the original data term, so the simplest case of Eq. 3.3 becomes:

$$W_{data}(p, \alpha) = -(I_\alpha - I_p)^2 \tag{3.9}$$

$$W_{data}(p, \beta) = -(I_\beta - I_p)^2 \tag{3.10}$$

for the weights of the edges attached to the 2 class nodes. The smoothness term is the original smoothness term, for the case $L_p \neq L_q$. In the simplest case this is just a constant, Eq. 3.6 becomes:

$$W_{smooth}(p, q) = 1 \tag{3.11}$$

for every edge connecting adjacent pixels. It is important that the smoothness terms always be non-negative so, everything else being equal, few cuts are always less costly than many cuts.

The idea is to separate the graph into 2 unconnected subgraphs, each containing one of the label nodes. Thus each pixel node is attached to one and only one label node. Doing this by minimizing the sum of the cut edges is equivalent to the energy minimization problem.

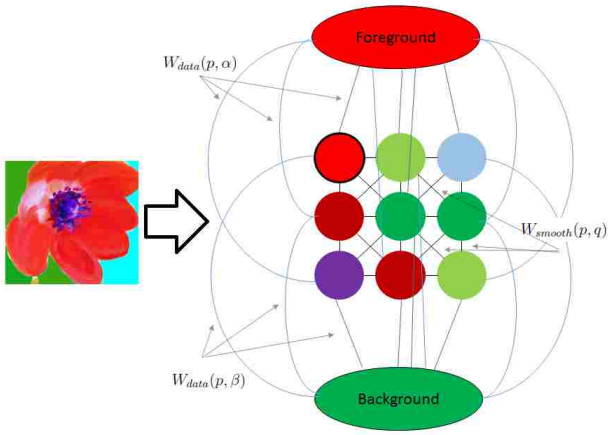

Figure 3.3: Node for each class and pixel, higher energy costs convert to lower edge weights.

3.1.3 Graph Cuts

Boykov et al. [28] introduced the idea of a graph cut, formally stated: *Let* $\mathcal{G} =< \mathcal{V}, \mathcal{E} >$ be a weighted graph with vertices $\mathcal{V}$, edges $\mathcal{E}$, and 2 unique vertices called the terminals. A *cut* $\mathcal{C} \subset \mathcal{E}$ is a set of edges such that the terminals are separated in $\mathcal{G}' =< \mathcal{V} - \mathcal{C}, \mathcal{E} >$, and no proper subset of $\mathcal{C}$ separates the terminals in $\mathcal{G}'$.

Figure 3.4 shows an example of a graph cut. The minimum cut is the cut with the smallest sum of edge weights. There are numerous min cut algorithms available with low polynomial time, some of which function in near linear time (with number of pixels) in practice. Notably, the solving the max flow problem for a network is equivalent to determining the min cut.

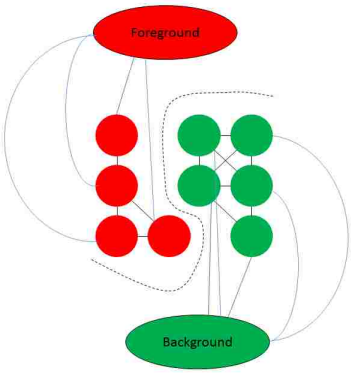

Figure 3.4: Find the minimum cost cut to divide the graph into 2 subgraphs, each containing one class node.

3.1.4 Muticlass Graph Cuts and Efficient Approximations

Some segmentation problems simply need to separate foreground from background, and thus have only 2 possible pixel labellings. However many segmentation tasks involve dividing the image into multiple segments, either a fixed number, or based on some quality metric. For these situations it is useful to extend the idea of graph cuts based segmentation to more than 2 labels.

[28] developed two energy minimization algorithms: swap-move (Algorithm 1) and expansion-move (Algorithm 2). They are designed to give efficient approximations of the optimal labeling for more than 2 classes. They are guaranteed to converge in a finite number of cycles, and in practice converge very quickly. Furthermore, the expansion-move algorithm is guaranteed to end within a factor of two of the global minimum. Both these algorithms rely on solving modified min cut problems as an integral step.

Swap-Move [28]

During a swap-move, pixels from 2 classes either stay the same or change to each others' labels (Figure 3.5). Two labels α and β are chosen from the set of all labels $\mathcal{L}$. $P_{\alpha\beta}$ is the union of all pixels initially having either label. Each pixel p and the terminals for the chosen classes, T_α and T_β are vertices on the graph $\mathcal{G}_{\alpha\beta}$. Each pixel vertex is connected by edges to the vertices of its neighbors, and to the 2 terminals. Unlike in the 2 label examples shown above, the swap move and expansion move assume the terminal a pixel is *severed* from is the labeling it is assigned, so $E_{data}(p, L_p)$ is used directly for the edge weights. The edge weights to set up the graph for this problem are shown in Table 3.1.

Expansion-Move [28]

During an expansion move, the pixels from one class expand into a superset of themselves, relabeling pixels from 1 or more other classes (Figure 3.6). Two labels α and $\bar{\alpha}$ are

Algorithm 1 swap-move algorithm

1. Start with an arbitrary labeling L
2. Set success to $false$
3. **for** each pair of labels $\{\alpha, \beta\} \subset \mathcal{L}$
   3.1. Find $\hat{L} = argmin\ E(L')$ among L' within one $\alpha - \beta$ swap of L
   3.2. **if** $E(\hat{L}) < E(L)$, **then** Set $L := \hat{L}$, Set success to $true$
4. **if** success is $true$, **then** goto 2
5. Return L

Table 3.1: Swap-move edge weights

edge	weight	for
$\{p, T_\alpha\}$	$E_{data}(p,\alpha) + \sum_{\substack{q \in N_p \\ q \notin P_{\alpha\beta}}} E_{smooth}(\alpha, L_q)$	$p \in P_{\alpha\beta}$
$\{p, T_\beta\}$	$E_{data}(p,\beta) + \sum_{\substack{q \in N_p \\ q \notin P_{\alpha\beta}}} E_{smooth}(\beta, L_q)$	$p \in P_{\alpha\beta}$
$\{p, q\}$	$E_{smooth}(\alpha, \beta)$	$\substack{p,q \in P_{\alpha\beta} \\ \{p,q\} \in N}$

chosen from the set of all labels $\mathcal{L}$. P_α is initial α labeled region, and $P_{\bar{\alpha}}$ is everything else. Similarly to swap-move, each pixel p and the terminals for the chosen classes T_α and $T_{\bar{\alpha}}$ are vertices on the graph $\mathcal{G}_\alpha$. Additionally, an auxiliary node a is created for each pair of pixels in the same neighborhood but with different labels. Edges are created between each pair of pixels with the same label, but for pixels with different labels, both are connected to the auxiliary node instead. The auxiliary nodes are also connected to $T_{\bar{\alpha}}$. As with the swap-move, it assumes the terminal a pixel is severed from is the labeling it is assigned. The edge weights to set up the graph for this problem are shown in Table 3.2.

3.1.5 GrabCuts [149]

A minimal amount of user provided information can be used to set up hard constraints for graph cut based segmentation [29]. These constraints consist of a few seed pixels chosen in advance to be part of each segment (Figure 3.7). The method provides a globally optimal solution when a cost function is clearly defined. This method often produces good results with just a few loosely placed seeds. If more seeds are added later, the new optimum can be computed very efficiently.

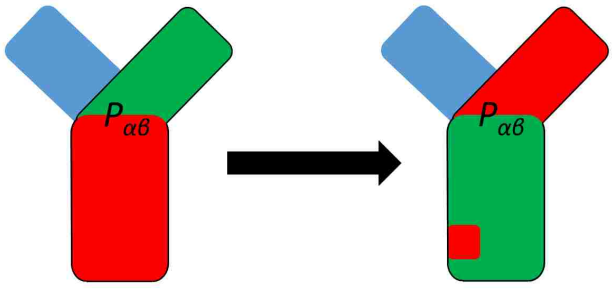

Figure 3.5: Swap-move

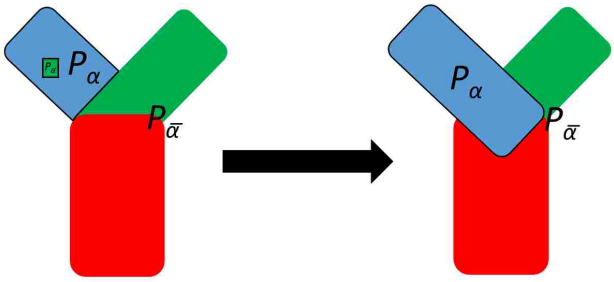

Figure 3.6: Expansion-Move

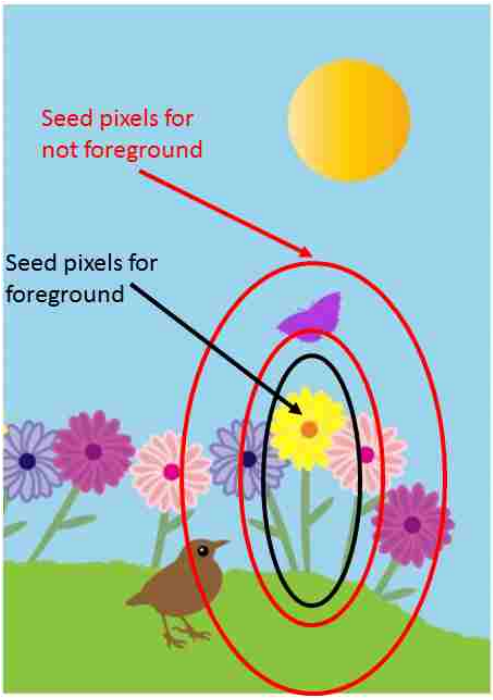

Figure 3.7: Small number of user provided seed pixels for initial conditions.

Algorithm 2 expansion-move algorithm

1. Start with an arbitrary labeling L
2. Set success to $false$
3. **for** label $\alpha \in \mathcal{L}$
   3.1. Find $\hat{L} = argmin\ E(L')$ among L' within one $\alpha-$ expansion of L
   3.2. **if** $E(\hat{L}) < E(L)$, **then** Set $L := \hat{L}$, Set success to $true$
4. **if** success is $true$, **then** goto 2
5. Return L

Table 3.2: Expansion-move edge weights

edge	weight	for
$\{p, T_{\bar{\alpha}}\}$	$\overset{\infty}{E_{data}(p, L_p)}$	$\overset{p \in P_\alpha}{p \notin P_\alpha}$
$\{p, T_\alpha\}$	$E_{data}(p, \alpha)$	$p \in P$
$\{p, q\}$	$E_{smooth}(L_p, \alpha)$	$\{p, q\} \in N, L_p = L_q$
$\{p, a\}$	$E_{smooth}(L_p, \alpha)$	$\{p, q\} \in N, L_p \neq L_q$
$\{a, q\}$	$E_{smooth}(\alpha, L_q)$	$\{p, q\} \in N, L_p \neq L_q$
$\{a, T_{\bar{\alpha}}\}$	$E_{smooth}(L_p, Lq)$	$\{p, q\} \in N, L_p \neq L_q$

The GrabCut algorithm, introduced in [150], is a practical implementation of the graph cut idea for image segmentation. Unlike standard graph cuts, the algorithm uses iterative estimation and allows for incomplete labeling, minimizing the amount of user input needed. A guess based on a few "probably known" pixels will converge to a good solution in only a few iterations. A rough box around the area containing the object of interest is often sufficient for very accurate segmentation (Figure 3.8).

Figure 3.8: left:very rough user provided input, right: accurate results

GrabCut explicitly models the problem for color images; their smoothness term is essentially the same as the original standard:

$$E_{smooth}(L_p) = \sum_{\{p,q\} \in N} \begin{cases} 0 & \text{if } L_p = L_q \\ \exp\left(- \frac{\| \langle r_q, g_q, b_q \rangle - \langle r_p, g_p, b_p \rangle \|^2}{2\sigma^2} \right) & otherwise \end{cases} \tag{3.12}$$

Histograms in 3 dimensions are much sparser than one dimensional (standard histograms), for the same number of pixels. For the data term, rather than try to extract 3d histograms, they approximate foreground and background distributions as mixtures of Gaussian functions, with around 5 Gaussian components each. This lets them get a good approximation with less data. Each pixel is assigned one Gaussian Mixture Model (GMM) component k, from the foreground or background model according to its label. So (L_p, k_p) identifies a single specific Gaussian distribution in 3 dimensional RGB space. The data term becomes:

$$E_{data}(p, L_p, k) = -\log \pi(L_p, k_p) + \frac{1}{2} \log \det \Sigma(L_p, k_p) + \frac{1}{2} [z_p - \mu(L_p, k_p)]^\intercal \Sigma(L_p, k_p)^{-1} [z_p - \mu(L_p, k_p)] \tag{3.13}$$

where π is the GMM component weight, Σ is its covariance, and μ is its mean. For conciseness, the RGB values of a pixel are represented by the vector z_p.

Iterative estimation is useful mostly because it allows the algorithm to bootstrap itself from very little user input, and is guaranteed to converge to some lower energy when used with the proposed GMM data term. This is what allows for incomplete labeling. Figure 3.8 labeled a strip of pixels outside the bounding box as permanently background, but there was no explicit foreground mapping. Pixels inside the box were used to estimate an initial foreground model without any information about their true labels.

3.1.6 Discussion

Graph cuts based segmentation is state of the art in image segmentation, allowing for user defined connectivity and object models in a way that makes it applicable to a variety of segmentation problems, including ours. Furthermore, if its algorithmic complexity can be overcome, it may be useful in setting up video segmentation as an optimization problem in the future.

3.2 Texture Analysis

Hand crafted features i.e., any features that were not automatically generated by some algorithm, are state of the art in texture analysis. Because these features tell the network everything it knows about the input data, a network is only as good as its input features, and a clear understanding of both the problem domain and the features is necessary for success. Hand crafted textural features take many forms, with no one type dominating so much that it alone can solve any problem.

Textural features refer to structural or statistical properties of a multi-pixel localized area in an image. When basic color information isn't enough, but full object recognition isn't a viable solution, texture can be useful to characterize objects or regions [154]. The most basic type of texture features refer to properties of the grayscale channel of the image,

but textural features can refer to the relative positions of different color information as well. Applying the same set of feature extractions to the Hue, Saturation, and Intensity channels of an image is the standard way to do this.

When analyzing the texture of the intensity channel, it may be advisable to normalize by subtracting the mean and dividing by the standard deviation before analyzing. This is in order to account for lighting and exposure variations; as is done with normalized cross correlation. This can also be done separately for each local region in order to account for local lighting variations and to remove the effect of brightness/contrast on purely textural features.

Object/region classification is only one application of texture analysis. Textural features can be used to enable image segmentation on complex natural landscapes where there are no clear edges and color information is not sufficient. Shape from texture is the idea of extracting 3d orientation information by examining texture. Texture synthesis refers to using an analysis to make a similar synthetic texture that can be applied at will.

3.2.1 Statistical

Statistical texture refers to features that are primarily based on the overall statistical properties of the region in question. These features can be further subdivided into those that depend on the relative position of pixels; and the purely statistical ones that do not. The latter group are called first order statistics.

Texture-based classification is generally accomplished moving an $n \times n$ sliding window over image (or some transformation thereof) in steps of $n \leq m$ pixels. In this way the local region can be characterized. Smaller windows collect less data but provide higher resolution output, whereas larger windows can't differentiate fine details, but collect more reliable information. Filtering in the frequency domain, discussed below, can particularly benefit from a larger window size.

First Order Statistics

Since pixel position is irrelevant, another way of thinking of first order statistics is those that can be derived from the histogram. In the simplest case, each bin of the normalized histogram can be used as a feature, and in fact well designed local histogram features have shown very good results in many detection and classification tasks [1, 34, 41, 83]. Principal Component Analysis (PCA) has proven effective tool for reducing dimensionality of histogram and histogram-like features [131].

Local standard deviation provides a simple and effective measure of the general roughness of a region. The other most commonly used first order statistics are mean, variance, skewness, and kurtosis (Figure 3.9).

Heterogeneity is another 1st order statistic that has seen recent use [60]. It is defined as the fraction of pixels whose intensity is more than 10% different from the average. Rougher surfaces tend to have higher homogeneity values.

Higher Order Statistics

Gray Level Run Length Matrix (GLRLM) is a 2d matrix with one axis corresponding to gray level, the other to run length, with the value in each cell a count of the number of times this run type occurs. These gray levels and run lengths are usually quantized to

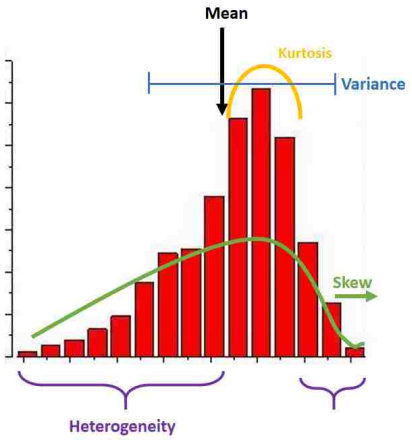

Figure 3.9: First order statistics can be derived from a histogram.

limit the matrix to a reasonable size in all matrices of this general type. If the image is different along the x and y directions, a separate matrix can be used along each direction. In these cases an angle between 0 and 90 degrees is used to characterize the run direction. A simple GLRLM is shown in Figure 3.10.

Image (quantized)

0	0	0	1	0
0	2	2	1	1
0	3	3	2	2
1	3	3	3	3
1	2	2	3	2
1	2	2	3	2

Run Length

Gray Level \ Dir: 0°	1	2	3	4+
0	3	1	1	0
1	4	2	0	0
2	2	3	0	0
3	2	0	0	1

Figure 3.10: Gray Level Run Length Matrix (right).

Alternatively, a run length vector is formed containing the number of runs of each length; irrespective of the brightness (Figure 3.11). These runs can and usually should have a similarity threshold instead of the coarse quantization of GLRLM.

	Dir: 0° Run Length			
	1	**2**	**3**	**4+**
# Runs:	11	6	1	1

Figure 3.11: Gray Level Run Length Vector.

A Gray Level Co-occurrence Matrix (GLCM) has both axes as pixel intensity values, and cells contain counts of number of times this pairing occurs at a given x,y offset (Figure 3.12). A separate matrix is needed for each x,y offset; (0,n) (n,0) (n,n) and (-n,n) with n=1 through 5 is an example of common offset values. The co-occurrence matrix is often added to its transpose to make it symmetrical assuming flipping the texture doesn't change its significance. It may also be normalized by the sum of its elements.

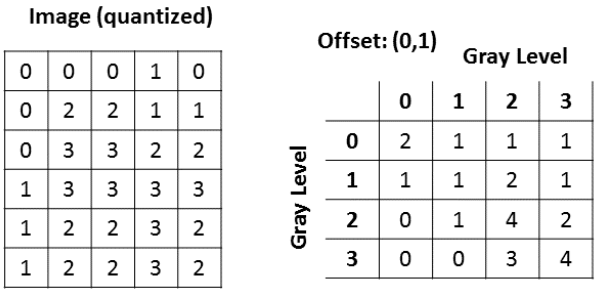

Figure 3.12: Gray Level Co-Occurrence Matrix (right).

Sum and Difference Histograms are a similar but less computationally intensive alternative to GLCM. Used together they provide almost as much discrimination as co-occurrence with lower memory and processor usage [175].

Neighboring Gray Level Dependence Matrix (NGLDM) is a 2d matrix where the axes are pixel values and number of similar pixels in their neighborhood; and the values are counts for each combination (Figure 3.13).

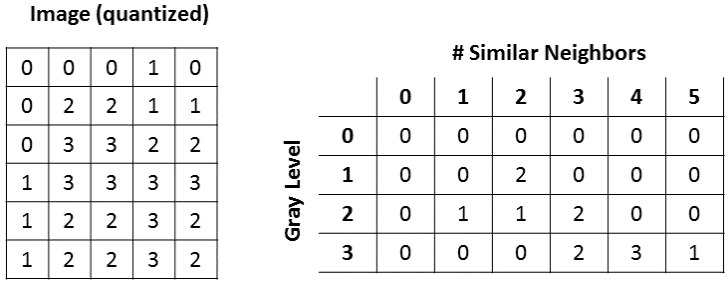

Figure 3.13: Neighboring Gray Level Dependence Matrix.

For Run Length Matrix (RLM), Color Co-occurrence Matrix (CCM), and NGLDM, once the matrix itself is completed, various statistical measures are computed to be used as the actual features. Notably, Haralick et al. [71] suggested the quintessential set of features for the CCM based on various statistical properties. These include contrast, correlation, variance, average, second moment, inverse difference moment, entropy, and other statistical properties.

Direct multivariate approach

This approach is to use a single one-dimensional vector of pixel values directly as features, applying Principal Component Analysis (PCA) or Partial Least Squares Discriminant Analysis (PLS-DA) to reduce dimensionality. Converting from 2 dimensions to 1 is accomplished by attaching rows back to back, or some more complicated unfolding. One major weakness of this approach is that spatial information perpendicular to the direction of unwrapping is lost [22]. Therefore the image is augmented with different altered versions of itself to form a multivariate image [22].

Angle Measurement Technique(AMT)

The Angle Measurement Technique (AMT) [6] was originally designed as a tool for characterizing geological lines. As shown in Figure 3.14, the average angularity at different

scales is plotted. The resulting plots are reminiscent of fractal analysis, but do not assume similar properties at different scales. Later the same method was used to analyze texture by converting it to a 1 dimensional function, then analyzing the angularity of the plotted function [55]. The conversion is accomplished by individually sampling rows/columns of the input, or "unwrapping" the image into a single long array along a snake or spiral path. The similarity of the resultant spectrum to the Fourier spectrum was pointed out, and it was proposed as a general purpose signal analysis. PCA has been shown to be effective in reducing the dimensionality of the resultant spectra.

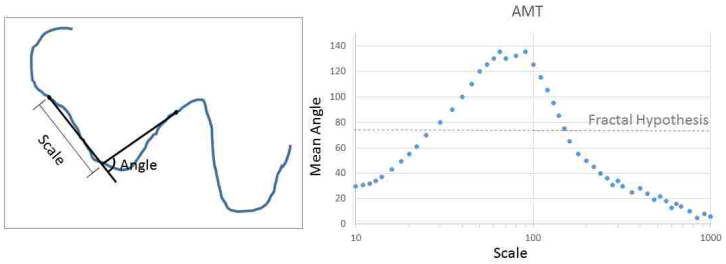

Figure 3.14: Angle Measurement Technique.

3.2.2 Structural

Structural texture analysis involves breaking down a texture into a repeating series of texture elements, the texture being defined in terms of those primitives [70]. The type and quality of these primitives combined with their spatial relation to each other describes the texture. A primitive can be a particular arrangement of pixels, but it can also be a group of pixels that share similar properties, for example, a connected set of pixels with similar intensity or edge direction [70]. A texture can be classified as weak or strong based on the degree of spatial relation between primitives. This can be quantified by the number of each primitive within some fixed range, some fixed angle, or the minimum distance to each type of primitive from the current primitive. An example of extracting groups of similar primitives from an image is shown in Figure 3.15.

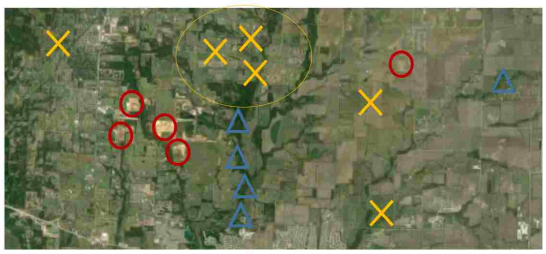

Figure 3.15: Extracting textural primitives from an image. Primitive frequency and clustering characterize texture.

Many of the structural methods that have been proposed only work well on regular textures, and they are rarely used in texture analysis [22]. Nevertheless, with well designed primitives that are statistical or extracted features rather than templates, it is a theoret-

ically sound way to recognize some types of textures. The same idea has been applied to part based and non-rigid object recognition with good results [34, 102].

3.2.3 Transform Based

These types of features are obtained by applying some type of transform to the image before using the result. After transformation, values may be used directly or some type of statistical method may be applied.

Convolution by Mask

The most basic type of transform is an image filtered by convolution by a mask. Convolution, designated by the $*$ operator, is defined as:

$$f(x) * g(x) = \int_{-\infty}^{\infty} f(\tau)g(x-\tau)d\tau \tag{3.14}$$

where f and g are 1d functions in space. In the case of an image, in 2 dimensions, the equation becomes:

$$f(x,y) * g(x,y) = \int_{\tau_1=-\infty}^{\infty}\int_{\tau_2=-\infty}^{\infty} f(\tau_1,\tau_2)\cdot g(x-\tau_1, y-\tau_2)d\tau_1 d\tau_2 \tag{3.15}$$

and for discrete digital images:

$$f[x,y] * g[x,y] = \sum_{n_1=-\infty}^{\infty}\sum_{n_2=-\infty}^{\infty} f[n_1,n_2]\cdot g[x-n_1, y-n_2] \tag{3.16}$$

When an image is convolved with a filter, f is the entire image, and g is the (usually much smaller) filter. An example of image convolution is shown in Figure 3.16, where a pair of circles are blurred by convolving them with an elliptical Gaussian mask.

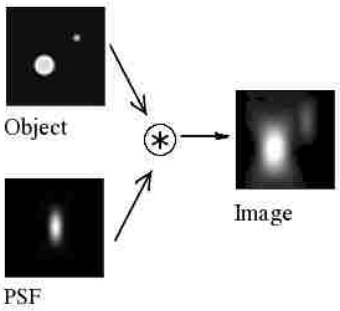

Figure 3.16: Convolution of image "object" by Gaussian mask "psf".

Convolution by the Sobel mask:

$$G_x = \begin{bmatrix} -1 & 0 & +1 \\ -2 & 0 & +2 \\ -1 & 0 & +1 \end{bmatrix} * P_{blur} \tag{3.17}$$

marks out vertical edges in the image, or horizontal edges if rotated 90 degrees (Figure 3.17). or the results of both transforms can be combined to mark out all edges. The

eigenfilter mask developed by Ade (1983) [203] is another type of mask, but is very computationally intensive. It relies on decomposing the training examples into a linear combination of sub-images. The most relevant subset of these sub-images, i.e., the set that can reproduce the original data with smallest error, are then used as filtering masks. Figure 3.18 shows a filter bank of the 4 most relevant filters for training data of faces. Using these filters on an image containing faces of the proper scale will create features that detect and classify faces.

Figure 3.17: lena.png (left), horizontal Sobel filtered (middle), and vertical Sobel filtered (right).

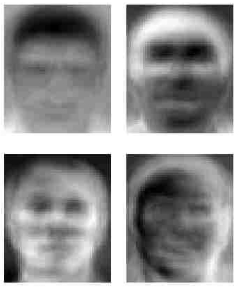

Figure 3.18: Eigenfilters computed from the ORL Face database(Eigenfaces.png, Source: Ylebru - Wikimedia Commons).

Frequency Transform Methods

The Fourier theorem states that all continuous periodic functions can be approximated by the sum of a series of sine and cosine functions:

$$f(x) = \frac{1}{2}a_0 + \sum_{n=1}^{N} \left[a_n \cos(nx) + b_n \sin(nx)\right] \tag{3.18}$$

where a_0, a_n, and b_n are constants that can be calculated easily for known functions. As the length of the series, N, approaches infinity, the approximation becomes arbitrarily close to the original function. As a practical matter some functions with discontinuities can be approximated as well, with minor artifacts that eventually vanish as N approaches infinity. An example of Fourier series approximation is shown in Figure 3.19. Non-periodic signals with finite limits (such as images) can be handled as well by treating them as one period in an imaginary larger periodic signal.

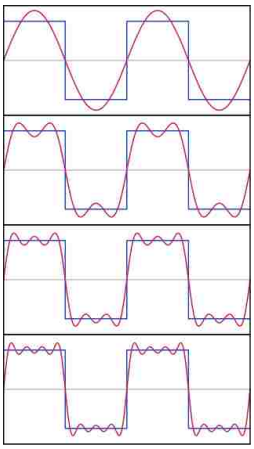

Figure 3.19: All periodic functions can be approximated by the sum of a series of sine and cosine functions.

The weights of the various sine/cosine functions used in these approximations can be used to characterize the "frequency spectrum" of the waveform (Figure 3.20). Lower frequencies correspond to larger, coarser shapes, and higher frequencies correspond to finer patterns, and are of course required to estimate sharp edges. This idea of frequency characterization and analysis has extremely broad applications, and is of particular interest for textural analysis.

The idea of frequency analysis can be easily extended to 2 dimensions and spatial frequency rather than temporal. With the assumption that the image is one cycle of an infinitely repeating, even function, equation 3.18 simplifies to:

$$f(x) = \frac{1}{2}a_0 + \sum_{n=1}^{N} a_n \cos(nx) \tag{3.19}$$

Furthermore since the NyquistShannon sampling theorem states maximum frequency that can be uniquely represented in a digital image is 1 cycle / 2 pixels, the maximum frequency N is far from infinite.

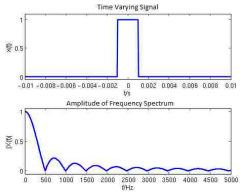

Figure 3.20: Frequency Spectrum.

Frequency Transform Methods generally perform a frequency transform on a local window, generating a 2d matrix. Then various properties of the resulting matrix are extracted as features, as with the CCM. The most basic are 2d fast Fourier transform and Discrete Cosine Transform (DCT).

Fast Fourier Transform and Discrete Cosine Transform - Digital images are not continuous functions, but rather arrays of discrete values, so equation 3.18 becomes:

$$F(x) = \frac{1}{2}a_0 + \sum_{n=1}^{N-1} a_n \cos\left[\frac{\pi n}{N}\left(x + \frac{1}{2}\right)\right] \tag{3.20}$$

where F is a discrete function, and N is the image width/height in pixels. This is the basic idea behind the discrete cosine transform; and the DCT can be taken over a localized region (commonly 4x4 or 8x8) by a simple linear filtering, although it may use a sliding window rather than calculate at every pixel. One weakness of approximating an infinite sine wave by a filter that cuts off sharply after 4 or 8 pixels is that this causes boundary artifacts and poor convergence at the boundaries [176]. A bank of DCT filters is shown in Figure 3.21.

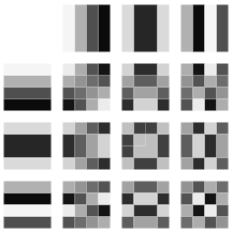

Figure 3.21: Discrete Cosine Transform Filters.

The FFT is usually taken over a larger region - for example [22] uses the magnitude of the 2D Fast Fourier Transform (2DFFT) on entire 479 × 508 pixel images of steel surfaces to characterize their texture. In this case the FFT magnitudes are filtered by a Gaussian kernel with $\sigma = 0.5$. The FFT is generally used to characterize frequency over a wide area, or an entire image for compression purposes, so it is not very common in modern texture analysis, where well localized information is preferred. And example of the FFT magnitude for a 184 × 184 image is shown in Figure 3.22. The center represents low frequency components. It can be seen from this example that most real images have a cluster of higher weights near low frequency "DC" values, and sometimes lines of higher magnitude along the x/y axes and along the directions of significant textures in the image.

Figure 3.22: Fast Fourier Transform Results.

Gabor Transform and Wavelet Transforms - The Gabor transform is a variant of the Fourier transform that analyzes the phase and frequency content of a localized region only. A Gabor filter is similar to a DCT filter in a single direction, but it is multiplied by a Gaussian, so its maximum magnitude falls off gradually rather than cutting off suddenly at the filter boundary (Figure 3.23). This soft cutoff eliminates the kind of boundary artifacts that happen with DCT. Image analysis by Gabor transform is believed to be similar to the

workings of the human visual system, and has been applied to things like Chinese character recognition and face recognition. Additionally, Gabor features have been widely used in texture based segmentation [22].

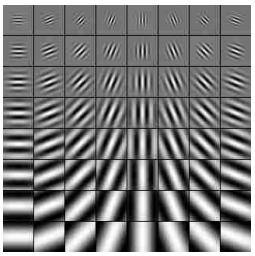

Figure 3.23: Gabor Filters.

Wavelet transform is a more advanced method, similar to Gabor transform (which is often considered a subset wavelet) but with different filters. These filters are generated in such a way as to have various desirable mathematical properties, while still sampling localized frequency-type information. The normalized energies of the wavelet coefficients can be used directly as features. Other popular features are entropy and averaged L1-norm [22]. The wavelet transform is superior to the Gabor transform in that its properties allow it to have variable resolution in the frequency domain. In real images, the vast majority of frequency information is localized in a few parts of the frequency domain (in fact this observation is the basis of jpeg compression) so this is a very useful feature. Additionally many wavelet transforms maintain spatial and frequency localization better when discretized than Gabor features [22]. Because of these advantages and their competitive performance, Wavelet Texture Analysis (WTA) methods are considered to be the state of the art in texture analysis. A simple example of a 1d wavelet is shown in Figure 3.24.

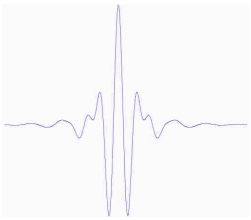

Figure 3.24: A Meyr Wavelet.

3.2.4 Model Based Approaches

Model based texture recognition involves creating a texture model, with some parameters, then finding the parameters that best approximate the data. The biggest advantage of these methods are that they allow easy generation of synthetic textures.

Autoregression Model

The basic autoregression model assumes each pixel value is some linear function of its neighbors plus random noise/error:

$$I_p = \sum_{q \in N} k_r I_q + e_p \tag{3.21}$$

where p and q are pixels in the same neighborhood N in image I, k_r is a parameter associated with the relative position r of q with respect to p, and e_p is noise in the result. So for example if neighbors are defined as the 4 adjacent pixels to p, there are 4 values of r.

These 4 parameters plus the noise level can be calculated by finding the parameters that minimize the MSE:

$$\sum_p e_p^2 = \sum_p \left(I_p - \sum_{q \in N} k_r I_q \right)^2 \tag{3.22}$$

These model parameters can be used directly as features for machine learning [167].

Markov Random Fields

A Markov random process is a process which can be modeled by a Markov chain [24]. A Markov chain is a memoryless process where the probability of an event depends only on the current state, not on any previous states. A graph representing a Markov chain is shown in Figure 3.25; each circle represents a state, and each arrow is a state transition. More complex nth order Markov chains can depend on the last n states instead [24].

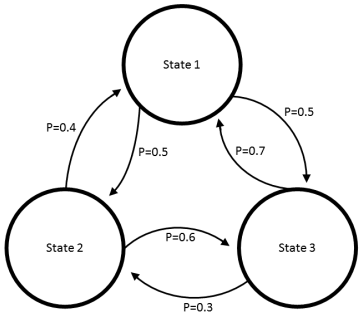

Figure 3.25: A simple Markov chain.

In the simplest case, one dimensional textures, or textures where rows are assumed to be independent, have been modeled directly by Markov chains [42]. In this case, each state represents a pixel value, and transitions correspond to the likelihoods of the next value, based on one or more previous values. The idea can be extended to two dimensions by basing the probable pixel value on the value of multiple immediate neighbors rather than multiple previous values. The direct 2 dimensional extension of a Markov chain bases pixel value off of the values of half its neighbors, and uses it to help determine the values of the other half. This is called a Markov mesh (Figure 3.26). An dependence of this type where each value can be successively generated from previously generated ones in order is called causal.

For modeling image textures, it is more logical and useful to replace the Markov mesh with an undirected graph [24], where the value of every pixel is dependent on the values of all of its neighbors:

$$L_p|P = L_p|N_p \qquad p \in P \tag{3.23}$$

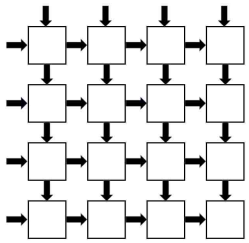

Figure 3.26: A Markov Mesh.

where L_p is the label (value) of pixel p, N_p is the neighborhood of p in all directions (excluding p), and P is the entire image. This is a non-causal model known as a Markov random field. Homogeneous textures can be defined as those where a pixel's probable value is dependent only on the local neighborhood [43]; these are the types of textures a Markov random field can model. The parameters of the model that best models a texture can be used as feature vectors for classification [38].

Fractal Models

The definition of a fractal shape or process is one that is self similar at different scales. Many real world objects do in fact exhibit similar properties when examined different scales, over many orders of magnitude [141]. Some natural textures, which are poorly described by classical geometry, can be described by fractal models [94, 124].

For example, the fractal properties of surface roughness can be used to characterize a texture [141]. A random function $I(x)$ is called a fractal Brownian function if

$$\mathrm{Var}(\Delta I) = \sigma^2 |\Delta x|^{2H} \tag{3.24}$$

where Δx is any distance in pixels, ΔI is the intensity difference at that offset, and H is a constant called the Hurst parameter, and $0 < H < 1$. The fractal dimension for a 2 dimensional function or curve is

$$D = 3 - H \tag{3.25}$$

This fractal dimension is a useful and scale invariant feature that can be used to measure the "roughness" of real world textures in a useful way - as a practical matter equation 3.24 holds true for uniformly lit homogeneous fractal surfaces [141]. Of course real images are not fractal at all scales, so we need to pick upper and lower limits for Δx. Fortunately, equation 3.24 can be used to measure the suitability of fractal dimension to characterize a texture, as well as to actually characterize it. The Angle Measurement Technique of section 3.2.1 uses a similar idea, but makes no assumption of self similarity.

Chapter 4
Food Image Analysis for Measuring Food Intake in Free Living Conditions

4.1 Introduction

Nutritional research is of particular practical interest in the modern world in light of the still not fully understood effects of heavily manufactured foods on health and the growing prevalence of obesity worldwide. Hence, there is an abundance of active research along these lines, a key component of which is efficiently and accurately measuring food intake. The most accurate method of measuring intake is the energy balance method, where total daily energy expenditure (TDEE) is measured with doubly labeled water. This method is prohibitively expensive in terms of materials and expertise, however, and does not quantify nutrient intake. The digital photography of foods method has been shown to be accurate in cafeteria settings [189], and the RFPM is accurate in naturalistic/free living conditions [119, 120]. When using these methods, trained human raters estimate food intake based on these images but the process is labor intensive, with the most time intensive task being food identification, which necessarily precedes identifying a match for foods in a nutrient database for calculation of nutrition information.

This chapter presents an automated image analysis application that takes digital food images as input and provides food identification and intake/nutrition information as output. It makes use of a large database of sample images with known weights and nutritional data to compare against. The application consists of a number of image processing modules, including segmentation, pattern classification, and volume estimation. Ideally it is fully automated, and in partially automated mode it greatly simplifies the food identification and volume estimation processes. The fully automated mode is primarily discussed in this chapter.

Our food recognition system is intended to be a complete system, potentially fully automated from beginning to end. i.e., food snapshot in, nutritional intake out. It also necessarily needs to allow for manual intervention at any point along the process. For our system, food images are taken before and after meals, and the difference is used to estimate nutritional intake.

As discussed in sections 1.1.1 and 2.1.3, there are 3 major problems associated with food image analysis. Firstly, segmentation of multimode foods (single foods with multiple segments with different properties). This is difficult because in order to segment these type of foods into only one segment, classification must be done during or before segmentation. But for large numbers of food types, classification becomes prohibitively hard without prior segmentation. Secondly, computationally feasible classification with thousands of classes, with multiple membership allowed, and a meaningfully sorted list of runners up for classifications. This is difficult because for most machine learning algorithms, either computation time grows exponentially with the number of classes, or only the most likely

(first place) classification is meaningful. The third problem is finding some method of volume estimation. This requires us to somehow guess 3d information from a 2d image, or come up with a very cheap and user friendly way of 3 dimensionally scanning an object without any equipment beyond an average cell phone. Our system solves the second and third problems. For the first, it segments training images automatically, but still requires simple manual segmentation for testing images.

4.2 Contributions

This system makes several contributions. It is the most advanced published complete system of its type. It has significant and more extensive data on 2d food volume estimation than any study of its type, and compares the results against those of trained human analysts. It presents a novel food recognition scheme, which provides multilabel classification on a large number of classes parallelizably, in linear time with number of classes, and generates meaningful runners up. Much of research [118, 125, 144, 145, 205, 207] in the area does not meet these requirements, which are absolutely necessary to solve the problem. To the best of our knowledge, at the time it was published our food recognition scheme was the most advanced and accurate system of its kind that fulfilled these requirements. Our system was published in the Proceedings of the SPIE on Medical Imaging [51].

4.3 Methods and Algorithms

Our algorithm is comprised of four major portions, preprocessing, segmentation, classification, and volume estimation. During preprocessing the image's color, scale, and orientation are analyzed and normalized. During segmentation, the part of the (normalized) image containing food is detected. During classification, the food region is compared to known foods in a database and assigned a class. Finally in the volume estimation step, the surface area is cross referenced with database info to estimate the food mass, which can be used with the database entries to give nutritional data. A high level block diagram of the process is shown in Figure 4.1.

We want our system to be as automatic as possible, and to smoothly integrate any necessary user corrections into the process. This means giving meaningful feedback after each processing step. Failures during the preprocessing, segmentation, and volume estimation steps can be corrected straightforwardly (a few mouse movements or keystrokes), so their algorithms are not affected much. However, in order for the classification step to support simple user intervention it needs to provide a list of the most likely alternative classifications that a user can take in at a glance.

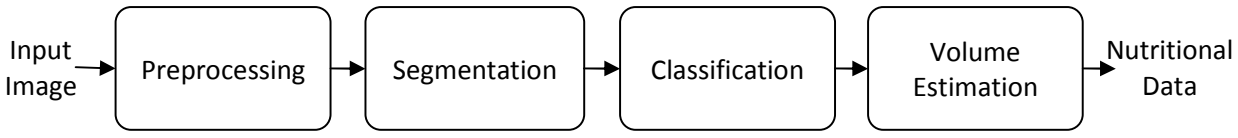

Figure 4.1: High level block diagram of automated food photo analysis.

4.3.1 Preprocessing

There are a two main preprocessing tasks that are needed. First, we need to normalize the scale and orientation of the image. Second, we need to normalize the color differences due to lighting.

Scale/Orientation

In anything but laboratory conditions, the distance and angle of the viewpoint vary from image to image. It is necessary to know the scale and orientation of a given image to estimate the food volume based on it. Accurate scale is also necessary for the feature extraction that precedes the machine learning used for the classification step, excepting scale invariant features. One possibility is to use camera metadata, but a given camera may not have it, and it may be inaccurate. The other possibility is to use an object of known dimensions in the image as a reference.

The most common approach is to use a patterned reference card [118]. Several authors [145, 205, 207] use a card with multicolored squares on it, which can also be used for color normalization, but they require the user to manually click on each card corner. A black and white bullseye patterned reference card can be detected automatically [121]. A black and white card can be used to normalize black and white levels, and white balance; so it still provides basic color normalization.

Alternatively, a plate of known dimensions can be used as a reference [80]. Ellipse detection is a solvable problem, so automatic detection is possible, and the fact that plates are circular and lie flat against a table makes them ideal for estimating orientation and scale. In a free living environment, a given user might use the same plates every meal; but not reliably. But in a cafeteria environment, where much meal research is done, plates of standardized dimensions are the norm.

More sophisticated volume estimation methods use multiple images for 3d modeling [50, 145], with results similar to or slightly better than human perception. They are more accurate than 2d estimation, and may even be more accurate than human perception. They still require the same reference card or object discussed above.

For our system we chose to use the bullseye patterned cards of [121] because of the ease of automatic detection, and alternatively use a plate of known dimensions in cases where a card is not available. The volume estimation accuracy that can be achieved is only a little worse than human perception [45, 119, 155, 188, 190], and it minimizes the amount of interaction required by the user taking the photos or reviewing them.

Automatic reference card detection is achieved using the method of [97], with a reference card like the one depicted in Figure 4.2. First the image is binarized, with local thresholding to account for variance in local lighting conditions. This is accomplished by using a square averaging filter to find the local brightness, and subtracting this from the original image. In the output, pixels with positive values are brighter than the local average and become white, and those with negative values become black. They then use a pattern detection algorithm that produces maximal response near the center of areas with mirror symmetry and alternating color. In real-world pictures, the two strongest such responses in a given photo are usually the patterns on the card. Once these 2 peaks are located, simply connecting them with a line segment and using that segment as the seed pixels for a region

fill on the binary image will produce a single segment for the entire card. The segment is smoothed by Gaussian smoothing to get rid of any rough edges, and at that point Harris corner detection [73] can be applied to find the 4 corners.

Figure 4.2: Feature rankings.

Our ellipse detection is via a public implementation of the efficient ellipse detection of [195]. It considers every pair of edge points as possible ends to the major axis of an ellipse, and scores these possibilities by Hough transform [77]. In short, The Hough transform builds a parameter space for the object to be matched. For an ellipse centered at $\langle i, j \rangle$ with axis radii a and b and rotated to angle θ:

$$\frac{((x-i)\cos(\theta)+(y-j)\sin(\theta))^2}{(a^2)}+\frac{((x-i)\sin(\theta)-(y-j)\cos(\theta))^2}{(b^2)}=1 \tag{4.1}$$

where x and y are the coordinates of the point on the ellipse. There are 5 parameters, so each pixel in the image generates a hyperplane in a 5 dimensional space corresponding to every possible ellipse that it could lie on. This is used in a voting procedure - multiple hyperplanes are accumulated and summed in the Hough space, and regions where many planes intersect are maximized and correspond to good matches. Needless to say, it is helpful to limit the range of parameters to keep the 5 dimensional accumulator space as small as possible. Even simplified, ellipse detection is processor and memory intensive, so to further increase efficiency we perform a multiscale pyramidal search similar to the manner of [26], and use results at lower resolutions to sharply limit parameters at higher ones.

The details of the scaling and geometric transformation are discussed in section 4.3.4, since aside from potentially using scale information to improve classification, it is primarily a volume estimation issue.

Color Normalization

There are several common methods for automatic color normalization in photographs. The naive "gray world" algorithm assumes the global average of red, green, and blue pixels in an image will be the same - for an image P with 3 color channels r, g, and b:

$$P_{avg}=\frac{\bar{P}_r+\bar{P}_g+\bar{P}_b}{3} \tag{4.2}$$

$$P_{rgb}(x,y)'=\left\langle \left(P_r(x,y)+(P_{avg}-\bar{P}_r)\right),\left(P_g(x,y)+(P_{avg}-\bar{P}_g)\right),\left(P_b(x,y)+(P_{avg}-\bar{P}_b)\right)\right\rangle \tag{4.3}$$

where x and y are individual pixel coordinates. The naive "white patch" algorithm assumes that the brightest area in a given photo is white:

$$P_{gray}(x,y) = (P_r(x,y) + P_g(x,y) + P_b(x,y)) \tag{4.4}$$

$$\langle i,j \rangle = argmax(P_{gray}) \tag{4.5}$$

$$P_{rgb}(x,y)' = \left\langle \left(P_r(x,y) \times \frac{255}{P_r(i,j)}\right), \left(P_g(x,y) \times \frac{255}{P_g(i,j)}\right), \left(P_b(x,y) \times \frac{255}{P_b(i,j)}\right) \right\rangle \tag{4.6}$$

where x and y are individual pixel coordinates, and $\langle i,j \rangle$ are the coordinates of the brightest pixel in the image. More sophisticated methods try to guess which areas are gray/white by various methods [197], or maximize contrast on each color channel separately [185], or a mixture of both these methods [169]. Another approach is to find color balance under different lightings, then require a human user to estimate the lighting type [63] or try to predict the lighting type from the correlation of image features [39, 99].

As a practical matter, there is a limit to the estimation quality without a known reference object, or at least scene context to know what is actually white. So for professional digital photography it is common to use a white reference object [132], or a multicolored reference card [140]. At each visible wavelength λ, the measured brightness depends on lighting, the object's color, and the sensor's spectral response. Biological or silicon, a sensor does not detect specific wavelengths, but sums the energy according to its spectral response such that:

$$C(measured) \propto \int_{\lambda=0}^{\infty} C_\lambda(lighting) \cdot C_\lambda(object) \cdot C_\lambda(sensor) \tag{4.7}$$

where $C(measured)$ is the amount of energy of a certain color measured, aka. brightness. $C_\lambda(lighting)$ is the spectral energy of the light source at a given wavelength λ, $C_\lambda(object)$ is the spectral characteristics of the object (what colors of light it reflects), and $C_\lambda(sensor)$ is the spectral response of the sensor(how much the voltage changes per unit of brightness at the given λ). Because of this, for light sources such as fluorescent lighting that have narrow spikes in their spectral response (Figure 4.3) are particularly problematic, as they attenuate most of the spectral response of objects in that image, making it hard to estimate what they might look like under white (spectrally flat) lighting. So color correction is always an estimate, although multiple reference colors with known spectral responses should be able to improve it in many situations.

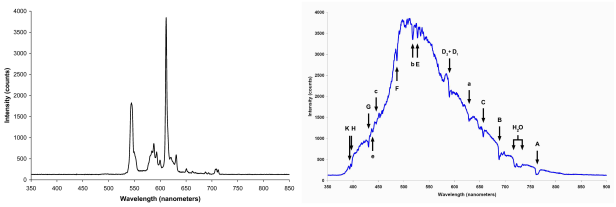

Figure 4.3: Yellow fluorescent light spectrum.png(left), Spectrum of blue sky.png(right) (Source: Deglr6328 - Wikimedia Commons)

In [121], the reference card is black and white, which can serve as a base to normalize color channels. Since these cards can be printed out simply, must be used for scale

estimation anyway, and color balanced from a white reference often gives good results, we continued using this method. With black and white levels the colors can be balanced such that:

$$P_r(x,y)' = \Big((P_r(x,y) - \mu_{B_r} + \sigma_{B_r}) \times \frac{255}{\sigma_{B_r} + \mu_{W_r} - \mu_{B_r} + \sigma_{W_r}})\Big) \tag{4.8}$$

$$P_g(x,y)' = \Big((P_g(x,y) - \mu_{B_g} + \sigma_{B_g}) \times \frac{255}{\sigma_{B_g} + \mu_{W_g} - \mu_{B_g} + \sigma_{W_g}})\Big) \tag{4.9}$$

$$P_b(x,y)' = \Big((P_b(x,y) - \mu_{B_b} + \sigma_{B_b}) \times \frac{255}{\sigma_{B_b} + \mu_{W_b} - \mu_{B_b} + \sigma_{W_b}})\Big) \tag{4.10}$$

Black and white pixel segments B and W are found from the local thresholding described for reference card detection in section 4.3.1 above. Their means and standard deviations are used to white balance the 3 channels r, g, and b in the image P.

Other preprocessing

Additionally, it might be necessary to do image enhancement, such as denoising, deblurring, sharpening, or contrast enhancement [53]. However, since modern digital cameras are of good quality and take very high resolution images, these are used rarely except in archaic systems.

4.3.2 Segmentation

If a single food has areas with different colors/textures (which many do [16]), then standard approaches to segmentation are unlikely to work, and segmentation by classification is required for optimal performance. This creates a chicken and egg problem, where either one problem must be solved suboptimally or some iterative approach must be taken. This is discussed in more detail in section 2.1.2.

We experimented with segmentation by classification, and it worked well for small numbers of food types. As the number of food types got larger, we used median filtering, which is a nonlinear filter, and relatively slow compared to linear filters like Gaussian blur or averaging. As we added even more food types, this broke down as well. With a region matching a dozen or more mutually inclusive or nearly identical food types, each pixel cannot be classified separately; connectedness becomes a big issue. A possible solution would be to solve some sort of optimization problem, perhaps setting it up as a graph cuts problem, like the segmentation described in section 3.1. The classification result for each food type would be treated as an image channel. Unfortunately, a high resolution image with not 3 but thousands of channels is difficult to even hold in memory, and an optimization with that many classes would be computationally unfeasible. Nevertheless, trying this approach with some simplifying assumptions and a multiclass Graphcut algorithm might be a good direction for future work. In the meantime, we choose to segment and classify separately, with segmentation done first.

We detect the edges in the image via Canny edge detection, which consists of Gaussian blurring, followed by Sobel edge detection along both axes, giving gradient images:

$$P_{blur} = g(\sigma) * P \tag{4.11}$$

$$G_x = \begin{bmatrix} -1 & 0 & +1 \\ -2 & 0 & +2 \\ -1 & 0 & +1 \end{bmatrix} * P_{blur} \tag{4.12}$$

$$G_y = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ +1 & +2 & +1 \end{bmatrix} * P_{blur} \tag{4.13}$$

where $g(\sigma)$ is the 2d discrete Gaussian function used for blurring, and P is the image. The images are combined into a gradient magnitude image and a gradient direction image by:

$$G = \sqrt{G_x^2 + G_y^2} \tag{4.14}$$

$$\Theta = atan2(G_x, G_y) \tag{4.15}$$

Next, non-maximum suppression is applied perpendicular to edge direction to thin edges to around one pixel wide, and edges detected in anomalous (low gradient) regions are removed. The rest of the edges are rated weak or strong depending on gradient intensity, and strong edges plus weak edges connected to strong edges form the final edge map.

Once the edges are detected we detect the plate. Ellipse detection (discussed in section 4.3.1) on the edge map is used to locate the plate in the scene [158].

We assume the plate is white, which is usually true, and that there is only one food per plate, which is true for our training database but not in actual meals. We experimented with k-means segmentation [164], but not knowing the exact number of foods in a meal in advance, picking a reasonable k was a problem. As noted above, multimode foods are hard to segment without classification. Methods that set k automatically still require a threshold to be chosen, and over segment multimode foods even when optimal. In any case, for multiple food segmentation the user manually marks the general regions for GrabCuts segmentation.

We use simple thresholding to separate plate pixels from food pixels. This initial segmentation is very poor, and is used as an initial guess for GrabCuts [149] to do the final, detailed segmentation. The OpenCV [30] implementation of GrabCuts allows the user to define initial conditions with 4 classes of pixels: known foreground and background pixels, and probable foreground and background pixels. The algorithm then iterates and returns improved classifications for the pixels. Figure 4.4 illustrates the segmentation process.

4.3.3 Classification

Ambiguous food classes, foods that belong to multiple classes, and classes that overlap with or are subsets of other classes are common. So user corrections aside, we require a classification scheme that allows for multiple class membership and provides a meaningfully sorted list of most likely classes. Since there are many, many kinds of food, our classification algorithm also needs to be applicable to a very large if not open ended number of classes.

We originally tried standard multiclass ANN with backpropagation, but it stopped working after we reached a total of around 30 classes. We experimented with an imple-

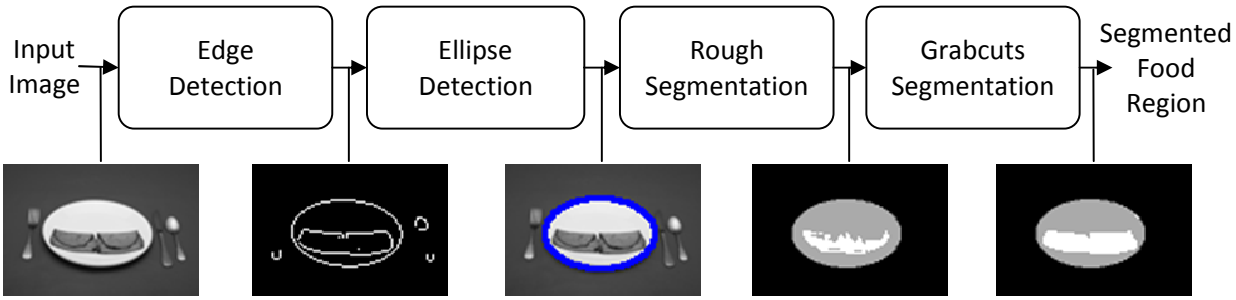

Figure 4.4: Segmentation.

mentation of multi class SVM as well, but it became computationally unfeasible even more quickly. There are much better multiclass SVM implementations than the one we tried, but theoretical and experimental analysis does show that even under the best conditions, with supercomputing capability and the best algorithms, SVM are not well suited to very large numbers of classes [111].

We also observed that the standard way of class scoring in inherently multiclass classifiers does not provide meaningful runners up and account for different but visually identical foods - requirements for large food datasets. The reason why is obvious after a brief examination of how standard multiclass classification works in neural networks. Feature vectors are fed into a network, and it produces an output vector:

$$\mathbf{F} = \langle o_1, o_2, o_3, ..., o_C \rangle \tag{4.16}$$

where C is the number of food classes. The target vector for a sample of food class c is:

$$\mathbf{T}_c = \langle o_1, o_2, ..., o_c, ..., o_{C-1}, o_C \rangle = \langle 0, 0, ..., 1, ..., 0, 0 \rangle \tag{4.17}$$

with zeros everywhere but at the output associated with that food class. To classify an arbitrary feature vector we find the closest target vector by minimizing the MSE $\|\mathbf{F} - \mathbf{T}_c\|$. Similarly, the training process for any type of classifier seeks to minimize the MSE between the training samples and the targets for their known labels, albeit under other classifier specific constraints. Consider the case where a sample is similar to multiple classes - to get a sorted list, we want a feature vector with multiple high values like:

$$\mathbf{F} = \langle 0, .8, .6, .9, .95, .5, 0, 0, .7, 0 \rangle \tag{4.18}$$

The correct classification, class 5:

$$\mathbf{T}_5 = \langle 0, 0, 0, 0, 1, 0, 0, 0, 0, 0 \rangle \tag{4.19}$$

gives an MSE of 1.60, and an incorrect classification of class 4 gives an MSE of 1.63. A target where only the most likely class has a high value such as:

$$\mathbf{F} = \langle 0, .1, .1, .1, .95, .1, 0, 0, .1, 0 \rangle \tag{4.20}$$

gives an MSE of 0.23 for the correct classification, an MSE of 1.32 for incorrect. So minimizing the MSE on a standard multiclass classifier incentivizes "guessing" only the most likely class and setting all other outputs near zero; so it is fundamentally incompatible with runners up. The problem only becomes more dominant with large numbers of classes.

We considered several different designs for our new classifier. Although SVM classifiers are inherently binary (2 class), they have many multiclass extensions. They achieve this by combining ensembles of binary classifiers [76]. These fall into 3 major groups, one-vs-all, one-vs-one, and hierarchical trees. These methods are also applicable to the general problem of multiclass classification. One-vs-one seemed likely to become unmanageable with large number of classes since for n food types it requires $n(n-1)/2$ individual classifiers, giving $O(n^2)$ if these classifications become the dominant part of the algorithm. A naive hierarchical tree seemed unlikely to be able to produce a sorted list of runners up, since a food type likely to match can be eliminated early on if it happens to be on the node next to a more likely one. A more reasonable hierarchical implementation via clustering is discussed in chapter 7.3.

After considering various designs, we settled on our One Versus The Rest (OVTR) approach, since it should work in time linear with number of classes and could provide good confidence scores for memberships in multiple classes. It also largely handled the problem of classifying multimode foods; its useful properties are detailed in the discussion in section 4.5.3.

Classifier Overview

Texture-based classification is generally accomplished moving an $n \times n$ sliding window over an image (or some transformation thereof) in steps of m pixels such that $m \leq n$. Feature vectors are then known at every window center, and can be estimated elsewhere by a nearest neighbor approach.

Our system uses a block-based feature extraction, where $n = m = 8$. To avoid training from background, only windows completely inside the food region are used. A feature vector is generated for each 8x8 square inside each region, as shown in Figure 4.5. This only includes squares that are fully inside the region; areas intersecting the region boundary are not used for either training or classification. Rather than create a monolithic classifier to distinguish between hundreds of foods, one "specialist" neural network is assigned to recognize each food. All 8x8 patches inside regions of interest are examined by each specialist network, as shown in Fig 4.5. Each specialist network looks at all feature vectors, giving a verdict of "positive" or "negative" for each patch. The number of positive votes by each specialist network is counted and the results are sorted. The most likely classification is the one that gave the most votes, as shown in Figure 4.6. The classification process is shown in Figure 4.7 and algorithm 3.

S is the set of food segments in the image, V is an accumulator array of votes for each of F food types, w is the current window, $f()$ is a feature extraction algorithm, and c is an array of binary classifiers, one for each food type.

Classifier Training

Our individual specialist classifiers are simple backpropagation networks, with 2 hidden layers with 10 neurons each. We used Levenberg-Marquardt backpropagation training

Algorithm 3 "One vs. the Rest" food classification

for Each Food segment S_z **do**
  Let votes V_1 to $V_F := 0$
  for $i := 1$ **to** $\max(x) + 1 - m$ **step** n **do**
    for $j := 1$ **to** $\max(y) + 1 - m$ **step** n **do**
      Window $w := P(i : i - 1 + m, j : j - 1 + m)$
      if $w \subset S_z$ **then**
        Extract Feature vector $fv := f(w)$
        for Each Food Type $f \in F$ **do**
          Classify with c_f: $match := c_f(fv)$
          if $match = true$ **then**
            Add votes: $V_f := V_f + 1$
          end if
        end for
      end if
    end for
  end for
  Classify segment: food type $T_z := f \mid V_f = \max(V)$
  Sort and store V_f for later use
end for

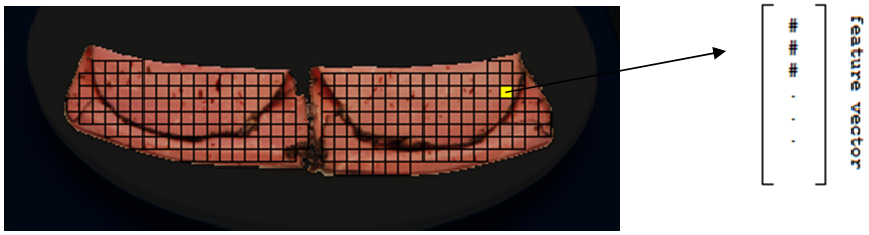

Figure 4.5: Feature vector generation.

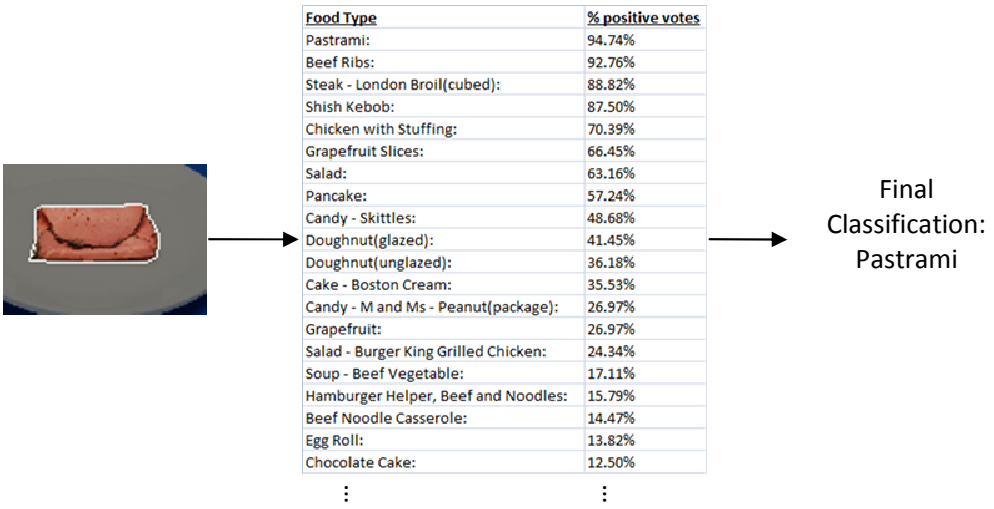

Figure 4.6: Classification results.

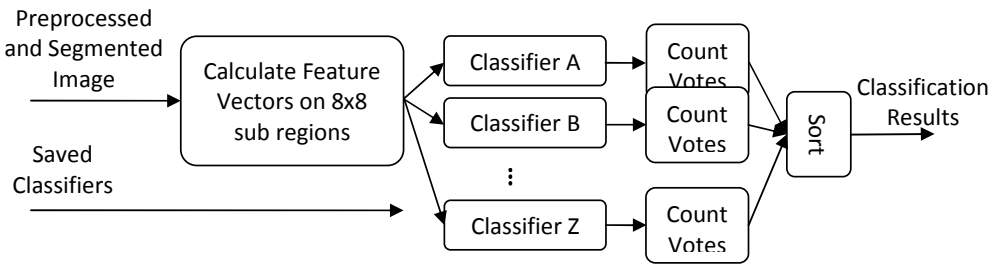

Figure 4.7: OVTR Classification.

function. Our learning function was Gradient descent with momentum weight and bias. For each food type, random examples are gathered from the training set, and an equal number of negative examples are picked at random from among other types. A negative example is a (preferably statistically representative) example of anything that is not a member of the target class. Our batch size is currently limited to around 600 feature vectors to speed up processing; half positive and half negative examples. The output layer of each specialist contains a single neuron responsible for making a true/false determination of whether the data matches its food type. During training, the target output values are set to 1 for positive examples and 0 for negative. The training process is shown is shown in Figure 4.8 and algorithm 4.

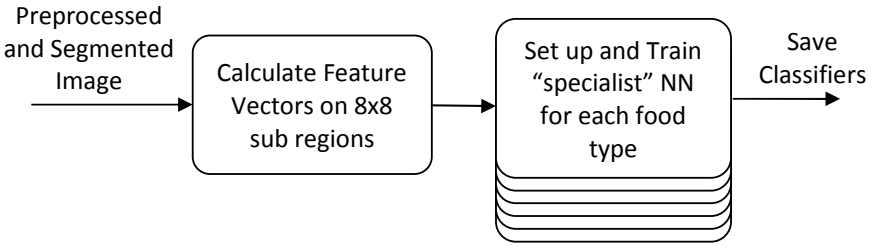

Figure 4.8: OVTR Training.

Feature Selection

We used Haralick features [72], which are extracted from the Color Co-occurrence Matrix (CCM) for an image block rather than the block itself, and include contrast, correlation, variance, average, second moment, inverse difference moment, entropy, and a number of others. In addition to Haralick, we used various other standard features including mean, 2nd moment, variance, standard deviation, and 2 dimensional Discrete Cosine Transform (DCT). We also experimented with histogram features and Gabor features, but they weren't used in our final system.

Two dimensional DCT on an 8x8 region produces an 8x8 DCT matrix; the features "dct01-04" each represent the mean of a 2 pixel wide band of this matrix, each representing progressively higher frequency information. The Color Co-occurrence Matrix (CCM) were taken over either hue, saturation, or intensity channel, are 16x16, and use an offset of (0, 1).

In addition to standard textures, which colors occur together is particularly important for classifying foods. In light of this we applied these texture features over each of the H, S, and I channels, in the manner of the food recognition scheme of [32].

Algorithm 4 "One vs. the Rest" training

```
Let length of training data l := 0
for Each Training Food segment S_z do
    for i := 1 to max(x) + 1 − m step n do
        for j := 1 to max(y) + 1 − m step n do
            Window w := P(i : i − 1 + m, j : j − 1 + m)
            if w ⊂ S_z then
                l = l + 1
                Extract Feature vector v_l := f(w)
                save food type of sample types_l = type(S_z)
            end if
        end for
    end for
end for

for Each Food Type f ∈ F do
    for i := 1 to I step 2 do
        pick a random k | types_k ≡ f, v_k ∉ d
        Training data d(i) := v_k
        Target t(i) := 1
        pick a random l | types_l ≢ f, v_l ∉ d
        Training data d(i + 1) := v_l
        Target t(i + 1) := 0
    end for
    train(c_f, d, t)
end for
```

In order to reduce the dimensionality of the feature space, some feature removal or dimensionality reduction scheme is required. We used feature ranking to remove less discriminative features. We experimented with perturbation analysis, with a method similar to but less sophisticated than [209], the idea being to make small perturbation in the classifier inputs and see how much the outputs are affected. Presumably, inputs that don't affect the outputs much are less significant and can be removed. But in practice it didn't seem to give good results, many weak but collectively valuable features tended to get ignored, so we used another method.

The features were ranked with an absolute value two-sample t-test with pooled variance estimate (Student's t-test) [44,110,172] (via the default option of the MatLab rankfeatures command). The t-test measures the likelihood that 2 distributions are distinct. For our purposes, these distributions are the distributions of some feature of interest for 2 different food types. Pooled variance makes the simplifying assumption that the variance of the 2 distributions is the same. The significance of the difference between the distributions is measured by the t-statistic:

$$t = \frac{\bar{X}_1 - \bar{X}_2}{\sigma_P \cdot \sqrt{\frac{1}{n_1} + \frac{1}{n_2}}} \tag{4.21}$$

where X_1 and X_2 are the feature distributions for the 2 food classes, n_1 and n_2 are the sample sizes, and σ_P^2 is the pooled variance:

$$\sigma_P = \sqrt{\frac{(n_1 - 1)\sigma_{X_1}^2 + (n_2 - 1)\sigma_{X_2}^2}{n_1 + n_2 - 2}} \tag{4.22}$$

The same data from the training set was used for the t test. Figure 4.9 shows the initial feature rankings. Each food had a separate classifier with a feature quality for each feature. The maximum quality of each feature across all foods was used, on the assumption that some features might be well suited to only a few foods. This method gave more meaningful rankings than perturbation and improved the Mean Squared Error (MSE) from .035 to .030 over basic HSI plus local std. The final feature list is shown in Table 4.1.

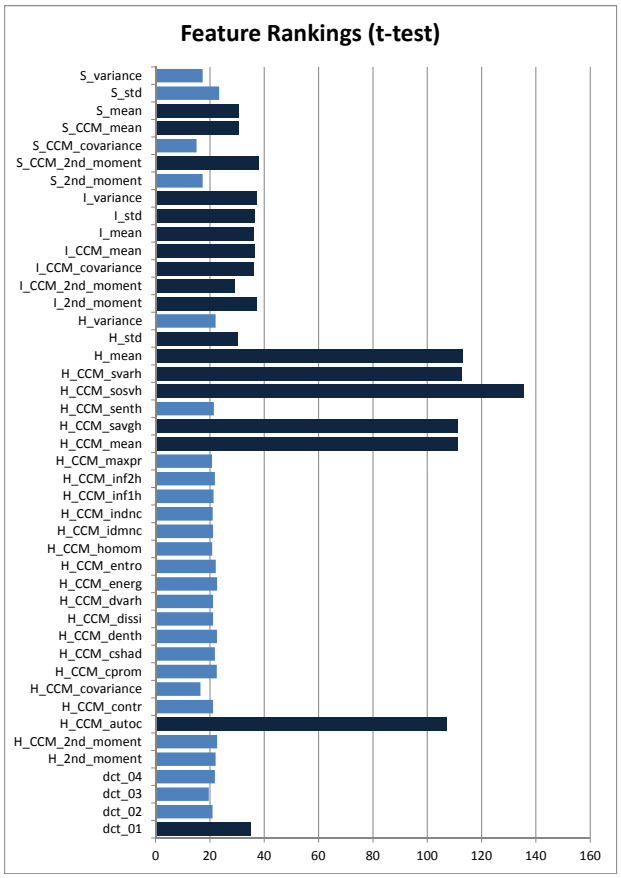

Figure 4.9: Feature rankings.

Database

Food regions are selected from a reference database automatically by plate detection and segmentation discussed in sections 4.3.1 and 4.3.2, and verified manually. We used

Table 4.1: Final feature list.

	Feature	Description	Quality
1	H_CCM_sosvh	Hue CCM sum of squares: variance	153.317
2	H_mean	Hue mean	126.2723
3	H_CCM_mean	Hue CCM mean	123.5519
4	H_CCM_savgh	Hue CCM sum average	123.5105
5	H_CCM_svarh	Hue CCM sum variance	123.4424
6	H_CCM_autoc	Hue CCM autocorrelation	113.6674
7	I_CCM_mean	Intensity CCM mean	37.5593
8	I_mean	Intensity mean	37.3651
9	I_variance	Intensity variance	37.1314
10	I_2nd_moment	Intensity 2nd moment	37.1314
11	S_CCM_2nd_moment	Saturation CCM 2nd moment	37.0023
12	I_CCM_covariance	Intensity CCM covariance	36.2997
13	I_std	Intensity local standard deviation	35.001
14	H_std	Hue local standard deviation	34.1799
15	dct_01	Discrete cosine transform 01	33.5224
16	I_CCM_2nd_moment	Intensity CCM 2nd moment	32.8855
17	S_CCM_mean	Saturation CCM mean	28.7689
18	S_mean	Saturation mean	28.6047

existing databases generated for the FIRSSt and ASA24 studies [11]. These databases have one food type per plate and known distance, angle, and lighting conditions, among other things, so in this case automatic segmentation is greatly simplified. After the food region is detected, the user can verify and classify the region. Combining this region with orientation information gives visible surface area, and combining that with the number of grams can be used to establish a relation between surface area and mass. The training databases we used contain spreadsheets including mass measurements, so the class and mass lookups could be automated.

Optional Re-Segmentation

In the case of automatic segmentation the initial regions can be cleaned up with the aid of classification results. Adjacent or overlapping segments can be merged based on size and similarity. Small, scattered, or otherwise invalid segments can be removed. We did this successfully for around 30 classes, but as discussed in section 4.3.2 our segmentation by classification method did not scale to large numbers of classes.

It is worth noting that a region merge would force re-classification for the new bigger regions. Fortunately in the case of our algorithm that simply means summing the final vote tallies for each of the sub-regions, which are readily available with no additional processing.

4.3.4 Volume Estimation

We estimate food volume by estimating surface area, then finding a correlation between surface area and volume for each food type. Our results and others [180] show that surface area is well correlated with volume for many foods. Its accuracy is slightly lower than that of human analysts [45], but not much, and it requires a minimum of equipment and user interaction.

Several assumptions simplify the surface area estimation. Firstly, both the plate and the reference card lie flat on the same plane, i.e., the table. Secondly, most foods lie fairly flat on a plate. Third, the reference card is around the same distance from the camera as the plate. Finally, most food photos are taken from above the image at an angle of 45 degrees or more, and depth does not vary much between the front and back of the plate.

Originally, we tried to analyze the images by the direct approach using camera parameters (sensor geometry, focal length) to convert 2d coordinates to 3d and thereby calculate

food size. Unfortunately focal length varies by camera, and is not always easy to look up. Our experience showed that reported focal length, either from spec sheets or metadata may vary significantly from actual focal length, and that it changes slightly as camera is focused. The numbers are probably intended to generate rules of thumb, and are not accurate enough for fine calculations. Fortunately there is a way to bypass camera parameters via combining multiple geometric transformations and making some assumptions.

Geometric transformations can be used to convert from one set of coordinates to another; allowing us to calculate the original spatial coordinates from image coordinates, or convert between different viewpoints. For a camera directly above a flat plane (table), pixel coordinates $\langle x_{img\_top}, y_{img\_top} \rangle$ as a function of spatial coordinates $\langle x_{top}, y_{top}, z_{top} \rangle$ relative to the camera are:

$$x_{img\_top} = c \cdot \frac{x_{top}}{z_{top}} \tag{4.23}$$

$$y_{img\_top} = c \cdot \frac{y_{top}}{z_{top}} \tag{4.24}$$

Here, c is a scaling constant dependent on camera focal length and image scale. On flat tabletop from above, z_{top} is constant as well.

If the camera is not directly above the table, but rather held by someone standing in front of the table at some angle θ, the 3d coordinates of objects on the table relative to the camera's true position can be found by applying the 3x3 rotation matrix:

$$R_x(\theta) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\theta & -\sin\theta \\ 0 & sin\theta & \cos\theta \end{bmatrix} \tag{4.25}$$

to give:

$$x_{top} = x_{cam} \tag{4.26}$$

$$y_{top} = y_{cam} \cos\theta - z_{cam} \sin\theta \tag{4.27}$$

$$z_{top} = z_{cam} \cos\theta - y_{cam} \sin\theta \tag{4.28}$$

The image coordinates associated with the camera's true position are:

$$\begin{gathered} x_{img} = c \cdot \frac{x_{cam}}{z_{cam}} \\ \frac{1}{c} x_{img} z_{cam} = x_{cam} \end{gathered} \tag{4.29}$$

$$\begin{gathered} y_{img} = c \cdot \frac{y_{cam}}{z_{cam}} \\ \frac{1}{c} y_{img} z_{cam} = y_{cam} \end{gathered} \tag{4.30}$$

We can combine these equations to derive the perspective transform:

$$x_{img\_top} = c \cdot \frac{x_{cam}}{z_{cam}\cos\theta - y_{cam}\sin\theta} = c \cdot \frac{\frac{1}{c}x_{img}z_{cam}}{z_{cam}\cos\theta - \frac{1}{c}y_{img}z_{cam}\sin\theta} = \frac{x_{img}}{\cos\theta - \frac{1}{c}y_{img}\sin\theta} \tag{4.31}$$

$$y_{img\_top} = c \cdot \frac{y_{cam}\cos\theta - z_{cam}\sin\theta}{z_{cam}\cos\theta - y_{cam}\sin\theta} = c \cdot \frac{\frac{1}{c}y_{img}z_{cam}\cos\theta - z_{cam}\sin\theta}{z_{cam}\cos\theta - \frac{1}{c}y_{img}z_{cam}\sin\theta} = \frac{y_{img}\cos\theta - c\cdot\sin\theta}{\cos\theta - \frac{1}{c}y_{img}\sin\theta} \tag{4.32}$$

This shows that if we can measure the scale and angle of rotation, we can calculate the pixel coordinates on the top view image without any information on camera parameters or depth. Furthermore, on a flat table where z_{top} is constant, the x, y distances on the image are directly proportional to the actual x, y, z distances in space. So with a reference card or known plate we can estimate scale and rotation objects flat on the table accurately. The results of a perspective transform are shown in Figure 4.10.

Figure 4.10: left: raw image, middle: perspective transform, right: standard with known weight

We found that in practice there were several problems with this. Firstly, z_{top} varied meaningfully between the tabletop or plate and the top of some foods. Second, minor errors in corner locations are compounded farther away from the reference card. This can be seen in Figure 4.10. As a practical matter we used a simplification - the affine transformation, in 2d:

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \tag{4.33}$$

which accounts for the differences in scale in the x and y directions due to rotation, but ignores differences in depth. As shown in Figure 4.11, this is more robust against small differences in depth and errors in card detection. Equation 4.33 shows that the 2d affine transformation has 6 degrees of freedom, so it requires a minimum of 3 $\langle x, y \rangle$ control points to specify. Since we have 4 card corners, this is an overdetermined system, and in an affine transformation the edges of the card will not be parallel afterwards. So we set the parameters to minimize the error between all 4 card corners and the card corners of the ideal "correct" rectangular card with the correct height, width and orientation. This method ignores scale variations due to different depth but in practice gives good, stable

results for our problem; even in the best cases for perspective transform, affine transform gave indistinguishably accurate results.

Figure 4.11: left: raw image, middle: affine transform, right: standard with known weight

Next, mass is estimated from surface area using the known masses of training samples. This makes the assumption that mass is linearly related to volume i.e., the average density of a food type is always the same. Presently the mass per surface area k_{food} is calculated for each training sample and these values are averaged to estimate the relation (Eq. 4.34). Testing showed that most foods, when scooped or placed onto a flat plate, tend to have a strong linear relationship between mass and surface area, as shown in Figure 4.12. As shown in Table 4.2, 22 food types were tested, with a mean error of around 15%, varying significantly by food type. Figure 4.13 and Figure 4.14 show linearity graphs for the best and worst cases tested.

$$mass = k_{food} \times surface\ area \tag{4.34}$$

For liquids in opaque cups and soup in bowls there is also a linear relationship between volume and surface area, or at worst a very simple curve. In this case though, the relationship depends on the properties of the container rather than those of the food, so in this case k_{food} becomes $k_{container}$. Figure 4.15 shows this phenomenon, it's easy to pick out the curves associated with 2 different bowls across several food types. So for these food types, volume estimation would only be viable with standardized containers, for example in a cafeteria setting.

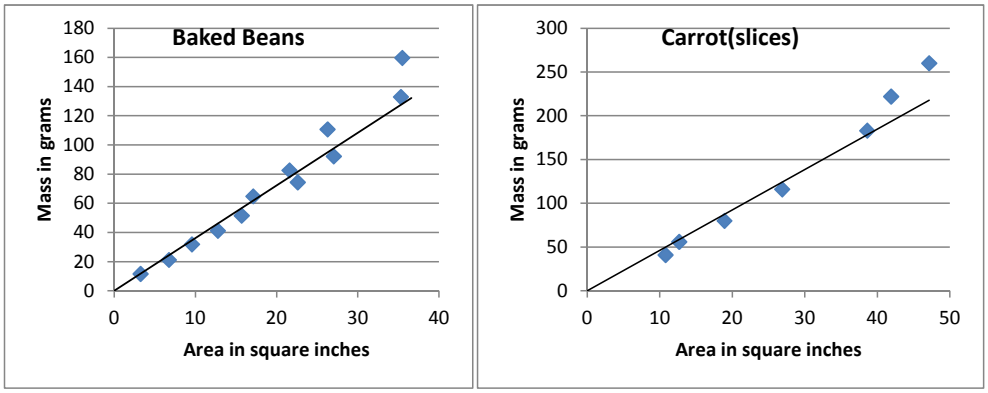

Figure 4.12: Volume estimation on solid foods.

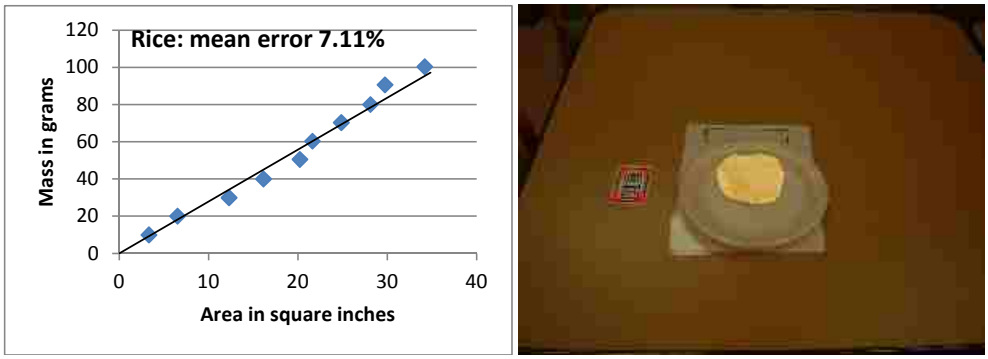

Figure 4.13: Best linearity.

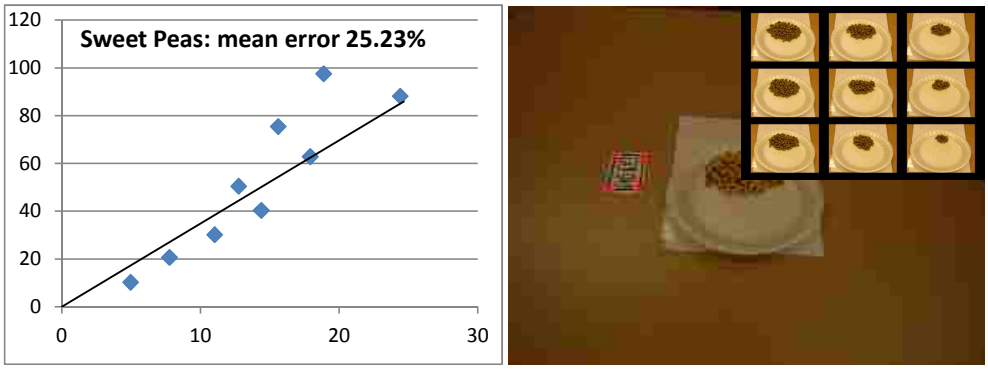

Figure 4.14: Worst linearity.

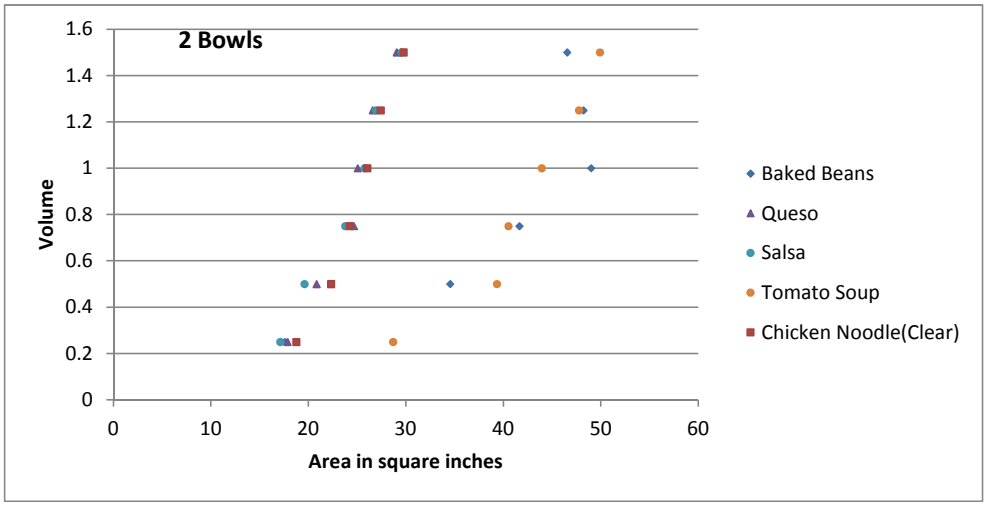

Figure 4.15: Volume estimation on liquid foods. The two different curves for bowl A and B are clearly visible

Table 4.2: Volume estimation results.

Food Type	# Samples	Mean Error
Baked Beans	12	9.19%
Baked Drumstick	3	5.51%
Broccoli with Cheese	10	18.59%
Chicken Breast Pieces	10	20.37%
Chili Mac	12	15.71%
Corn on the Cob	2	0.84%
Enchilada Casserole - messy	10	15.42%
Enchilada Casserole - whole	3	7.45%
Garlic Toast	3	15.11%
Lasagna	12	22.37%
Macaroni and Cheese	10	11.47%
Pinto Beans	10	8.09%
Potato Salad with Mayo	11	16.55%
Rice	10	7.11%
Spanish Rice	10	13.72%
Spinach	10	22.02%
Sweet Peas	9	25.23%
Carrots	7	10.58%
Hot dog	8	8.21%
Pineapple chunks	8	16.79%
Potato Salad	8	10.35%
Sugar Snap Peas	8	13.78%

4.4 Results

Our training and testing sets were taken partially from the FIRST and ASA24 [11] datasets, and partially in house at the Baylor College of Medicine Children's Nutrition Research Center (CNRC). With such a large number of classes, many of which were similar, nearly indistinguishable or even legitimately overlapping, the best way to report and compare the results seemed to be percentile scores, what percentile of results the correct food was in (Table 4.3, Figures 4.16 and 4.17).

Table 4.3: Classification results.

Ranking	Percentile	Percentage of Foods
First Place	100	76%(84/110)
Top 3	98+	92%(101/110)
Top 5	96+	95%(105/110)

Overall, our classification results (92% correct within 98th percentile, 110 foods), were good compared to those of comparable to those of previously/concurrently published food recognition systems such as [83] (80.05% correct within the 96th percentile, 50 foods), and their follow up work in [95] (81.55% within 92nd percentile, 50 foods). Many other systems were not comparable because they didn't work on, and were not designed for, large numbers of foods; we aren't aware of any systems that work on more than 30 food types.

The next year, [83] further extended their work [95] to include 100 food types, and achieve a recognition rate of 92% within 96th percentile. Their revised system made use of

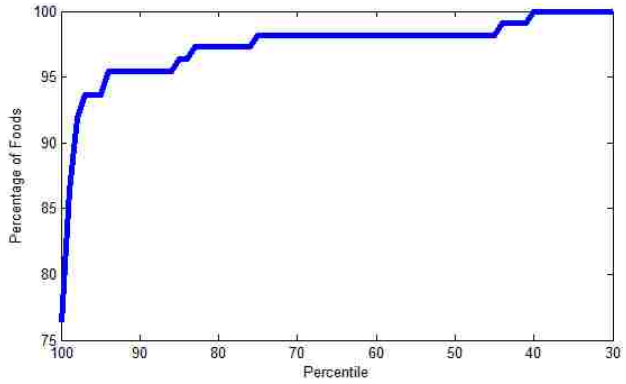

Figure 4.16: Classification results.

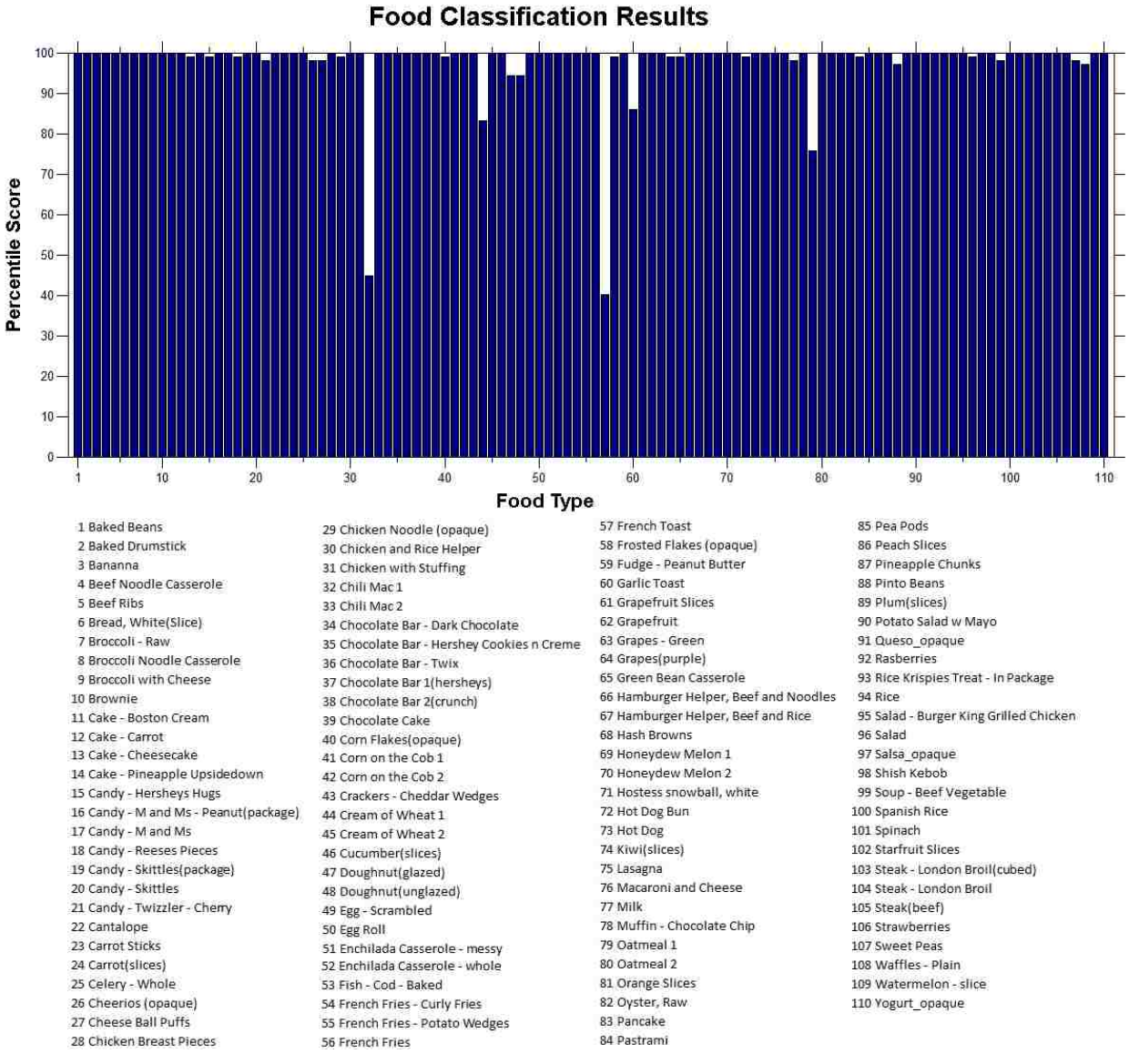

Figure 4.17: Detailed testing results.

the newly popular Convolutional Neural Network (CNN), which would soon come to dominate high powered (as opposed to high speed) image recognition, and has some application in texture analysis as well. There were other CNN methods at the time [87], but they still didn't provide a list of runners up, and were not applicable to large numbers of classes.

4.5 Discussion

Practical implementation of the system gave us a number of insights into the unique problems of food recognition. Upon examining our main sources of error, we found a number of food types in our database raised interesting and unforeseen classification issues. We noticed other issues that would be difficult for other classifications methods were inherently easy for our OVTR, and it may have other useful features we can take advantage of.

4.5.1 Main sources of error

The Primary sources of error for OVTR in this dataset seem to be residual lighting differences between the 2 subsets (Figure 4.18), and cases where the appearance of the training and testing set are different due to several possible 'types' among the same food. The first case can be mitigated by implementing better lighting normalization; the second requires training samples from each type.

Figure 4.18: Lighting differences.

4.5.2 Interesting Cases

A common special case is that of multimodal food, where a food has 2 or more colors/textures throughout, or has distinct regions of different colors/textures. Our classifier performed well on these types of foods. Some examples of multimodal foods are show in Figure 4.19.

In a large food database, there will be many types of very similar looking foods, for example even in our relatively small training sample there were 3 types of steak, 4 types of

Figure 4.19: Multimode foods.

chocolate bars, chocolate cake and chocolate brownie. All of these are basically the same color, and within each group even the texture was often essentially the same.

Furthermore, in practice an image can legitimately match with multiple classes. If there's a database entry for 'chocolate' and a separate entry for 'Hershey chocolate bar', one is a subclass of the other and equally legitimate for a match. It can be concluded then, that for real world data, a ranked list of results is needed, rather than a single classification.

Another interesting case is foods where two examples of the same food can look very different. For example lumpy vs smooth oatmeal, lasagna with different cheese-to-sauce ratio or physical layout than the training sample, or new vs old bananas. For our system, this is similar the multimodal case mentioned above; however this type is impervious to global features, and the training set must include examples of each type for ideal results. A few examples of this are shown in Figure 4.20. In our results for these cases, both 'types' were not included in the training set.

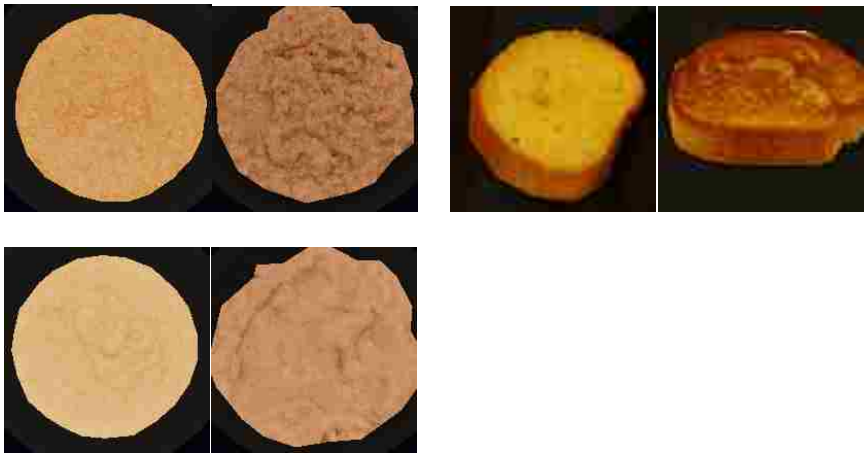

Figure 4.20: Multiple types/textures (top left: oatmeal, top right: garlic toast, bottom left: cream of wheat.)

4.5.3 "One versus the rest" advantages and features

Using individual specialist networks for each food makes this system parallelizable. Histogram comparison also is more workable in this case because it generates only one training feature for each network. Having a one element target vector reduces stress on computer memory and processing time. The specialist networks stress positive classification, preferring false positives over false negatives; so each network should have the maximum chance

to detect its assigned food type. The voting scheme has several novel advantages. Firstly it handles multimode foods well, since if a food is composed of 2 adjacent regions, the votes for both types of regions will be positive. Even a small "crust" that takes up 5-10% of the image will effectively break ties without needing to do a very detailed global analysis of the types and amount of different modes. Foods with multiple "types" such as green, yellow, and brown bananas are seamlessly handled as well. Secondly, with hundreds or thousands of food types, correct classification is not viable in every place, so any classification method needs to return a sorted list of results; voting achieves this far more effectively than directly using a neural net to classify.

For any classifier trying to discriminate between several similar food types, naturally any distance measures in the contested region are liable to be inflated. This runs counter to our goal of creating a sorted list based on absolute quality of the match against each training food. So from this standpoint multiple binary classifications are preferable as well.

Chapter 5
An Agile Framework For Real-Time Visual Tracking in Videos

5.1 Introduction

Automated tracking of moving objects in a video in real time is important for different applications such as video surveillance, activity recognition, etc. Existing visual tracking algorithms [61, 89, 100, 103] cannot automatically adapt to changes in lighting conditions, background, types of sensors (e.g., electro-optic vs. infrared) and their dynamics (zooming, panning, etc.) easily. They cannot gracefully handle data that simultaneously contains different types of motions such as both slow and fast moving objects, motion behind an occlusion, etc. Many of the existing tracking algorithms [61, 89, 100, 103] cannot start the tracking process automatically; they require a user to draw a box on an object that needs to be tracked for the process to be initiated.

We present an agile framework for automated tracking of moving objects in full motion video (FMV). The framework is robust, being able to track multiple foreground objects of different types (e.g., person, vehicle) having disparate motion characteristics (like speed, uniformity) simultaneously in real time under changing lighting conditions, background, and disparate dynamics of the camera. It is able to start tracks automatically based on a spatio-temporal filtering algorithm and is able to gracefully handle objects in occluded surroundings. Unlike many existing tracking algorithms [89], with high likelihood, it does not lose or switch tracks while following multiple similar closely-spaced objects. The framework is based on an ensemble of tracking algorithms that are switched automatically for optimal performance based on a performance measure without losing state. Only one of the algorithms that has the best performance in a particular state is active at any time, providing computational advantages over existing ensemble frameworks like boosting. We prove theoretically (lemmas 1 and 2) that the presented agile tracking framework is more accurate than existing individual/ensemble-based algorithms. A C++ implementation of the framework (for the purposes of this chapter, we only consider two algorithms in our ensemble: Gaussian Mixture Background Subtraction (GM) and optical flow) has outperformed existing visual tracking algorithms on most videos in the Video Image Retrieval and Analysis Tool [178] (/acrshortvirat: www.viratdata.org) and the Tracking Learning Detection (TLD) [89] data-sets.

5.2 Contributions

The tracker makes several contributions. Firstly, our agile tracker provides a domain specific solution applicable to car and human tracking both in poor quality aerial videos and good quality surveillance videos. Although intended to be part of a larger system, it is designed to be a complete tracking system suitable for practical stand alone application in the specified domain. In contrast to similar surveillance trackers, our tracker uses multiple main tracking engine types and can switch among them when performance drops. Secondly, the agile framework provides a novel method of merging multiple trackers

to produce better results than any individual in the ensemble, by allowing each tracker to run in the situations it is most applicable to. This relies on an estimate of tracker applicability that the agile framework generates from various video and object metrics. Our agile tracker was published at a workshop in the 39th Annual International Computers, Software & Applications Conference (COMPSAC) [20].

5.3 Related Work

A spatio-temporal tracking algorithm was proposed in [103] that involved tracking articulated objects in image sequences through self-occlusions and changes in viewpoint. However, they did not provide capabilities of automatic track starting or tracking multiple objects. The work in [100] combines background subtraction, feature tracking, and grouping algorithms. However, their work didn't have any suitable classification method based on the spatial features of the objects detected. A Kernel Particle Filter (KPF) was introduced in [37] for tracking for objects in image sequences. The idea proposed in [187] shows tracking using a single classification SVM. A boosting-based approach was proposed in [177] that used a cascade of classifiers for object detection. However, it didn't address the problem of tracking objects through consecutive frames of a video sequence.

Among the existing tracking frameworks the one most relevant to our work is the TLD algorithm proposed in [89]. This algorithm is far more general and robust compared to most other trackers and deals with drift well. The main problem inherent in this algorithm is its inability to start tracks automatically as well as lacking a multi-object tracking feature. Also, TLD is based on template matching and hence fails for videos with multiple numbers of similar looking objects.

5.4 The Proposed Approach

Figure 5.1 shows the schematic of our approach. First, a moving object must be automatically identified as part of the foreground. This involves starting tracks at particular pixels on the subsequent frames that have a higher probability of being part of the moving foreground object. This is achieved by 1) stabilizing the image and 2) feeding the stabilized image to the spatial and temporal filtering algorithms described below. Once the track starter algorithm has precisely marked the object coordinates, the objects must be tracked if any motion is to be identified. Issues such as camera instability (shaking, panning, rotating) come into play and require image stabilization for the tracking to be successful.

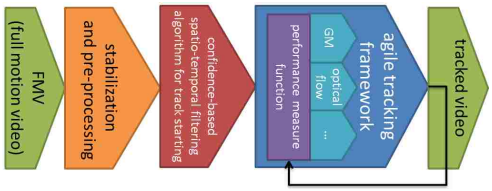

Figure 5.1: Schematic representation of our approach.

5.4.1 Image Stabilization

An incoming video is first stabilized using an iterative algorithm:

1. Apply Shi and Tomasi's edge-finding algorithm to the first frame to identify significant feature points in the image.

2. For each subsequent frame, apply Lucas-Kanade optical flow to track the motion of the features identified by Shi and Tomasi's algorithm, refreshing the feature points when necessary.

3. With increasing precision for each iteration:

   (a) For each angle of rotation in a certain range, determine the translation of each point.

   (b) Find the most common (mode) translation/rotation pair (Θ, x) and (Θ, y) of all the features.

4. Warp the image to adjust for the total mode of the motion.

At present, our method stabilizes the image for small amounts of translational and rotational camera movement. Thus, for wide camera sweeps or changes in perspective or scale, our stabilization method is not, at present, appropriate.

5.4.2 Track Starting

The automated track starting algorithm based on a confidence-based spatio-temporal filtering algorithm first detects blobs using the GM Background Subtraction method [86]. This yields difference images, which are fed into the spatial filtering module below.

The data structure prob_obj represents the blob picked out by the initial spatio-temporal filtering as a possible object of interest. It is a group of pixels, and also has a height, width, and confidence measure δ associated with it.

1) Opening or Closing images of Images via Image Morphing

The image obtained through the background subtraction algorithm is initially opened by a structuring element with diameter 3 pixels to filter out unnecessary noise. By opening, we mean the dilation of the erosion of a set A by a structuring element B. Then it is closed with k-means clustering [52]. This helps in detecting blobs over subsequent frames.

2) Spatial Filtering

Once blobs are detected in the difference images, they are filtered according to their spatial features. The pseudo code for the spatial filtering algorithm is provided below. Scale information available from the metadata accompanying the videos is used to filter blobs specifically based on their area and orientation. The filtered blobs are then passed as input to the temporal filtering algorithm below.

3) Temporal Filtering

To filter blobs in the temporal domain we use a *confidence measure*. Each blob has a confidence measure δ associated with it.

Initially the confidence value for each blob is zero. Confidence value for a blob increases as it is detected across successive frames In case a blob appears in consecutive frames, the confidence value increases according to a prior confidence measure. The confidence update equation is as follows:

Equation for confidence gain:

$$\delta = 0.5^{-n} \tag{5.1}$$

Equation for confidence loss:

$$\delta = -0.5^{-n} \tag{5.2}$$

where n is the frame number.
The composite confidence update equation is as follows:

$$\delta = 0.5^{-n} \vee -0.5^{-n} \tag{5.3}$$

So, the confidence update equation takes the form portrayed in Figure 5.2.

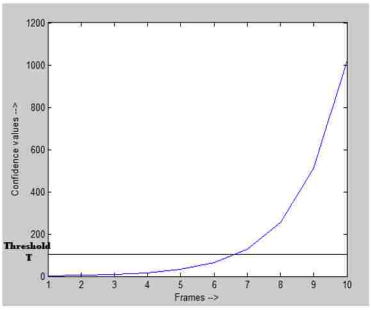

Figure 5.2: Confidence value update for the frames (for increasing confidence)

4) Adaptive Thresholding

If the confidence value for a blob exceeds a specified upper threshold σ, a track is started on it. The moment the confidence value for a blob falls beneath a lower threshold τ, the corresponding object is discarded. If the confidence value is between σ and τ, the corresponding blob is maintained in the list of prospective tracks. If the confidence measure increases to a value higher than the upper threshold σ, then a track is started at the pixel representing the object coordinates. For videos that have higher noise, clutter and random changes in lighting conditions, as is often the case for outdoor videos taken from moving cameras, the upper threshold value σ is set higher. On the other hand, for videos with more stable conditions σ is set lower because of the lesser probability of encountering random classification noise.

5.4.3 The Agile Tracking Framework

Object tracking is a matter of determining the apparent motion of the target object, keeping track of its pixel coordinates. Many object tracking methods are based on optical flow, or use it as a subpart. Existing methods like Kalman Filter [61] based on a Bayesian model and TLD [103] based on Template Matching primarily use a single learner to perform the underlying computations. In the field of machine learning, ensemble methods consist of a collection of models that can be used to improve the classification accuracy as compared to any of the individual models [138, 143, 147]. The No Free Lunch (NFL) theorem of Wolpert [191] states that all learners have the same average performance across all problem classes. It follows that a learning algorithm can produce better than average results on some problem classes, always at the cost of producing worse than average results on all other classes. This result was one of the motivations behind the advent of boosting algorithms, which combine multiple weak learners into a stronger one. In a similar manner, if we can

Algorithm 5 The Track Starting Algorithm

```
procedure TRACKSTARTING
    img ← getFrame(video);
    img ← STABILIZE_IMAGE(img);
    bw_img ← GM_BACKGROUND_SUBTRACTION(img);
    sl ← create_structuring_element(3);    ▷ 3 is the diameter of the structuring element
    img ← PERFORM_OPEN_ON_IMAGE(bw_img,sl);    ▷ morphological opening on image
    sl ← create_structuring_element(n);    ▷ n is chosen adaptively acc. to the image
    img ← PERFORM_CLOSE_ON_IMAGE(img,sl);  ▷ morphological closing on image
    contour_img ← FIND_CONTOUR(img);    ▷ finds the boundaries on the image
    count = 0;
    while (contour != NULL) do
        prob_obj ← GET_OBJ_FROM_CONTOUR(contour_img);▷ probable object blob
        count ← count + 1;
    end while
    for i = 0 to count do
        temp ← SPATIAL_FILTERING(prob_obj);
    end for
    while (temp != NULL) do
        obj ← TEMPORAL_FILTERING(temp);
    end while
end procedure
```

Algorithm 6 The Track Starting Algorithm - Spatial Filtering

```
function SPATIAL_FILTERING(prob_obj)
    if (prob_obj.size < τ1 AND prob_obj.size > τ2
            AND prob_obj.height/prob_obj.width < τ3 AND prob_obj.height/prob_obj.width > τ4) then
                                    ▷ τ1 ,τ2 ,τ3 and τ4 indicate the respective thresholds
        return prob_obj;
    else
        return NULL;
    end if
end function
```

Algorithm 7 The Track Starting Algorithm - Temporal Filtering

```
function TEMPORAL_FILTERING(temp)
    for each prob_obj do
        δ_prob_obj ← 0;                          ▷ init. weight of each object detected as 0.
    end for
    if (for video.nextframe) obj_detected = prob_obj then
        δ_prob_obj ← δ_prob_obj + (0.5)^-n;                ▷ confidence update equations
    else
        δ_prob_obj ← δ_prob_obj − (0.5)^-n;
    end if
    if δ_prob_obj ≤ τ then
        remove prob_obj from list of objects;
    else      ▷ append to the list of objects detected. Φ represents the append operator
        obj ← obj Φ prob_obj;
    end if
    for each obj do
        if δ_prob_obj ≥ σ then
            start tracks on obj_(x,y);          ▷ start tracks on object centroids (x,y)
        end if
    end for
    return obj;
end function
```

estimate the error rate of a given tracker in a given situation, an ensemble of trackers can perform better than any individual.

It can be shown through the following lemma that an ensemble learner performs better than any of the constituent learners.

Lemma 5.4.1 *Even a strong learner cannot endure situational variances, i.e., it cannot perform well at all situations.*

Proof The Boosting algorithm [156] described by Schapire and subsequently proposed implementations like Adaboost use Convex Potential Boosters. As shown in [113], for a wide range of convex potential functions, any boosting algorithm is bound to encounter random classification noise. They show that any such boosting algorithm is able to classify examples correctly in absence of noise but in the presence of noise the learner cannot learn to an accuracy better than 1/2. This holds even if the boosting algorithm stops early or the voting weights are bounded.

Consider two sets of disjoint concept classes C_1 and C_2 such that $C_1 \Lambda C_2 = \emptyset$. Now, if we consider an instance space X containing elements from C_1, then any $c \in C_2$ can be classified as random noise in X. So, effectively at least two different learners L1 and L2 are needed for classifying the instances in X according to C_1 and C_2. ∎

In the light of this pre-defined notion, we present a new agile learning based tracker, that uses a combination of two methods for computing position: Gaussian Mixture (GM)

background subtraction [86] for quick-moving and the Lucas-Kanade method for slow-moving [114] objects in order to account both for fast and slow velocities. By the agile learning based tracker, we imply that our tracker can adaptively switch between the constituent learners at runtime based on the video and object properties.

We present a new agile tracking framework that uses an ensemble of k individual trackers. The framework allows adaptive switching between the constituent trackers dynamically based on a performance measure. The algorithm for adaptive switching is described below.

Algorithm 8 The Switching Algorithm

```
procedure SWITCH
    j ← 1;
    active_tracker ← T_j        ▷ T_j is the j^th tracker compute the performance measure λ
    if λ ≥ threshold Φ then
        CHECKPOINT_CURRENT_STATE();                    ▷ saves the current state
        active_tracker ← CALL_TRACKER_SELECTOR();      ▷ calls a new tracker
        state ← GET_CHECKPOINTED_STATE();              ▷ returns the checkpointed state
        state ← active_tracker(state);
    else
        continue;
    end if
    if performance measure λ is minimized then
        i ← i+1
    end if
end procedure
```

The switching module is called by the agile tracking algorithm:

Algorithm 9 The Tracking Algorithm

```
procedure AGILE_TRACKER(freq)
    for each frame i do
        if frame_number % freq = 0 then
            call SWITCH()
        end if
    end for
end procedure
```

In the above algorithm, state refers to the set of tuples (x,y,n,I), where x and y are the pixel coordinates, n is the frame number and I is the intensity. The agile tracker calls the switching algorithm at a user specified frequency. The switching algorithm computes the performance measure at the current state. If it exceeds a threshold the current tracker is substituted with a new one obtained from an ensemble through a pre-defined policy in such a way that the application of the new tracker to the current state results in one whose performance measure value is below the threshold. While switching, the current state is checkpointed so that it can be accessed by the new tracker. We use the linear function given below as the performance measure.

$$P = k_1 \times \text{stabilization\_error} + k_2 \times \text{track\_overlap\_amount} + k_3 \times \text{probability\_jump\_detected} + k_4 \times \text{probability\_drift\_detected} + k_5 \times \text{track\_speed} \tag{5.4}$$

where $k_1, k_2, \ldots, k_5$ are constants whose sum is 1 and whose values depend on the constituent trackers in the ensemble. The performance measure quantifies the tracking error at the current state.

The next lemma shows that dynamic switching between individual trackers yields more accurate results.

Lemma 5.4.2 *Switching between individual trackers dynamically can decrease the upper bound for error up to a certain pre-defined value.*

Proof Suppose c(v) is the correct classification for v and $h_1(v)$, $h_2(v)$ etc. are the classifications produced by the trackers T_1, T_2, etc respectively. h(v) is the estimate produced by the effective composite tracker T.

Here, $T = T_1 \Delta T_2 \Delta \ldots \Delta T_n$, where, T_1, T_2, etc indicates the trackers and Δ indicates the switch operator on the trackers.

Also, let a_1, a_2, etc be the respective probabilities of error or misclassification Also, for switching between trackers dynamically at runtime we incorporate the idea of defining adaptive thresholds τ_1, τ_2 etc. So, we define the set $\tau = \{\tau_1, \tau_2, \tau_3, \tau_4, \tau_5, \ldots, \tau_n\}$ as the threshold for the number of misclassifications. If the number of misclassifications for a particular tracker T_i exceeds the corresponding threshold τ_i we switch the learner.

Suppose for the i^{th} tracker, the number of misclassifications become $(\tau_i + 1)$ at the $(n_i + 1)^{th}$ instance. So, up to the n_i^{th} instance, probability of error or misclassification is

$$Pr(h(v) \neq c(v)) = \left(\frac{\binom{n_i}{\tau_i}}{n_i!}\right) \times a_i^{\tau_i} \tag{5.5}$$

Also, let $\dot{\alpha}$ be the upper bound of error on any of the individual trackers. Hence, for the total tracking process, the composite probability of misclassification is given by

$$\begin{aligned} Pr(h(v) \neq c(v)) &= Pr((h_1(v) \neq c(v)) \Lambda (h_2(v) \neq c(v)) \Lambda \ldots \Lambda (h_n(v) \neq c(v))) \\ &= \left(\frac{\binom{n_1}{\tau_1}}{n_1!}\right) \times a_1^{\tau_1} \times \left(\frac{\binom{n_2}{\tau_2}}{n_2!}\right) \times a_2^{\tau_2} \times \left(\frac{\binom{n_3}{\tau_3}}{n_3!}\right) \times a_3^{\tau_3} \times \cdots \times \left(\frac{\binom{n_N}{\tau_N}}{n_N!}\right) \times a_N^{\tau_N} \\ &\leq \left(\frac{\binom{n_1}{\tau_1}}{n_1!}\right) \times \dot{\alpha}^{\tau_1} \times \left(\frac{\binom{n_2}{\tau_2}}{n_2!}\right) \times \dot{\alpha}^{\tau_2} \times \left(\frac{\binom{n_3}{\tau_3}}{n_3!}\right) \times \dot{\alpha}^{\tau_3} \times \cdots \times \left(\frac{\binom{n_N}{\tau_N}}{n_N!}\right) \times \dot{\alpha}^{\tau_N} \\ &\quad \text{[Since, for all i, } a_i \leq \dot{\alpha}\text{]} \\ &= \left(\frac{\binom{n_1}{\tau_1}}{n_1!}\right)\left(\frac{\binom{n_2}{\tau_2}}{n_2!}\right)\left(\frac{\binom{n_3}{\tau_3}}{n_3!}\right) \cdots \left(\frac{\binom{n_N}{\tau_N}}{n_N!}\right) \dot{\alpha}^{(\tau_1+\tau_2+\tau_3+\cdots+\tau_N)} \leq \dot{\alpha} \end{aligned} \tag{5.6}$$

Here, N is the number of switches performed at runtime.

Observations:

1. Inequality (5.6) holds because each of the terms $\left(\frac{\binom{n_i}{\tau_i}}{n_i!}\right) \leq 1$ as well as $\dot{\alpha}^{(\tau_1+\tau_2+\tau_3+\cdots+\tau_N)} \leq \dot{\alpha}$, since, $\dot{\alpha} \leq 1$.

2. So, the overall upper bound for the error of the composite tracker is reduced owing to switching at runtime.

3. Inequality (5.6) proves that the effective composite error bound of the *agile tracker* T is less than any of the individual trackers T_i.

1, 2 and 3 justify our argument that using switching reduces the overall error bound.

Threshold value selection is a very important criterion in optimizing the agile tracker. In order to evaluate the threshold selection criteria, let us concentrate on the simplified version of the equation presented in (5.6).

So, we have classification error

$$\Pr(h(v) \neq c(v)) \leq \prod_{i=1}^{N} \left(\frac{\binom{n_i}{\tau_i}}{n_i!}\right) \dot{\alpha}^{\tau_i} = \left(\frac{1}{\tau_1!(n_i-\tau_i)!}\right) \dot{\alpha}^{\tau_i} \tag{5.7}$$

The error bound can be minimized by increasing τ_i until $\tau_i = \lceil n_i/2 \rceil$. ∎

In a typical video scenario, most features are stationary from frame to frame with only a few objects moving. The stationary features are considered to be in the background, and the moving objects are foreground. The GM background subtraction method described in [86] efficiently segments foreground and background objects in real time, allowing for effective object tracking. However, as is typical of background segmentation methods, it becomes less effective when there is camera instability. Even with a stable camera, this method tends to lose foreground objects if there is relatively small movement in the foreground. To compensate for these deficiencies, we also use a more traditional and robust optical flow method for object tracking.

The Lucas-Kanade method, like many algorithms used to compute optical flow, imposes a constraint on the optical flow problem: the displacement $(\delta x, \delta y)$ of the image intensity from a pixel (x, y) to a pixel $(x + \delta x, y + \delta y)$ in the subsequent frame is small and constant over time. That is, it must satisfy for all pixels p the equation:

$$I_x(p)\, V_x \;+\; I_y(p)\, V_x \;=\; -I_t(p) \tag{5.8}$$

where I_x, I_y and I_t are the partial derivatives of the image intensity with respect to x, y and t, and V_x and V_y are the velocity vectors. This usually results in an over-determined system and uses least-squares to find a solution. Due to the constraint imposed by the method, it is best suited for a object moving slowly with constant velocity. We use pyramidal Lucas-Kanade. That is, we compute Lucas-Kanade at the lowest-resolution image I_0; then, having obtained this lower-resolution result, we compute Lucas-Kanade incrementally for the next lowest resolution I_1. Similarly, we obtain I_2 from I_1, and so forth until reaching the full resolution.

Combined, the Lucas-Kanade method and GM background tracking ensure motion-tracking performance superior that of either method used alone.

When used on Unmanned Aerial Vehicle (UAV) videos, object tracking presents an array of challenges. One is camera instability; often, during recording, the camera shakes, pans, or rotates, which causes background objects to appear to move. A second is poor image quality due to low-definition recording equipment or long distance; this obscures images and interferes with the tracking process. A third is the need for real-time tracking, which requires simple, efficient methods to keep up with the pace of real-time input.

1) Agile Tracking vs. Other Ensemble Based Trackers

A tracker based on an ensemble machine learning technique like boosting would create, based on training data, a tracker of the form:

$$\mathrm{T} = \sum_{p=1}^{P} \alpha_{\mathrm{p}} \mathrm{t}_{\mathrm{p}} \tag{5.9}$$

where P is the number of rounds, t_p is a tracker in the ensemble, and α_{p} are weights such that $\sum_{p=1}^{P} \alpha_{\mathrm{p}} = 1$. T is the tracking vector describing the target's motion.

While running on actual data T will need to run all the P trackers on each data point (i.e., frame) and compute a weighted sum of the outputs. In our case only one tracker is active at any particular time, i.e., only one tracker is run on each data point. This is crucial for real time performance.

Moreover, in boosting, the weights α_{p} are fixed once the training is over. This can create problems if the character of the data changes drastically from the examples on which the training is performed due to changes in background, lighting conditions, etc. This can be avoided in the agile framework by having multiple boosted trackers in the ensemble and switching them accordingly using the SWITCH() method (of course increasing the computational cost) but definitely yielding higher performance.

2) Image Quality and Real-time Tracking

How do we accommodate both poor image quality and the need for real-time tracking? The combination of GM background subtraction and the Lucas-Kanade method ensures a better result than either one alone; Lucas-Kanade tends to succeed where GM background subtraction fails, and vice versa. For a blurry, low-quality, quickly-moving object, GM background subtraction works well as long as the image is stable so that background and foreground objects can be distinguished. If a failure, defined as a large jump or the object is not moving quickly enough to show up in the GM background image, is detected, we track the object's movement according to the output of the Lucas-Kanade flow field.

Fortunately for the sake of efficiency, the incremental cost to stabilize the image is small, since Shi and Tomasi's algorithm need only run once and the Lucas-Kanade flow field is already being computed to track foreground objects.

3) Object Passing One problem with GM background subtraction is when two moving objects are nearby or occluded, it becomes difficult to separate them. Likewise, with Lucas-Kanade, the boundaries of the tracked objects must be approximately known. To account for this, we create a probability image when two objects are nearby, consisting of two Gaussians. The first object cannot move to where the probability is 0 (e.g., at the center

of the second object), and likewise for the second object. This, along with preventing large jumps, usually solves the problem with two objects passing each other in the near vicinity.

5.5 Implementation of our Approach

We implemented tracking in C++ using the OpenCV library [30] for real-time computer vision. The ensemble in our case consisted of two individual algorithms: Gaussian Mixture Background Subtraction and Lucas-Kanade optical flow (LK). The GM algorithm works well at high speeds while the LK performs well at lower speeds. The parameters k_1, k_2, k_3, and k_4 are currently equally weighted, with the exception of k_5 which has been determined by the ability of LK to obtain certain track speeds. Also the switching algorithm was called by the agile tracker every frame.

5.6 Results and Comparative Studies

We compare the results from our tracker against seven existing trackers whose outputs are available at the publicly available TLD dataset [89]. Table 5.1 and Table 5.2 show the number of frames after which the trackers lost track for the first time. The measure proves to be effective in the absence of a track merging algorithm. The agile tracker performs well in most of the cases. Figure 5.3 shows the outputs of the agile tracker on the TLD dataset. Also TLD is based on template matching and hence fails for videos with multiple numbers of similar looking objects. This is illustrated in Figure 5.4 where TLD switches tracks arbitrarily between similar looking foreground objects whereas the agile tracker keeps tracking a particular object for the entire time frame of its visibility. The full length tracked videos along with further results on /acrshortvirat data are available at [3].

Table 5.1: Comparison of the various trackers - number of frames after which the trackers lost track for the first time

	Total frames	**BeyondSemi-Boost**	**coGD**	**CVPR**	**MIL**
Jumping	313	14	1	96	313
Car	945	28	34	29	220
Motocross	2665	6	1	59	63
Car chase	9928	66	1	334	321
Panda	3000	130	1	358	992

Table 5.2: Comparison of the various trackers - number of frames after which the trackers lost track for the first time(contd.)

	Total frames	**Online Boost**	**SemiBoost**	**TLD**	**Agile Tracker**
Jumping	313	26	21	313	313
Car	945	545	652	802	581
Motocross	2665	15	59	173	110
Car chase	9928	316	190	244	402
Panda	3000	1004	83	277	2568

5.7 Conclusions

Our novel approach to track starting using confidence measure and adaptive thresholding not only performs in real time but is also accurate. The *agile tracking framework*

allows dynamic switching within an ensemble of tracking algorithms based on a performance measure while preserving state providing more accuracy than any of the individual algorithms. We believe that the presented framework provides the foundation for real time video activity recognition.

Figure 5.3: Results from the agile tracker.

Figure 5.4: The left one represents the output from the agile tracker and the right one represents that from TLD, which has trouble with nearly identical objects.

Chapter 6
MAPTrack: a Probabilistic Real Time Tracking Framework by Integrating Motion, Appearance and Position Models

6.1 Introduction

Tracking moving objects in a streaming video in real time is important for many applications such as video surveillance, activity recognition, robotics, etc. A statistical method for parametric modeling of object geometry as well as illumination changes owing to variance in lighting conditions was proposed in [69]. However, their approach was only used particularly in tracking human faces; no results are available for videos involving objects having different types of motion such as vehicles, humans, etc. that interact with each other (closely) as is often the case in surveillance videos. In [128] the authors provide a Bayesian framework for combining information obtained about appearance and object geometry for robust visual tracking. However, their framework cannot track multiple moving objects simultaneously; in addition, it cannot handle occlusions.

In this chapter, we present a probabilistic real time tracking framework that combines the motion model of an object with its appearance and position. The motion of the object is modeled using the Gaussian Mixture Background Subtraction algorithm, the appearance, by a color histogram and the projected location of the tracked object in the image space/frame sequence is computed by applying a Gaussian to the Region of Interest. Our tracking framework is robust to abrupt changes in lighting conditions, can follow an object through occlusions, and can simultaneously track multiple moving foreground objects of different types (e.g., vehicles, human, etc.) even when closely spaced. A spatio-temporal filtering algorithm helps in automatic track initialization and a "dynamic" integration of the framework with optical flow allows us to track videos with significant camera motion. A C++ implementation of the framework has outperformed existing visual tracking algorithms on most videos in the Video Image Retrieval and Analysis Tool (VIRAT) [178], TUD [5], and the Tracking-Learning-Detection [89] datasets.

6.2 Contributions

MAPTrack improves on the agile tracker by using mean shift filtering to integrate motion, appearance, shape, and expected position into one hypothesis, providing "soft switching" that is less threshold sensitive and accounts for multiple models simultaneously. Rather than handle occlusion/track loss and becoming stationary in separate modules like previous approaches, they are integrated into the soft switching equations, providing better robustness in poor conditions. This system was published in the International Conference on Computer Vision Theory and Applications (VISAPP) [21], and has a patent pending [129].

6.3 Related Work

A new particle filter - Kernel Particle Filter (KPF) - was proposed in the [36] for visual tracking for multiple objects in image sequences. The idea proposed in [187] shows tracking using a single classifier SVM. A boosting-based approach was proposed in [177] that used a cascade of classifiers for object detection. However, it didn't address the problem of tracking objects through consecutive frames of a video sequence. A spatio-temporal tracking algorithm was proposed in [103] that involved tracking articulated objects in image sequences through self-occlusions and changes in viewpoint. However, they did not provide capabilities for automatic track initialization or tracking multiple objects.

The TLD algorithm proposed in [89] is the basis of one of the well-known frameworks for tracking moving objects. The TLD framework does not start tracks automatically; it lacks a multi-object tracking feature. Also, TLD is based on template matching and hence fails for videos with multiple numbers of similar looking objects (e.g., in the Indian driving scene video, Figure 6.5). The approach proposed in [115] uses color histograms as the only feature. They use a cascade composition of a particle filter and mean shift. The method proposed in [14] is similar to the approach proposed in TLD. The difference between the work reported in it and TLD is that they use multiple instances as the positive examples in each frame. However, like TLD, their framework does not start tracks automatically as marking the location of the object initially is a prerequisite. A Bayesian estimation-based object tracking algorithm that takes into account the motion models, shape and appearance constraints has been proposed in [171] but it has trouble when the motion layers are infiltrated with clutter, occlusion etc., another method for detecting event sequences in surveillance videos that is applicable only to low frame rate videos is proposed in (Lombardi and Versino, 2011).

Our approach is based on using the motion model, color histogram, and position information of objects to track them with a recursive probabilistic estimation of the composite model. Unlike the previous approaches, it can simultaneously track multiple moving objects, does not fail significantly when there is no motion, or when the object is occluded, is resistant to clutter, and is also able to initialize tracks without human supervision.

Recently, there have been a lot of works that combine multi-object tracking, multi-person tracking, and association between different tracked objects for activity recognition [137]. Our framework tracks multiple objects in a video in each frame or multiple frames efficiently; this capability could be used as a part of a co-related and collective activity recognition framework.

6.4 The Proposed Approach

Figure 6.1 shows the schematic of our approach. First, a moving object must be automatically identified as part of the foreground. This involves track initialization at particular pixels on the subsequent frames that have a higher probability of being part of the moving foreground object. This is achieved by - 1) stabilizing the image and 2) feeding the stabilized image to the spatial and temporal filtering algorithms described below. Issues such as camera instability (shaking, panning, rotating) come into play and require image stabilization for the tracking. These issues and the components of the tracking framework are described in detail below.

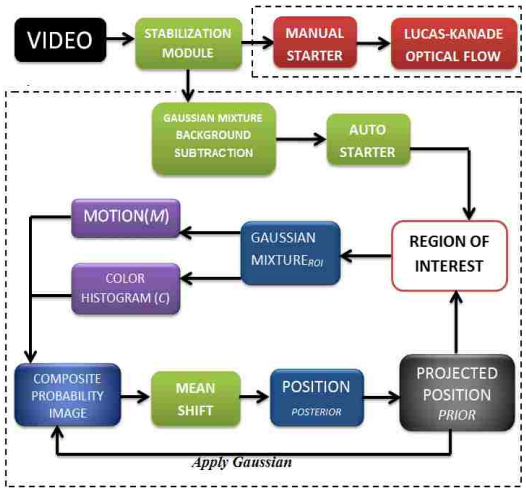

Figure 6.1: Schematic representation of our approach.

6.4.1 Image Stabilization

In order to stabilize an incoming streaming video, we use an iterative algorithm which attempts to hold each background pixel in the same position regardless of lateral and rotational camera motion.
1. Apply Shi and Tomasi's edge-finding algorithm to the first frame to identify significant feature points in the image.
2. For each subsequent frame, apply Lucas-Kanade optical flow to track the motion of the features identified by Shi and Tomasi's algorithm, refreshing the feature points when necessary.
3. With increasing precision for each iteration:
(a) For each angle of rotation in a certain range, determine the translation of each point.
(b) Find the most common translation/rotation (mode) pair (?, x) and (?, y) of all the features.
4. Warp the image to adjust for the total mode of the motion.
Before adjusting for background motion, we must identify features of the frame; to do so, we use the Shi-Tomasi method [157]. The Shi-Tomasi method detects features such as corners and edges by approximating the weighted sum of squares of image patches shifted by certain values.

Next, we apply a pyramidal Lucas-Kanade method [114] for determining optical flow at each point of interest. We then find the mode of the resulting flow value pairs, including rotation, by placing the pairs in bins. At every iteration, the bin widths are decreased, yielding an increasingly accurate estimate of the motion. The image is then adjusted to account for the determined background movement. When the image is stabilized in this manner, fewer false foreground objects detected and correct coordinates of objects are also maintained.

6.4.2 Automated Track Initialization

The automated track initialization algorithm based on a confidence-based spatio-temporal filtering algorithm first detects blobs using the GM Background Subtraction method [86]. This yields difference images, which are fed into the spatial filtering module.

Noise Removal through Morphological Operations

The image obtained through the background subtraction algorithm is initially opened and then closed by a structuring element with diameter λ pixels to filter out unnecessary noise. λ depends upon the scale of the video.

Spatial Filtering

Once blobs are detected in the difference images, they are filtered according to their spatial features. Scale information available from the metadata accompanying the videos is used to filter blobs specifically based on their area and orientation. The filtered blobs are then passed as input to the temporal filtering algorithm below.

Temporal Filtering

To filter blobs in the temporal domain we use a *confidence measure*. Each blob has a confidence measure δ associated with it. δ is initially 0 and increases as a blob is detected across successive frames.

The probabilistic framework that we present takes into account three parameters, namely, the motion of the object modeled using the Gaussian Mixture Background Subtraction algorithm, the appearance of the tracked object using a color histogram, and the projected location of the tracked object in the image space/frame sequence computed by applying a Gaussian to the Region of Interest.

Defining an Adaptive Threshold

If the confidence value for a blob exceeds a specified upper threshold σ, a track is started on it. The moment the confidence value for a blob falls beneath a lower threshold τ, the corresponding object is discarded. If the confidence value is between σ and τ, the corresponding blob is maintained in the list of prospective tracks. For videos that have higher noise, clutter and random changes in lighting conditions, as is often the case for outdoor videos taken from moving cameras, the upper threshold value σ is set higher. On the other hand, for videos with more stable conditions σ is set lower because of the lesser probability of encountering random classification noise.

The composite confidence update equation is as follows.

$$\delta = (0.5^{-n}) \vee (-0.5^{-n}) \tag{6.1}$$

6.4.3 The MAPTrack Framework

Motion – The Gaussian Mixture Background subtraction method helps in determining the positional estimates for all *moving* objects in the scene. It is reasonable to consider all *moving* objects to be a part of the foreground. Our framework builds a background model of Gaussians, with a mean and standard deviation for each pixel. If a new pixel does not fit well with the Gaussians, it is considered to be part of the moving foreground.

Appearance – The appearance of any object in a scene is another important parameter in visual tracking. Object appearance can be modeled using the color histogram associated with it. Operationally, the motion image is used as a mask to create histograms of object pixels for each Region of Interest (ROI). Histograms are implemented as 3d RGB histograms with 32 bins in each R, G, and B direction. For example, bin(0,0,0) contains R=0 to 7, G=0 to 7, B=0 to 7, etc.

Histograms are created for foreground (h_{fg}) and background (h_{bg}) components of the current motion image at the current frame. Each bin in a current histogram contains the count of the pixels that fall in that bin. The background histograms are normalized to make the count of pixels in each equal to the number of foreground pixels in the motion image.

$$h_{bg} = \frac{h_{bg} \times \ |h_{fg}|}{|h_{bg}|} \tag{6.2}$$

So, both foreground and background image have magnitude equal to the number of foreground pixels in the motion image (e.g., when the object is stopped, both current-frame histograms have 0 magnitude). The cumulative histograms (H_{fg} and H_{bg}) are updated using a running average:

$$H = \ \frac{\acute{H} \times (n-1) + h}{n} \tag{6.3}$$

where n is minimum of the current frame number and the point at which the average will change to exponential decay and $\acute{H}$ is the cumulative histogram value from the last frame.

A probability image is created for the pixels in the ROI from the Bayes equation:

$$\begin{aligned} P\ \ (FG \mid \hat{z}) &= \frac{P\left(\hat{z} \mid FG\right) \times \ P\left(FG\right)}{P\left(\bar{z}\right)} \\ &= H\left(\hat{z}\right) \times \frac{avgFG}{\left(H\left(\hat{z}\right) \times avgFG + \left(H\left(\hat{z}\right) \times (1 - avgFG)\right)\right.} \\ P(x,y) &= \begin{cases} 1, & \text{if } P(FG|\hat{z}) > 0.5, \\ 0, & \text{otherwise.} \end{cases} \end{aligned} \tag{6.4}$$

where, avgFG is the sum of the motion history image described below. In other words, if a pixel color is more likely to lie in the object foreground, it will be '1'. Otherwise it will be '0'.

Projected Position – The estimate of the projected position of an object over an image sequence is another determining factor in visual tracking. The position is estimated using the previous position and estimated velocity:

$$p_{est} = \ \acute{p}_{act} + \ v \tag{6.5}$$

where, v is calculated as:

$$v = \frac{p - \ p_0}{f - f_0} \tag{6.6}$$

Here, f_0 is the nearest previous frame where the object is at a distance of at least 1 ROI width from current position if it exists and $max(0, f\text{-}150)$, otherwise. p_0 is the position at that frame.

A positional probability image for the ROI is created using a conical shape.

$$Prob\,(x,y) = max- \quad (max - min) \times \sqrt{\left(\frac{x - c_x}{c_x}\right)^2 + \left(\frac{y - c_y}{c_y}\right)^2} \tag{6.7}$$

where (c_x, c_y) is the center of the ROI, max is the probability value at the center that is equal to 1, and min is the probability at the edges.

Figure 6.2: a) Image b) Motion Pixels c) Appearance Pixels d) Projected Position Pixels

This image represents the estimated position or velocity of the object, and reduces movement from this estimated location. Thus, the probability is highest where the object is most likely to be present (in the center).

In addition, a motion history image is created to estimate the probable object shape, size, and location within the ROI. Similar to the color histograms, it is updated as:

$$mh\,(f) = \frac{(mh\,(f-1) \times historysize\,(f-1) + \ MC \times w)}{(historysize\,(f-1) + w)} \tag{6.8}$$

where, $historysize(f) \ = \ min(historysize\,(f-1) \ + w, \ N)$, and $w \ = \ \sum_{i=1}^{T} (\frac{MC}{ROI_{area}})$ as a scale factor based on the amount of movement present. N is again the point at which the average will change to exponential decay. MC is the image of all moving pixels in the ROI matching the foreground color, as determined by the color histogram of the object. T represents all the pixels in ROI. Each of the Motion, Color, and Positional probability images is centered over the estimated position calculated above. Once the images are obtained, they are combined into a *composite probability image (CPI)* by using the following equation:

$$CPI \ = \ max \ (M \times C \times P \times \sigma_1, \ C \times P \times \ \sigma_2, \ P \times \ \sigma_3) \tag{6.9}$$

Here, σ_1, σ_2, and σ_3 are parameters that determine the relative weights given to the Motion, Color histogram, and Positional Probability images respectively towards the Composite Probability Image *I*. Intuitively, equation (6.9) is a exclusive OR over the values $M \times C \times P$, $C \times P$ and P where the $C \times P$ or P parts are used only when the M value is 0 and P is used only when both M and C are 0. It should be noted that $M \times C \times P$ uses the conical probability image for P, to utilize any motion of matching color within the ROI, whereas $C \times P$ uses the motion history image for P, such that still background of a matching color will not cause a track loss.

Since the motion probability image is the most important parameter for object tracking, *MCP* is assigned the highest weight. The color histogram probability image is less important, followed by the positional probability image. In fact, we found that the P image alone does not work well to deal with occlusions due to the effective velocity of the object decreasing immediately before an occlusion.

The occlusion detection algorithm described below is instead used to handle occlusions and changing lighting conditions. Finally, the mean shift algorithm is used to compute the actual position of the object by shifting the estimate to the new Center of Mass (COM) of the current observation. The mean shift equation is given in equation (6.10).

$$Pos_{act}(f) \; = \; Pos_{est}(f) \; + \; COM(f) \tag{6.10}$$

where, $Pos_{act}(f)$ is the actual position computed at frame f, $Pos_{est}(f)$ is the estimated position at frame f and $COM(f)$ is the Center of Mass used by the mean shift algorithm for estimating the actual position of the object at frame f.

So, the mean shift gives the posterior probability distribution given the prior and the likelihood function. The positional estimate for the actual object location generated by the mean shift algorithm for a given frame f is used to compute the positional estimate for the next frame $f+1$ according to equation (6.5) and the system continues.

Occlusion Detection – The problem with using the position probability image (P) to handle an occlusion was primarily due to the decreasing effective velocity (since the occluded edge is not effectively moving, the velocity of the center of mass effectively reduces) of the object prior to the occlusion since the partially occluded center of mass moves at approximately half of the actual velocity of the object. Since P would only be used where $M \times C \times P$ and $C \times P$ are very small, a metric is instead used to detect an occlusion:

$$occval \; = \sum_{i=1}^{T} \frac{(CP_{motionhistory})^2}{\sum_{i=1}^{T} C} \tag{6.11}$$

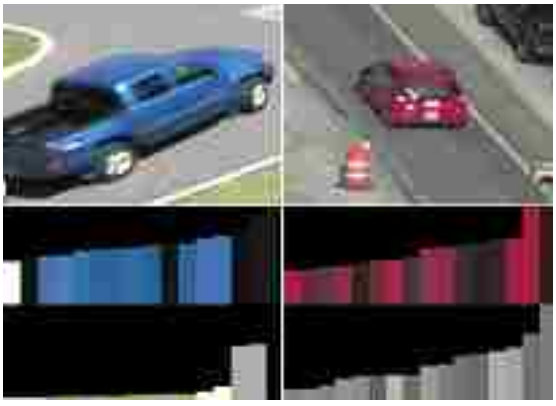

Figure 6.3: Foreground and Background Color Histograms of the two cars.

$\mathrm{CP}_{motionhistory}$ is the estimate of the amount of color matching in the object foreground, and C is the amount of matching color in all of the ROI. Thus, *occval* will be small when either the amount of color in the object is small, or the amount of matching color in the background is very large. An occlusion is detected if:

$$occval(f) \ < \ t_{occ} \times max(occval(n)) \tag{6.12}$$

where, n is a frame number between f_0 and f with f being the current frame and $f_0$the nearest frame where the object is at least one ROI width distance from the current position, or max(1, f-150) if that doesn't exist, and t_{occ} is the threshold for occlusion. When an occlusion is detected, the velocity from frame f_0 is used as an estimate of the current velocity, and the current position is adjusted to reflect that velocity.

$$p_{est}(f) \ = \ p_{est}(f_0) \ + \ v(f_0) \times \ (f - f_0) \tag{6.13}$$

This estimated velocity remains the same while the object is occluded. The ROI is allowed to drift up to half its length from the estimated position towards the center of mass while occluded to allow it to jump to the object when it is again present. The occlusion is ended when significant motion of matching color is again present:

$$\sum_{i=1}^{N} MCP \times \left(\frac{\sum_{i=1}^{T} CP_{motionhistory}}{\sum_{i=1}^{N} C} \right) \ > \ \tau * max\left(occval\left(n\right)\right) \tag{6.14}$$

where, τ is the threshold to end the occlusion, currently set to 0.3. If the occlusion does not resolve within 150 frames or 3 ROI widths, whichever is smaller, the track is ended.

6.5 Implementation Details

The tracking algorithm was implemented in C++ using the OpenCV library [30] for real-time computer vision. The experiments were conducted on an Intel I5 machine with 6 gigabytes of memory.

6.6 Results and Comparative Studies

We compare results from our tracker against existing trackers whose outputs are available at the publicly available TLD dataset [89]. Table 6.1 shows the different states of the tracked object inferred at different values for the Motion, Color and Positional Probability images. Table 6.3 gives the number of frames up to the first track loss for the TUD dataset [5]. It can be seen that MAPTrack outperforms the TUD Detector on both categories of the TUD Dataset. Table 6.2 shows the number of frames after which the trackers lost track for the first time. MAPTrack outperforms other trackers in all of the cases (except motocross). TLD is based on template matching and hence fails for videos with multiple similar looking objects. This is illustrated in Figure 6.4 where TLD switches tracks arbitrarily between similar looking foreground objects whereas MAPTrack keeps tracking a particular object for the entire time frame of its visibility. We also compare our tracker against the TUD Pedestrian Detector [5] for multi-object tracking. The performance metric used was taken from in [160]. Figure 6.7 shows the ROC curve for the tracker and Figure 6.8 shows the results from MAPTrack. Table 6.4 lists the results for occlusion on videos from the /acrshortvirat public dataset available online [178].

Our system performed object tracking plus some simple event detections at an average of 86.0027 frames per second per track, varying somewhat depending on image complexity and resolution. This is approximately 3 times real time, per track, but it is worth noting that the process can be multithreaded per track, and many of the underlying algorithms

Figure 6.4: Output from MAPTrack (Left) and TLD (right). TLD switches randomly between similar objects in noisy videos.

can be further sped up at least an order of magnitude by parallel processing on a video card.

Table 6.1: The different states of the tracked object.

Motion	**Color Histogram**	**Projected Position**	**Inferred State**
0	0	0	Lost Track
0	0	1	Occlusion
0	1	0	Wrong Object
0	1	1	Stopped
1	0	0	Wrong Object
1	0	1	Wrong Object
1	1	0	Wrong Object
1	1	1	Moving Object

Figure 6.5: MAPTrack results for TUD videos.

6.7 Conclusions

We presented a robust tracking framework that uses a probabilistic scheme to combine a motion model of an object with that of its appearance and an estimation of its position. Our tracking framework is robust to abrupt changes in lighting conditions, can follow an object through occlusions. The track starts automatically based on a spatio-temporal algorithm.
It can also simultaneously track multiple moving foreground objects of different types (e.g., vehicles, human, etc.) even when they are closely spaced. A "dynamic" integration of the framework with optical flow allows us to track videos resulting from significant camera motion.

We plan to use the results generated by the tracking algorithm to infer trajectory-based events like vehicle turns as well as other complex events like accidents and traffic violations.

Table 6.2: Comparison of single-object trackers in (Kalal et al., 2010) with MAPTrack. Shows the number of frames after which the trackers lost track for the first time

Algorithms	**Jumping** (frames=313)	**Car** (frames=45)	**Motocross** (frames=2665)	**Car chase** (frames=9928)	**Panda** (frames=3000)
Beyond semi-supervised	14	28	6	66	130
Co-trained Generative-Discriminative	11	34	1	1	1
"CVPR" results	96	29	59	334	358
Online Multiple Instance Learning	313	220	63	321	992
Online Boosting	26	545	-	-	-
Semi-Supervised Online Boosting	21	652	59	190	83
TLD	313	802	173	244	277
MAPTrack	**313**	**821**	**162**	**402**	**2568**

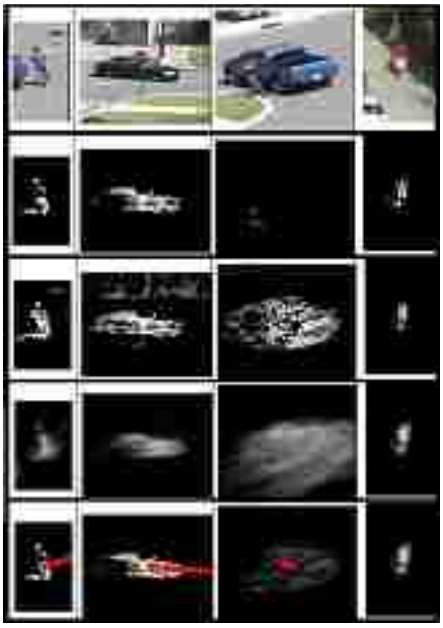

Figure 6.6: Image of people and cars, the images are the ROI images, followed by MCP, CP, Velocity Image and the Weighted Composite Image from top to bottom.

Table 6.3: Tracker results for TUD (Andriluka et al., 2008).

	Campus Correct (False)	Crossing Correct (False)
Expected	303	1008
TUD Detector	227(0)	692(7)
MAPTrack	**255(0)**	**723(5)**

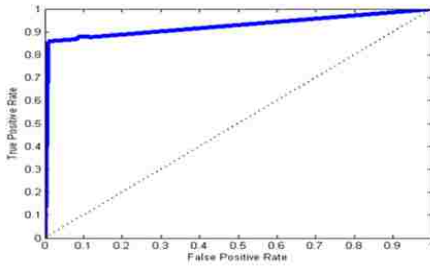

Figure 6.7: ROC curve for the tracker.

Figure 6.8: Results from MAPTrack.

Table 6.4: Results from the tracker (Metric used as in (Smith et al., 2005) [160]). Scores for configuration distance, multiple objects, multiple tracks, false positives, and tracker purity are defined.

Video	Duration	CD	MO	MT	FP	TP	Occlusions	TP	FP
VIRAT_S_010000_01_000184_000324	1m 49s	1.3823	0.036145	0.039078	0.008117	0.86488	20	16	3
VIRAT_S_040003_02_000197_000552	5m 54s	1.0281	0.020574	0.068743	0.007356	0.8692	25	21	4
VIRAT_S_050000_05_000696_000732	0m 35s	4.0775	0.089407	0.047252	0.007937	0.73795	3	3	0

Chapter 7
Conclusions and Future Work

7.1 Food Analysis

Implementing the system described in chapter 4 gave us new insights into the problems of food recognition. Our experiments show that the 3 major problems of food analysis are iterative segmentation, ranked classification with many classes, and food volume estimation. We provided solutions to 2 of these 3 problems with good results, and provided some insight into the third problem of iterative segmentation.

The full confusion matrix for our experiment with 110 food types is shown in Figure 7.1. Each row corresponds to images of a single food type, and each column corresponds to what those images were actually classified as. Brighter squares represent higher confidence of a match. The diagonal line down the center represents correct classifications. Single bright spots away from the center represent food pairs that can easily be confused with each other. The brighter horizontal lines under a few food classes represent food classes that contain many different possible colors, such that most foods can get confused for them.

We analyzed the erroneous classifications in our experiments and identified several notable sources of error. Lighting differences between training and testing sets caused problems, even with lighting normalization. This could be mitigated but not solved by using more sophisticated lighting normalization. Another possibility would be to get much more training data under different lighting conditions, or to simulate different lightings during training. Another source of error was testing foods that looked significantly different from any of the training samples of that type. We later mitigated the problem by adding online retraining, the other obvious solution would be to use a lot more training data with many different-looking versions of each food type. Finally, some types of salads with many subfoods matched well against most foods. This was not a problem with a few hundred types since the correct foods generally ranked higher, but as the system scales up several more orders of magnitude, having these types cluttering up the top tiers of the ranked list may eventually become a problem. To correct for this source of confusion, features targeted at these food types should be included in the feature extractor - either part-based features along the lines of [198], or global features like those used in some non-rigid object trackers [34].

7.2 Tracking

Our agile tracking framework described in chapter 5 provided a novel method for tracking in low quality videos, and we showed that in cases where the tracker's error rate can be estimated that it holds up well against similar robust-type trackers. Our agile framework allows dynamic tracker switching based on performance measure, resulting in a tracker with more accuracy than any of the individual algorithms. Our tracker has motion based automatic track starting, tracks multiple similar objects simultaneously, and accounts for near passes of similar objects.

Our MAPTrack, described in chapter 6, built on many of the features of the agile tracker, introducing shape and appearance modeling, and soft switching between tracking models to allow information from multiple pertinent models to be combined. We showed

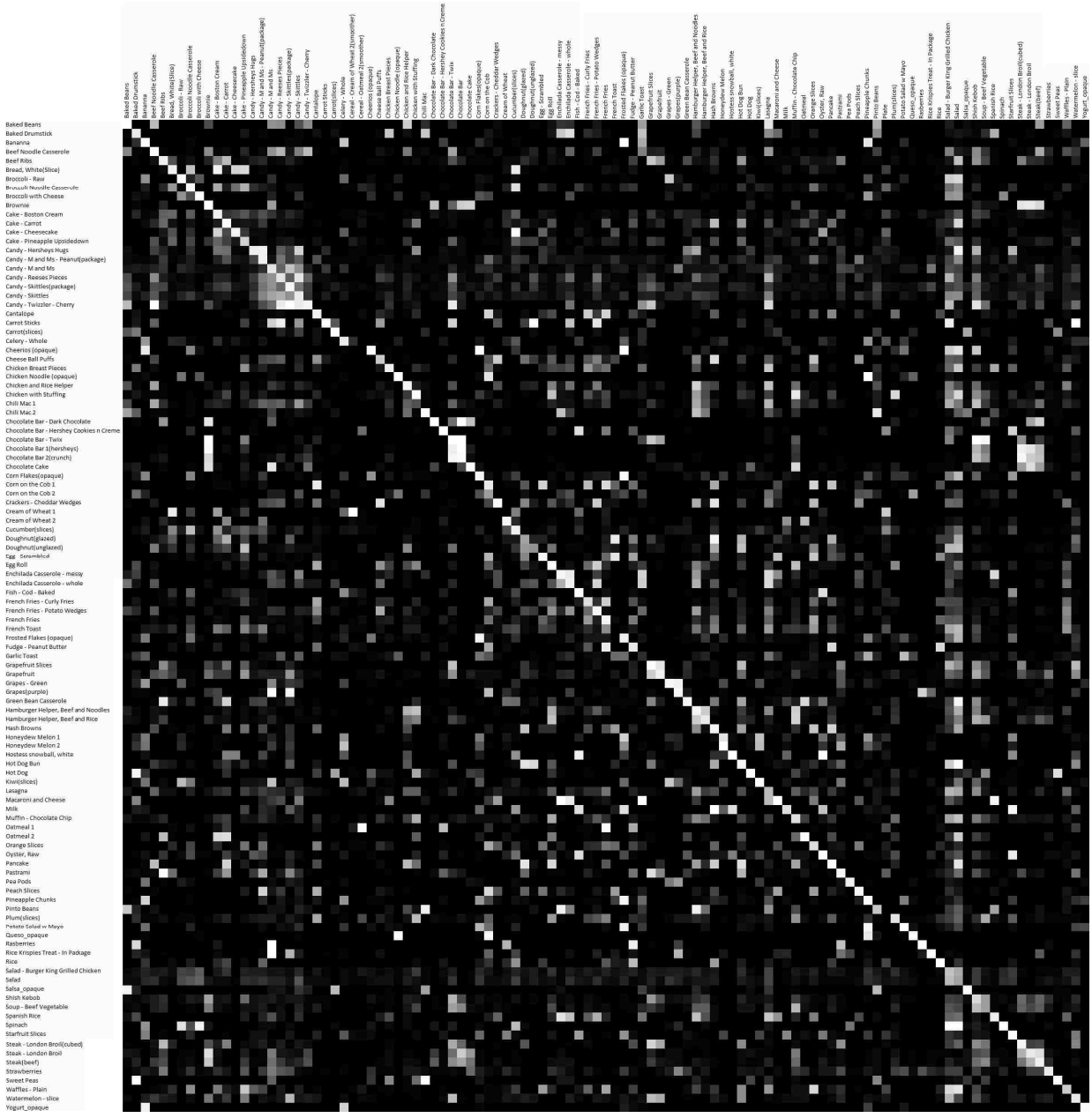

Figure 7.1: Confusion matrix.

that MAPTrack provided very good performance on a number of tracking problems videos of cars and humans, low quality videos, and videos with nonrigid moving objects. MAP-Track uses a probabilistic scheme to combine motion, appearance, and position information as available, and can follow an object through sudden changes in lighting and past occlusions.

7.3 Future Work - Food Recognition

Iterative Segmentation

There is the most room for future work in food segmentation, since the multifood segmentation problem is algorithmically interesting and has yet to be conclusively solved. We attempted segmentation by classification with no success on large numbers of foods, but other research suggests this could be handled by doing an initial segmentation with only a few generalized food groups [8]. With some kind of initial segmentation, be it color, texture, or food groups based, the votes at each patch could be used to region merge by classification. Alternatively, some kind of clustering could be used to automatically form a small set of common food groups for initial segmentation.

Without the assumption of one food per plate, there are 2 types of multimode regions that are difficult to segment. Firstly there are regions with several large regions with different colors, such as lemon meringue pie. Second there are regions that have higher frequency fluctuations between several different colors such as lasagna or mixed vegetables. It may be possible to use the classification data after the fact to intelligently merge nearby regions, even if they have different colors/textures. Since the system is based on voting block by block, you simply merge the 2 final vote tallies to reclassify a region after a merge. This makes the first type approachable. The second type can be approached the same way, with the additional option of instituting a minimum initial segment size to stop a multimode food from being able to fragment into dozens of regions. A split would be more problematic, requiring a re-voting.

Segmenting by very broad classification into food groups has shown some good results in other systems, but no automatic system for forming these groups has been proposed. It would be helpful if the groups were something that a classifier can distinguish between; if every color/texture of food is represented equally in every group, classification would be difficult even if the groups were correct. On the other hand, analyzing actual meals and finding which foods don't co-occur in the same meal would be the direct way of finding the true food groups. So segmentation would ideally require an optimization between these 2 constraints.

Ranked Classification

Convolution neural networks have shown a lot of promise for object recognition lately; but the consensus seems to be that they aren't any better, and are probably worse, than the old hand crafted Haralick features [72] for texture analysis. So most of the improvements to be made in the classification are algorithmic improvements to improve speed, and new handcrafted features that work well on the dataset.

If a specialist network does not get many positive results, it can stop running as soon as it can be confident it is statistically improbable that its food will be near the top of the results list. Similarly, foods near the top of the list no longer need to be queried once it is

statistically improbable they will change rankings during the rest of the search. This could speed up classification by orders of magnitude at minimal loss in accuracy.

Histograms can capture detailed color data, but can produce very high dimensional feature vectors, which are correlated and individually very weak classifiers. Histogram difference is the distance between the histogram of the current region and a "baseline" histogram stored for the target food type. So the distance between a given sample and every known food can be computed. Although this actually generates one feature per food type, each classifier only needs to check the distance associated with its assigned food; meaning a fairly meaningful histogram analysis can be computed with only a single additional distance feature for each classifier.

By grouping foods that match well against each training image, it may be possible to infer which foods are similar and form food clusters. Having specialist networks for recognizing each cluster could be another way to extend the algorithm to even larger numbers of foods. This type of hierarchical implementation is useful for truly large numbers of classes - in our case it is very important that foods that often get confused for each other be in the same cluster.

The simplest approach would be to cluster foods based on their coordinates in the feature space. A better way might be to form a high multidimensional solution space by using our classifier on every type of training sample, and clustering this way. But the high dimensionality would be a problem for many clustering algorithms. A compromise would be to use an unlabeled training method; which creates a solution space with any chosen number of dimensions. Since the data is unlabeled, these outputs don't correspond to actual classifications; but the clusters should correspond to the same things the final classifier would find similar.

Volume Estimation

Our volume estimation doesn't work as well on liquid foods; it could benefit from some way of identifying known containers. This would need to use both information on the user that had taken the image, and some sort of container detection/matching method. In order to significantly improve accuracy in other cases, a good 2 image estimation method or 3d scanning hardware would probably be required.

Other Future Work

Because most lighting was standardized for our dataset, only crude naive "white patch" lighting normalization was ever implemented. A better method should be used in future work. Scale normalization was not done before classification for the same reason, but should probably be done in the more general case as well.

It is worth noting that the "big data" approach of using massive amounts of training data from the Internet would probably solve the lighting problem and the problem of food with many possible appearances. This method has recently showed good results in similar object detection tasks.

7.4 Future Work - Tracking

As discussed in chapter 2.2.15, the best approaches to surveillance are application specific [49]. Our dataset contained both low quality aerial and good quality ground surveil-

lance videos. The extreme variance in quality within the /acrshortvirat dataset [178] combined with the requirement to automatically start tracks limits the available approaches to the very broadly applicable ones that will solve both problems.

Trackers like [162] that separate tracking into subproblems, or like [91] that optimize between tracking and detection seem to often produce the most practical trackers. Optimizing between a tracker suitable for high quality and low quality videos might be a good direction for future work.

An expanded approach might be to split the problem into 2 separate subproblems for the 2 major data types, and handle them with different trackers (via the agile framework). For example, classifier grids [148] are a powerful tool for fixed camera surveillance, but are useless for the aerial dataset. There are any number of more sophisticated approaches that give good results but don't apply to the low quality aerial data. The agile framework could be bolstered with expert knowledge on these 2 problem classes if necessary to let it distinguish accurately between them.

Similarly, our tracker could be integrated with stronger detection than our simple appearance model. While offline learning is not an option, our tracker could take advantage of online learning, using the existing tracker to bootstrap initial foreground and background models. After a few frames the classification could be integrated into the appearance model. This would add strong tracking by detection that could be used to simplify occlusion detection, track loss, and recovery of lost tracks.

We plan to use the results generated to infer trajectory based events such as different types of turns and traffic maneuvers, collisions, entering and exiting vehicles and buildings, etc.

References

[1] A. Adam, E. Rivlin, and I. Shimshoni. Robust fragments-based tracking using the integral histogram. In *Computer Vision and Pattern Recognition, 2006 IEEE Computer Society Conference on*, volume 1, pages 798–805, June 2006.

[2] Shivani Agarwal and Dan Roth. Learning a sparse representation for object detection. In Anders Heyden, Gunnar Sparr, Mads Nielsen, and Peter Johansen, editors, *Computer Vision ECCV 2002*, volume 2353 of *Lecture Notes in Computer Science*, pages 113–127. Springer Berlin Heidelberg, 2002.

[3] LSU agile framework tracker. `https://xythos.lsu.edu/users/mstagg3/web/tracker/`. Accessed: 2013.

[4] R. Almaghrabi, G. Villalobos, P. Pouladzadeh, and S. Shirmohammadi. A novel method for measuring nutrition intake based on food image. In *Instrumentation and Measurement Technology Conference (I2MTC), 2012 IEEE International*, pages 366–370, May 2012.

[5] M. Andriluka, S. Roth, and B. Schiele. People-tracking-by-detection and people-detection-by-tracking. In *Computer Vision and Pattern Recognition, 2008. CVPR 2008. IEEE Conference on*, pages 1–8, June 2008.

[6] Robert Andrle. The angle measure technique: A new method for characterizing the complexity of geomorphic lines. *Mathematical Geology*, 26(1):83–97, 1994.

[7] Anhui vision Optoelectronics Technology Co., Ltd. http://en.ahvision.cn/. Accessed: 04/14/2015.

[8] M. Anthimopoulos, L. Scarnato, P. Diem, and S. Mougiakakou. SEGMENTATION AND RECOGNITION OF FOOD IMAGES FOR CARBOHYDRATE COUNTING. *DIABETES TECHNOLOGY & THERAPEUTICS*, 15(1):A99, FEB 2013.

[9] Lenore Arab and Ashley Winter. Automated camera-phone experience with the frequency of imaging necessary to capture diet. *Journal of the American Dietetic Association*, 110(8):1238 – 1241, 2010.

[10] M.S. Arulampalam, S. Maskell, N. Gordon, and T. Clapp. A tutorial on particle filters for online nonlinear/non-gaussian bayesian tracking. *Signal Processing, IEEE Transactions on*, 50(2):174–188, Feb 2002.

[11] ASA24. http://riskfactor.cancer.gov/tools/instruments/asa24/. Accessed: 03/14/2012.

[12] S. Avidan. Support vector tracking. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 26(8):1064–1072, Aug 2004.

[13] S. Avidan. Ensemble tracking. In *Computer Vision and Pattern Recognition, 2005. CVPR 2005. IEEE Computer Society Conference on*, volume 2, pages 494–501 vol. 2, June 2005.

[14] B. Babenko, Ming-Hsuan Yang, and S. Belongie. Visual tracking with online multiple instance learning. In *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on*, pages 983–990, June 2009.

[15] M. O. Balaban. Quantifying Nonhomogeneous Colors in Agricultural Materials Part I: Method Development. *JOURNAL OF FOOD SCIENCE*, 73(9):S431–S437, NOV-DEC 2008.

[16] M. O. Balaban, J. Aparicio, M. Zotarelli, and C. Sims. Quantifying Nonhomogeneous Colors in Agricultural Materials. Part II: Comparison of Machine Vision and Sensory Panel Evaluations. *JOURNAL OF FOOD SCIENCE*, 73(9):S438–S442, NOV-DEC 2008.

[17] Tom Baranowski, Janice C Baranowski, Kathleen B Watson, Shelby Martin, Alicia Beltran, Noemi Islam, Hafza Dadabhoy, Su-heyla Adame, Karen Cullen, Debbe Thompson, Richard Buday, and Amy Subar. Childrens accuracy of portion size estimation using digital food images: effects of interface design and size of image on computer screen. *Public Health Nutrition*, 14:418–425, 3 2011.

[18] J.L. Barron, D.J. Fleet, S.S. Beauchemin, and T.A. Burkitt. Performance of optical flow techniques. In *Computer Vision and Pattern Recognition, 1992. Proceedings CVPR '92., 1992 IEEE Computer Society Conference on*, pages 236–242, Jun 1992.

[19] R.W Bass, V.D Norum, and L Schwartz. Optimal multichannel nonlinear filtering. *Journal of Mathematical Analysis and Applications*, 16(1):152 – 164, 1966.

[20] Saikat Basu, Robert Di Biano, Manohar Karki, Malcom Stagg, Jerry Weltman, Supratik Mukhopadhyay, and Sangram Ganguly. An agile framework for real-time motion tracking. COMPSAC, 2015.

[21] Saikat Basu, Manohar Karki, Malcolm Stagg, Robert DiBiano, Sangram Ganguly, and Supratik Mukhopadhyay. Maptrack - a probabilistic real time tracking framework by integrating motion, appearance and position models. In *Proceedings of the 10th International Conference on Computer Vision Theory and Applications*, pages 567–574, 2015.

[22] MH Bharati, JJ Liu, and JF MacGregor. Image texture analysis: methods and comparisons. *CHEMOMETRICS AND INTELLIGENT LABORATORY SYSTEMS*, 72(1):57–71, JUN 28 2004.

[23] Michael J. Black and Allan D. Jepson. Eigentracking: Robust matching and tracking of articulated objects using a view-based representation. *International Journal of Computer Vision*, 26(1):63–84, 1998.

[24] Andrew Blake, Pushmeet Kohli, and Carsten Rother. *Markov Random Fields for Vision and Image Processing*. The MIT Press, 2011.

[25] Marc Bosch, Fengqing Zhu, Nitin Khanna, Carol J. Boushey, and Edward J. Delp. Food texture descriptors based on fractal and local gradient information. August 2011.

[26] Jean-Yves Bouguet. Pyramidal implementation of the lucas kanade feature tracker description of the algorithm, 2000.

[27] C. J. Boushey, D. A. Kerr, J. Wright, K. D. Lutes, D. S. Ebert, and E. J. Delp. Use of technology in children's dietary assessment. *Eur J Clin Nutr*, 63 Suppl 1:S50–57, Feb 2009.

[28] Y Boykov, O Veksler, and R Zabih. Fast approximate energy minimization via graph cuts. *IEEE TRANSACTIONS ON PATTERN ANALYSIS AND MACHINE INTELLIGENCE*, 23(11):1222–1239, NOV 2001.

[29] Y.Y. Boykov and M.-P. Jolly. Interactive graph cuts for optimal boundary amp; region segmentation of objects in n-d images. In *Computer Vision, 2001. ICCV 2001. Proceedings. Eighth IEEE International Conference on*, volume 1, pages 105–112 vol.1, 2001.

[30] G. Bradski. Opencv library. *Dr. Dobb's Journal of Software Tools*, 2000.

[31] T Brosnan and DW Sun. Inspection and grading of agricultural and food products by computer vision systems - a review. *COMPUTERS AND ELECTRONICS IN AGRICULTURE*, 36(2-3):193–213, NOV 2002. International Conference on Engineering and Technological Sciences, BEIJING, PEOPLES R CHINA, OCT 11-14, 2000.

[32] T.F. Burks, S.A. Shearer, J.R. Heath, and K.D. Donohue. Evaluation of neural-network classifiers for weed species discrimination. *Biosystems Engineering*, 91(3):293–304, 2005.

[33] Emmanuel J. Candès, Xiaodong Li, Yi Ma, and John Wright. Robust principal component analysis? *J. ACM*, 58(3):11:1–11:37, June 2011.

[34] L. Cehovin, M. Kristan, and A. Leonardis. An adaptive coupled-layer visual model for robust visual tracking. In *Computer Vision (ICCV), 2011 IEEE International Conference on*, pages 1363–1370, Nov 2011.

[35] J. Chae, I. Woo, S. Kim, R. Maciejewski, F. Zhu, E. J. Delp, C. J. Boushey, and D. S. Ebert. Volume Estimation Using Food Specific Shape Templates in Mobile Image-Based Dietary Assessment. *Proc SPIE*, 7873:78730K, Feb 2011.

[36] Cheng Chang and R. Ansari. Kernel particle filter for visual tracking. *Signal Processing Letters, IEEE*, 12(3):242–245, March 2005.

[37] Cheng Chang, R. Ansari, and A. Khokhar. Multiple object tracking with kernel particle filter. In *Computer Vision and Pattern Recognition, 2005. CVPR 2005. IEEE Computer Society Conference on*, volume 1, pages 566–573 vol. 1, June 2005.

[38] Rama Chellappa and Shankar Chatterjee. Classification of textures using gaussian markov random fields. *Acoustics, Speech and Signal Processing, IEEE Transactions on*, 33(4):959–963, 1985.

[39] Cheng-Lun Chen and Shao-Hua Lin. Intelligent color temperature estimation using fuzzy neural network with application to automatic white balance. In *Systems Man and Cybernetics (SMC), 2010 IEEE International Conference on*, pages 796–803, Oct 2010.

[40] D. Comaniciu, V. Ramesh, and P. Meer. Real-time tracking of non-rigid objects using mean shift. In *Computer Vision and Pattern Recognition, 2000. Proceedings. IEEE Conference on*, volume 2, pages 142–149 vol.2, 2000.

[41] Dorin Comaniciu, Visvanathan Ramesh, Peter Meer, Senior Member, and Senior Member. Kernel-based object tracking. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 25:564–577, 2003.

[42] Richard W Conners and Charles A Harlow. A theoretical comparison of texture algorithms. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, (3):204–222, 1980.

[43] George R Cross and Anil K Jain. Markov random field texture models. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, (1):25–39, 1983.

[44] et.al. D. T. Ross. Systematic variation in gene expression patterns in human cancer cell lines. *Nature Genetics*, 24(3):227–235, 2000.

[45] A. Dahl Lassen, S. Poulsen, L. Ernst, K. Kaae Andersen, A. Biltoft-Jensen, and I. Tetens. Evaluation of a digital method to assess evening meal intake in a free-living adult population. *Food Nutr Res*, 54, 2010.

[46] N. Dalal and B. Triggs. Histograms of oriented gradients for human detection. In *Computer Vision and Pattern Recognition, 2005. CVPR 2005. IEEE Computer Society Conference on*, volume 1, pages 886–893 vol. 1, June 2005.

[47] B. L. Daugherty, T. E. Schap, R. Ettienne-Gittens, F. M. Zhu, M. Bosch, E. J. Delp, D. S. Ebert, D. A. Kerr, and C. J. Boushey. Novel technologies for assessing dietary intake: evaluating the usability of a mobile telephone food record among adults and adolescents. *J. Med. Internet Res.*, 14(2):e58, 2012.

[48] E. R. Davies. The application of machine vision to food and agriculture: a review. *IMAGING SCIENCE JOURNAL*, 57(4):197–217, AUG 2009.

[49] HannahM. Dee and SergioA. Velastin. How close are we to solving the problem of automated visual surveillance? *Machine Vision and Applications*, 19(5-6):329–343, 2008.

[50] J. Dehais, S. Shevchik, P. Diem, and S.G. Mougiakakou. Food volume computation for self dietary assessment applications. In *Bioinformatics and Bioengineering (BIBE), 2013 IEEE 13th International Conference on*, pages 1–4, Nov 2013.

[51] Robert Dibiano, Bahadir K. Gunturk, and Corby K. Martin. Food image analysis for measuring food intake in free living conditions. volume 8669, pages 86693N–86693N–10, 2013.

[52] Chris Ding and Xiaofeng He. K-means clustering via principal component analysis. In *Proceedings of the Twenty-first International Conference on Machine Learning*, ICML '04, pages 29–, New York, NY, USA, 2004. ACM.

[53] CJ Du and DW Sun. Recent developments in the applications of image processing techniques for food quality evaluation. *TRENDS IN FOOD SCIENCE & TECHNOLOGY*, 15(5):230–249, MAY 2004.

[54] CJ Du and DW Sun. Learning techniques used in computer vision for food quality evaluation: a review. *JOURNAL OF FOOD ENGINEERING*, 72(1):39–55, JAN 2006.

[55] Kim H. Esbensen, Kent H. Hjelmen, and Knut Kvaal. The AMT approach in chemometrics first forays. *Journal of Chemometrics*, 10:569–590, 1996.

[56] Norasyikin Fadilah, Junita Mohamad-Saleh, Zaini Abdul Halim, Haidi Ibrahim, and Syed Salim Syed Ali. Intelligent Color Vision System for Ripeness Classification of Oil Palm Fresh Fruit Bunch. *SENSORS*, 12(10):14179–14195, OCT 2012.

[57] Jialue Fan, Wei Xu, Ying Wu, and Yihong Gong. Human tracking using convolutional neural networks. *Neural Networks, IEEE Transactions on*, 21(10):1610–1623, Oct 2010.

[58] Li Fei-Fei, R. Fergus, and P. Perona. One-shot learning of object categories. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 28(4):594–611, April 2006.

[59] Pedro Felzenszwalb, David McAllester, and Deva Ramanan. A discriminatively trained, multiscale, deformable part model. In *Computer Vision and Pattern Recognition, 2008. CVPR 2008. IEEE Conference on*, pages 1–8. IEEE, 2008.

[60] Lorenzo Fongaro and Knut Kvaal. Surface texture characterization of an Italian pasta by means of univariate and multivariate feature extraction from their texture images. *FOOD RESEARCH INTERNATIONAL*, 51(2):693–705, MAY 2013.

[61] Nathan Funk. A study of the kalman filter applied to visual tracking. 2003.

[62] M. Godec, P.M. Roth, and H. Bischof. Hough-based tracking of non-rigid objects. In *Computer Vision (ICCV), 2011 IEEE International Conference on*, pages 81–88, Nov 2011.

[63] Vural Gokmen and Idris Sugut. A Non-Contact Computer Vision Based Analysis of Color in Foods. *INTERNATIONAL JOURNAL OF FOOD ENGINEERING*, 3(5), 2007.

[64] H. Grabner and H. Bischof. On-line boosting and vision. In *Computer Vision and Pattern Recognition, 2006 IEEE Computer Society Conference on*, volume 1, pages 260–267, June 2006.

[65] H. Grabner, M. Grabner, and H. Bischof. Real-time tracking via on-line boosting. In *Proc. BMVC*, pages 6.1–6.10, 2006. doi:10.5244/C.20.6.

[66] H. Grabner, J. Matas, L. Van Gool, and P. Cattin. Tracking the invisible: Learning where the object might be. In *Computer Vision and Pattern Recognition (CVPR), 2010 IEEE Conference on*, pages 1285–1292, June 2010.

[67] Helmut Grabner, Christian Leistner, and Horst Bischof. Semi-supervised on-line boosting for robust tracking. In David Forsyth, Philip Torr, and Andrew Zisserman, editors, *Computer Vision ECCV 2008*, volume 5302 of *Lecture Notes in Computer Science*, pages 234–247. Springer Berlin Heidelberg, 2008.

[68] M. Grabner, H. Grabner, and H. Bischof. Learning features for tracking. In *Computer Vision and Pattern Recognition, 2007. CVPR '07. IEEE Conference on*, pages 1–8, June 2007.

[69] G.D. Hager and P.N. Belhumeur. Efficient region tracking with parametric models of geometry and illumination. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 20(10):1025–1039, Oct 1998.

[70] RM HARALICK. STATISTICAL AND STRUCTURAL APPROACHES TO TEXTURE. *PROCEEDINGS OF THE IEEE*, 67(5):786–804, 1979.

[71] RM HARALICK, SHANMUGA.K, and I DINSTEIN. TEXTURAL FEATURES FOR IMAGE CLASSIFICATION. *IEEE TRANSACTIONS ON SYSTEMS MAN AND CYBERNETICS*, SMC3(6):610–621, 1973.

[72] Robert M. Haralick, K. Shanmugam, and Its'Hak Dinstein. Textural features for image classification. *Systems, Man and Cybernetics, IEEE Transactions on*, 3(6):610–621, 1973.

[73] Chris Harris and Mike Stephens. A combined corner and edge detector. In *In Proc. of Fourth Alvey Vision Conference*, pages 147–151, 1988.

[74] Robin Hewitt. Seeing with opencv, part 3: Follow that face! *SERVO Magazine*, pages 36–40, March 2007.

[75] Berthold K. P. Horn and Brian G. Schunck. Determining optical flow. *ARTIFICAL INTELLIGENCE*, 17:185–203, 1981.

[76] Chih-Wei Hsu and Chih-Jen Lin. A comparison of methods for multiclass support vector machines. *Neural Networks, IEEE Transactions on*, 13(2):415–425, Mar 2002.

[77] J. Illingworth and J. Kittler. A survey of the hough transform. *Comput. Vision Graph. Image Process.*, 44(1):87–116, August 1988.

[78] Michael Isard and Andrew Blake. Condensation - conditional density propagation for visual tracking. *International Journal of Computer Vision*, 29:5–28, 1998.

[79] Patrick Jackman and Da-Wen Sun. Recent advances in image processing using image texture features for food quality assessment. *TRENDS IN FOOD SCIENCE & TECHNOLOGY*, 29(1):35–43, JAN 2013.

[80] Wenyan Jia, Yaofeng Yue, John D. Fernstrom, Ning Yao, Robert J. Sclabassi, Madelyn H. Fernstrom, and Mingui Sun. Imaged based estimation of food volume using circular referents in dietary assessment. *JOURNAL OF FOOD ENGINEERING*, 109(1):76–86, MAR 2012.

[81] Yu-Gang Jiang, Chong-Wah Ngo, and Jun Yang. Towards optimal bag-of-features for object categorization and semantic video retrieval. In *Proceedings of the 6th ACM International Conference on Image and Video Retrieval*, CIVR '07, pages 494–501, New York, NY, USA, 2007. ACM.

[82] A Jordan. On discriminative vs. generative classifiers: A comparison of logistic regression and naive bayes. *Advances in neural information processing systems*, 14:841, 2002.

[83] Taichi Joutou and Keiji Yanai. A food image recognition system with multiple kernel learning. In *Proceedings of the 16th IEEE International Conference on Image Processing*, ICIP'09, pages 285–288, Piscataway, NJ, USA, 2009. IEEE Press.

[84] Simon J. Julier and Jeffrey K. Uhlmann. New extension of the kalman filter to nonlinear systems. volume 3068, pages 182–193, 1997.

[85] P. Kaewtrakulpong and R. Bowden. An improved adaptive background mixture model for realtime tracking with shadow detection, 2001.

[86] P. KaewTraKulPong and R. Bowden. An improved adaptive background mixture model for real-time tracking with shadow detection. In Paolo Remagnino, GraemeA. Jones, Nikos Paragios, and CarloS. Regazzoni, editors, *Video-Based Surveillance Systems*, pages 135–144. Springer US, 2002.

[87] Hokuto Kagaya, Kiyoharu Aizawa, and Makoto Ogawa. Food detection and recognition using convolutional neural network. In *Proceedings of the ACM International Conference on Multimedia*, MM '14, pages 1085–1088, New York, NY, USA, 2014. ACM.

[88] Z. Kalal, J. Matas, and K. Mikolajczyk. Online learning of robust object detectors during unstable tracking. In *Computer Vision Workshops (ICCV Workshops), 2009 IEEE 12th International Conference on*, pages 1417–1424, Sept 2009.

[89] Z. Kalal, J. Matas, and K. Mikolajczyk. P-n learning: Bootstrapping binary classifiers by structural constraints. In *Computer Vision and Pattern Recognition (CVPR), 2010 IEEE Conference on*, pages 49–56, June 2010.

[90] Z. Kalal, K. Mikolajczyk, and J. Matas. Forward-backward error: Automatic detection of tracking failures. In *Pattern Recognition (ICPR), 2010 20th International Conference on*, pages 2756–2759, Aug 2010.

[91] Z. Kalal, K. Mikolajczyk, and J. Matas. Tracking-learning-detection. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 34(7):1409–1422, July 2012.

[92] R. E. Kalman and R. S. Bucy. New results in linear filtering and prediction theory. *Trans. ASME, Ser. D, J. Basic Eng*, page 109, 1961.

[93] S. P. Kang and H. T. Sabarez. Simple colour image segmentation of bicolour food products for quality measurement. *JOURNAL OF FOOD ENGINEERING*, 94(1):21–25, SEP 2009.

[94] L.M. Kaplan and C.-C.J. Kuo. Extending self-similarity for fractional brownian motion. *Trans. Sig. Proc.*, 42(12):3526–3530, December 1994.

[95] Y. Kawano and K. Yanai. Real-time mobile food recognition system. In *Computer Vision and Pattern Recognition Workshops (CVPRW), 2013 IEEE Conference on*, pages 1–7, June 2013.

[96] Yoshiyuki Kawano and Keiji Yanai. Food image recognition with deep convolutional features. In *Proceedings of the 2014 ACM International Joint Conference on Pervasive and Ubiquitous Computing: Adjunct Publication*, UbiComp '14 Adjunct, pages 589–593, New York, NY, USA, 2014. ACM.

[97] Sertan Kaya. QUANTIFICATION OF ENERGY INTAKE USING FOOD IMAGE ANALYSIS. Master's thesis, LSU, USA, 2010.

[98] D. A. Kerr, C. M. Pollard, P. Howat, E. J. Delp, M. Pickering, K. R. Kerr, S. S. Dhaliwal, I. S. Pratt, J. Wright, and C. J. Boushey. Connecting Health and Technology (CHAT): protocol of a randomized controlled trial to improve nutrition behaviours using mobile devices and tailored text messaging in young adults. *BMC Public Health*, 12:477, 2012.

[99] Sujung Kim, Wook-Joong Kim, and Seong-Dae Kim. Automatic white balance based on adaptive feature selection with standard illuminants. In *Image Processing, 2008. ICIP 2008. 15th IEEE International Conference on*, pages 485–488, Oct 2008.

[100] Zuwhan Kim. Realtime object tracking based on dynamic feature grouping with background subtraction. In *Proc. IEEE Comput. Vis. Pattern Recog*, pages 1–8, 2008.

[101] Keigo Kitamura, Toshihiko Yamasaki, and Kiyoharu Aizawa. Food log by analyzing food images. In *Proceedings of the 16th ACM International Conference on Multimedia*, MM '08, pages 999–1000, New York, NY, USA, 2008. ACM.

[102] Junseok Kwon and Kyoung Mu Lee. Highly nonrigid object tracking via patch-based dynamic appearance modeling. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 35(10):2427–2441, Oct 2013.

[103] Xiangyang Lan and D.P. Huttenlocher. A unified spatio-temporal articulated model for tracking. In *Computer Vision and Pattern Recognition, 2004. CVPR 2004. Proceedings of the 2004 IEEE Computer Society Conference on*, volume 1, pages I–722–I–729 Vol.1, June 2004.

[104] B. Leibe, K. Schindler, N. Cornelis, and L. Van Gool. Coupled object detection and tracking from static cameras and moving vehicles. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 30(10):1683–1698, Oct 2008.

[105] B. Leibe, K. Schindler, and L. Van Gool. Coupled detection and trajectory estimation for multi-object tracking. In *Computer Vision, 2007. ICCV 2007. IEEE 11th International Conference on*, pages 1–8, Oct 2007.

[106] Bastian Leibe, Edgar Seemann, and Bernt Schiele. Pedestrian detection in crowded scenes. In *Proceedings of the 2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05) - Volume 1 - Volume 01*, CVPR '05, pages 878–885, Washington, DC, USA, 2005. IEEE Computer Society.

[107] C. Leistner, H. Grabner, and H. Bischof. Semi-supervised boosting using visual similarity learning. In *Computer Vision and Pattern Recognition, 2008. CVPR 2008. IEEE Conference on*, pages 1–8, June 2008.

[108] Xin Li and S. S. Iyengar. On computing mapping of 3d objects: A survey. *ACM Comput. Surv.*, 47(2):34:1–34:45, December 2014.

[109] Yuan Li, Haizhou Ai, T. Yamashita, Shihong Lao, and M. Kawade. Tracking in low frame rate video: A cascade particle filter with discriminative observers of different lifespans. In *Computer Vision and Pattern Recognition, 2007. CVPR '07. IEEE Conference on*, pages 1–8, June 2007.

[110] Huan Liu and Hiroshi Motoda. *Feature Selection for Knowledge Discovery and Data Mining*. Kluwer Academic Publishers, Norwell, MA, USA, 1998.

[111] Tie-Yan Liu, Yiming Yang, Hao Wan, Hua-Jun Zeng, Zheng Chen, and Wei-Ying Ma. Support vector machines classification with a very large-scale taxonomy. *SIGKDD Explor. Newsl.*, 7(1):36–43, June 2005.

[112] Paolo Lombardi and Cristina Versino. Learning to detect event sequences in surveillance streams at very low frame rate. In Liang Wang, Guoying Zhao, Li Cheng, and Matti Pietikinen, editors, *Machine Learning for Vision-Based Motion Analysis*, Advances in Pattern Recognition, pages 117–144. Springer London, 2011.

[113] Philip M. Long and Rocco A. Servedio. Random classification noise defeats all convex potential boosters. *Mach. Learn.*, 78(3):287–304, March 2010.

[114] Bruce D. Lucas and Takeo Kanade. An iterative image registration technique with an application to stereo vision. In *Proceedings of the 7th International Joint Conference on Artificial Intelligence - Volume 2*, IJCAI'81, pages 674–679, San Francisco, CA, USA, 1981. Morgan Kaufmann Publishers Inc.

[115] E. Maggio and A. Cavallaro. Hybrid particle filter and mean shift tracker with adaptive transition model. In *Acoustics, Speech, and Signal Processing, 2005. Proceedings. (ICASSP '05). IEEE International Conference on*, volume 2, pages 221–224, March 2005.

[116] S Majumdar and DS Jayas. Classification of bulk samples of cereal grains using machine vision. *JOURNAL OF AGRICULTURAL ENGINEERING RESEARCH*, 73(1):35–47, MAY 1999.

[117] S Majumdar and DS Jayas. Classification of cereal grains using machine vision: II. Color models. *TRANSACTIONS OF THE ASAE*, 43(6):1677–1680, NOV-DEC 2000.

[118] A. Mariappan, M. Bosch, F. Zhu, C. J. Boushey, D. A. Kerr, D. S. Ebert, and E. J. Delp. Personal Dietary Assessment Using Mobile Devices. *Proc SPIE*, 7246, Jan 2009.

[119] C. K. Martin, J. B. Correa, H. Han, H. R. Allen, J. C. Rood, C. M. Champagne, B. K. Gunturk, and G. A. Bray. Validity of the Remote Food Photography Method (RFPM) for estimating energy and nutrient intake in near real-time. *Obesity (Silver Spring)*, 20(4):891–899, Apr 2012.

[120] C. K. Martin, H. Han, S. M. Coulon, H. R. Allen, C. M. Champagne, and S. D. Anton. A novel method to remotely measure food intake of free-living individuals in real time: the remote food photography method. *Br. J. Nutr.*, 101(3):446–456, Feb 2009.

[121] C. K. Martin, S. Kaya, and B. K. Gunturk. Quantification of food intake using food image analysis. In *IEEE Int. Conf. Engineering in Medicine and Biology Society*, pages 6869–6872, 2009.

[122] C. K. Martin, J. L. Thomson, M. M. LeBlanc, T. M. Stewart, R. L. Newton, H. Han, A. Sample, C. M. Champagne, and D. A. Williamson. Children in school cafeterias select foods containing more saturated fat and energy than the Institute of Medicine recommendations. *J. Nutr.*, 140(9):1653–1660, Sep 2010.

[123] Takuma Maruyama, Yoshiyuki Kawano, and Keiji Yanai. Real-time mobile recipe recommendation system using food ingredient recognition. In *Proceedings of the 2Nd ACM International Workshop on Interactive Multimedia on Mobile and Portable Devices*, IMMPD '12, pages 27–34, New York, NY, USA, 2012. ACM.

[124] Andrzej Materka, Michal Strzelecki, et al. Texture analysis methods–a review. *Technical university of lodz, institute of electronics, COST B11 report, Brussels*, pages 9–11, 1998.

[125] Y. Matsuda, H. Hoashi, and K. Yanai. Recognition of multiple-food images by detecting candidate regions. In *Multimedia and Expo (ICME), 2012 IEEE International Conference on*, pages 25–30, July 2012.

[126] I. Matthews, T. Ishikawa, and S. Baker. The template update problem. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 26(6):810–815, June 2004.

[127] Fernando Mendoza, Petr Dejmek, and Jose M. Aguilera. Calibrated color measurements of agricultural foods using image analysis. *POSTHARVEST BIOLOGY AND TECHNOLOGY*, 41(3):285–295, SEP 2006.

[128] F. Moreno-Noguer, Alberto Sanfeliu, and D. Samaras. Dependent multiple cue integration for robust tracking. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 30(4):670–685, April 2008.

[129] Supratik Mukhopadhyay, Saikat Basu, Robert DiBiano, Malcolm Stagg, Jerry Weltman, and Manohar Karki. Computer implemented system and method for high performance visual tracking, 2013. US Patent App. 61/711,102.

[130] Per Munkevik, Gunnar Hall, and Tom Duckett. A computer vision system for appearance-based descriptive sensory evaluation of meals. *Journal of Food Engineering*, 78(1):246 – 256, 2007.

[131] P. Nagabhushan and R. Pradeep Kumar. Histogram pca. In Derong Liu, Shumin Fei, Zengguang Hou, Huaguang Zhang, and Changyin Sun, editors, *Advances in Neural Networks ISNN 2007*, volume 4492 of *Lecture Notes in Computer Science*, pages 1012–1021. Springer Berlin Heidelberg, 2007.

[132] N. Nakano, R. Nishimura, H. Sai, A. Nishizawa, and H. Komatsu. Digital still camera system for megapixel ccd. *Consumer Electronics, IEEE Transactions on*, 44(3):581–586, Aug 1998.

[133] S. Nashat and M. Z. Abdullah. Multi-class colour inspection of baked foods featuring support vector machine and Wilk's lambda analysis. *JOURNAL OF FOOD ENGINEERING*, 101(4):370–380, DEC 2010.

[134] G. Nebehay and R. Pflugfelder. Tlm: Tracking-learning-matching of keypoints. In *Distributed Smart Cameras (ICDSC), 2013 Seventh International Conference on*, pages 1–6, Oct 2013.

[135] HieuT. Nguyen and ArnoldW.M. Smeulders. Robust tracking using foreground-background texture discrimination. *International Journal of Computer Vision*, 69(3):277–293, 2006.

[136] Kenji Okuma, Ali Taleghani, Nando De Freitas, O De Freitas, James J. Little, and David G. Lowe. A boosted particle filter: Multitarget detection and tracking. In *In ECCV*, pages 28–39, 2004.

[137] Alessandro Oltramari and Christian Lebiere. Using ontologies in a cognitive-grounded system: Automatic action recognition in video-surveillance. In Paulo Cesar G. da Costa and Kathryn B. Laskey, editors, *STIDS*, volume 966 of *CEUR Workshop Proceedings*, pages 20–27. CEUR-WS.org, 2012.

[138] David Opitz and Richard Maclin. Popular ensemble methods: An empirical study. *Journal of Artificial Intelligence Research*, 11:169–198, 1999.

[139] Sebastian Paessler, Matthias Wolff, and Wolf-Joachim Fischer. Food intake monitoring: an acoustical approach to automated food intake activity detection and classification of consumed food. *PHYSIOLOGICAL MEASUREMENT*, 33(6):1073–1093, JUN 2012.

[140] Danny Pascale. Rgb coordinates of the macbeth color checker. 2006.

[141] Alex P Pentland. Fractal-based description of natural scenes. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, (6):661–674, 1984.

[142] Davide Periquito, JacintoC. Nascimento, Alexandre Bernardino, and Joo Sequeira. Single camera hand pose estimation from bottom-up and top-down processes. In Sebastiano Battiato, Sabine Coquillart, Robert S. Laramee, Andreas Kerren, and Jos Braz, editors, *Computer Vision, Imaging and Computer Graphics – Theory and Applications*, volume 458 of *Communications in Computer and Information Science*, pages 212–227. Springer Berlin Heidelberg, 2014.

[143] R. Polikar. Ensemble based systems in decision making. *Circuits and Systems Magazine, IEEE*, 6(3):21–45, Third 2006.

[144] M. Puri, Z. Zhu, J. Lubin, T. Pschar, A. Divakaran, and H.S. Sawhney. Food recognition using visual analysis and speech recognition, July 8 2010. US Patent App. 12/683,124.

[145] M. Puri, Zhiwei Zhu, Qian Yu, A. Divakaran, and H. Sawhney. Recognition and volume estimation of food intake using a mobile device. In *Applications of Computer Vision (WACV), 2009 Workshop on*, pages 1–8, Dec 2009.

[146] D. Ramanan, D.A. Forsyth, and A. Zisserman. Tracking people by learning their appearance. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 29(1):65–81, Jan 2007.

[147] Lior Rokach. Ensemble-based classifiers. *Artificial Intelligence Review*, 33(1-2):1–39, 2010.

[148] P.M. Roth, S. Sternig, H. Grabner, and H. Bischof. Classifier grids for robust adaptive object detection. In *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on*, pages 2727–2734, June 2009.

[149] C. Rother, V. Kolmogorov, and A. Blake. GrabCut interactive foreground extraction using iterated graph cuts. In *SIGGRAPH*, pages 309–314, 2004.

[150] Carsten Rother, Vladimir Kolmogorov, and Andrew Blake. "grabcut": Interactive foreground extraction using iterated graph cuts. *ACM Trans. Graph.*, 23(3):309–314, August 2004.

[151] A. Saffari, C. Leistner, J. Santner, M. Godec, and H. Bischof. On-line random forests. In *Computer Vision Workshops (ICCV Workshops), 2009 IEEE 12th International Conference on*, pages 1393–1400, Sept 2009.

[152] Peter Sand and Seth Teller. Particle video: Long-range motion estimation using point trajectories. *International Journal of Computer Vision*, 80(1):72–91, 2008.

[153] Juliana Freitas Santos Gomes and Fabiana Rodrigues Leta. Applications of computer vision techniques in the agriculture and food industry: a review. *EUROPEAN FOOD RESEARCH AND TECHNOLOGY*, 235(6):989–1000, DEC 2012.

[154] Dayanand G. Savakar and Basavaraj S. Anami. Recognition and Classification of Food Grains, Fruits and Flowers Using Machine Vision. *INTERNATIONAL JOURNAL OF FOOD ENGINEERING*, 5(4), 2009.

[155] T. E. Schap, B. L. Six, E. J. Delp, D. S. Ebert, D. A. Kerr, and C. J. Boushey. Adolescents in the United States can identify familiar foods at the time of consumption and when prompted with an image 14 h postprandial, but poorly estimate portions. *Public Health Nutr*, 14(7):1184–1191, Jul 2011.

[156] Robert E. Schapire. The boosting approach to machine learning: An overview. In David D. Denison, Mark H. Hansen, Christopher C. Holmes, Bani Mallick, and Bin Yu, editors, *Nonlinear Estimation and Classification*, volume 171 of *Lecture Notes in Statistics*, pages 149–171. Springer New York, 2003.

[157] J. Shi and C. Tomasi. Good features to track. In *Computer Vision and Pattern Recognition, 1994. Proceedings CVPR '94., 1994 IEEE Computer Society Conference on*, pages 593–600, Jun 1994.

[158] M. Simonovsky. Ellipse detection using 1d hough transform. http://www.mathworks.com/matlabcentral/fileexchange/33970-ellipse-detection-using-1d-hough-transform. Accessed: 03/14/2012.

[159] B. L. Six, T. E. Schap, F. M. Zhu, A. Mariappan, M. Bosch, E. J. Delp, D. S. Ebert, D. A. Kerr, and C. J. Boushey. Evidence-based development of a mobile telephone food record. *J Am Diet Assoc*, 110(1):74–79, Jan 2010.

[160] K. Smith, D. Gatica-Perez, J. Odobez, and Sileye Ba. Evaluating multi-object tracking. In *Computer Vision and Pattern Recognition - Workshops, 2005. CVPR Workshops. IEEE Computer Society Conference on*, pages 36–36, June 2005.

[161] S. Stalder, H. Grabner, and Luc Van Gool. Exploring context to learn scene specific object detectors. In *IEEE International Workshop on Performance Evaluation of Tracking and Surveillance, Miami, USA*, pages 63–70, June 2009.

[162] S. Stalder, H. Grabner, and L. Van Gool. Beyond semi-supervised tracking: Tracking should be as simple as detection, but not simpler than recognition. In *Computer Vision Workshops (ICCV Workshops), 2009 IEEE 12th International Conference on*, pages 1409–1416, Sept 2009.

[163] Chris Stauffer and W.E.L. Grimson. Adaptive background mixture models for real-time tracking. In *Computer Vision and Pattern Recognition, 1999. IEEE Computer Society Conference on.*, volume 2, pages –252 Vol. 2, 1999.

[164] H. Steinhaus. Sur la division des corp materiels en parties. *Bull. Acad. Polon. Sci*, 1:801–804, 1956.

[165] P. J. Stumbo. New technology in dietary assessment: a review of digital methods in improving food record accuracy. *Proc Nutr Soc*, 72(1):70–76, Feb 2013.

[166] P. J. Stumbo, R. Weiss, J. W. Newman, J. A. Pennington, K. L. Tucker, P. L. Wiesenfeld, A. K. Illner, D. M. Klurfeld, and J. Kaput. Web-enabled and improved software tools and data are needed to measure nutrient intakes and physical activity for personalized health research. *J. Nutr.*, 140(12):2104–2115, Dec 2010.

[167] Levon Sukissian, Stefanos Kollias, and Yiannis Boutalis. Adaptive classification of textured images using linear prediction and neural networks. *Signal Process.*, 36(2):209–232, March 1994.

[168] M. Swanson. Digital photography as a tool to measure school cafeteria consumption. *J Sch Health*, 78(8):432–437, Aug 2008.

[169] Shen-Chuan Tai, Tzu-Wen Liao, Yi-Ying Chang, and Chih Pei Yeh. Automatic white balance algorithm through the average equalization and threshold. In *Information Science and Digital Content Technology (ICIDT), 2012 8th International Conference on*, volume 3, pages 571–576, June 2012.

[170] Hai Tao, H.S. Sawhney, and Rakesh Kumar. Dynamic layer representation with applications to tracking. In *Computer Vision and Pattern Recognition, 2000. Proceedings. IEEE Conference on*, volume 2, pages 134–141 vol.2, 2000.

[171] Hai Tao, H.S. Sawhney, and Rakesh Kumar. Object tracking with bayesian estimation of dynamic layer representations. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 24(1):75–89, Jan 2002.

[172] S. Theodoridis and K. Koutroumbas. Pattern recognition. *Academic Press*, 1999.

[173] F. E. Thompson, A. F. Subar, C. M. Loria, J. L. Reedy, and T. Baranowski. Need for technological innovation in dietary assessment. *J Am Diet Assoc*, 110(1):48–51, Jan 2010.

[174] Jeffrey K Uhlmann. Algorithms for multiple-target tracking. *American Scientist*, pages 128–141, 1992.

[175] M Unser. Sum and difference histograms for texture classification. *IEEE Trans. Pattern Anal. Mach. Intell.*, 8(1):118–125, January 1986.

[176] Martin Vetterli and Jelena Kovacevic. *Wavelets and subband coding*. Number LCAV-BOOK-1995-001. Prentice-hall, 1995.

[177] P. Viola and M. Jones. Rapid object detection using a boosted cascade of simple features. In *Computer Vision and Pattern Recognition, 2001. CVPR 2001. Proceedings of the 2001 IEEE Computer Society Conference on*, volume 1, pages I–511–I–518 vol.1, 2001.

[178] Virat public dataset. `http://viratdata.org/`. Accessed: 2014.

[179] N.S. Visen, N.S. Shashidhar, J. Paliwal, and D.S. Jayas. Identification and segmentation of occluding groups of grain kernels in a grain sample image. *Journal of agricultural engineering research*, 79:159–166, 2001.

[180] Yuji Wada, Daisuke Tsuzuki, Naoki Kobayashi, Fumiyo Hayakawa, and Kaoru Kohyama. Visual illusion in mass estimation of cut food. *APPETITE*, 49(1):183–190, JUL 2007.

[181] E.A. Wan and R. Van Der Merwe. The unscented kalman filter for nonlinear estimation. In *Adaptive Systems for Signal Processing, Communications, and Control Symposium 2000. AS-SPCC. The IEEE 2000*, pages 153–158, 2000.

[182] J.Y.A. Wang and E.H. Adelson. Layered representation for motion analysis. In *Computer Vision and Pattern Recognition, 1993. Proceedings CVPR '93., 1993 IEEE Computer Society Conference on*, pages 361–366, Jun 1993.

[183] J.Y.A. Wang and E.H. Adelson. Representing moving images with layers. *Image Processing, IEEE Transactions on*, 3(5):625–638, Sep 1994.

[184] Qing Wang, Feng Chen, Wenli Xu, and Ming-Hsuan Yang. Online discriminative object tracking with local sparse representation. In *Applications of Computer Vision (WACV), 2012 IEEE Workshop on*, pages 425–432, Jan 2012.

[185] Su Wang, Yewei Zhang, Peng Deng, and Fuqiang Zhou. Fast automatic white balancing method by color histogram stretching. In *Image and Signal Processing (CISP), 2011 4th International Congress on*, volume 2, pages 979–983, Oct 2011.

[186] R. Weiss, P. J. Stumbo, and A. Divakaran. Automatic food documentation and volume computation using digital imaging and electronic transmission. *J Am Diet Assoc*, 110(1):42–44, Jan 2010.

[187] O. Williams, A. Blake, and R. Cipolla. A sparse probabilistic learning algorithm for real-time tracking. In *Computer Vision, 2003. Proceedings. Ninth IEEE International Conference on*, pages 353–360 vol.1, Oct 2003.

[188] D. A. Williamson, H. R. Allen, P. D. Martin, A. Alfonso, B. Gerald, and A. Hunt. Digital photography: a new method for estimating food intake in cafeteria settings. *Eat Weight Disord*, 9(1):24–28, Mar 2004.

[189] D. A. Williamson, H. R. Allen, P. D. Martin, A. Alfonso, B. Gerald, and A. Hunt. Digital photography: A new method for estimating food intake in cafeteria settings. *Eat Weight Disord*, 9:24–8, 2004.

[190] D. A. Williamson, H. R. Allen, P. D. Martin, A. J. Alfonso, B. Gerald, and A. Hunt. Comparison of digital photography to weighed and visual estimation of portion sizes. *J Am Diet Assoc*, 103(9):1139–1145, Sep 2003.

[191] David H. Wolpert. The lack of a priori distinctions between learning algorithms. *Neural Comput.*, 8(7):1341–1390, October 1996.

[192] I. Woo, K. Otsmo, S. Kim, D. S. Ebert, E. J. Delp, and C. J. Boushey. Automatic portion estimation and visual refinement in mobile dietary assessment. *Proc SPIE*, 7533, Jan 2010.

[193] Di Wu and Da-Wen Sun. Colour measurements by computer vision for food quality control - A review. *TRENDS IN FOOD SCIENCE & TECHNOLOGY*, 29(1):5–20, JAN 2013.

[194] Xiang Xiang. A brief review on visual tracking methods. In *Intelligent Visual Surveillance (IVS), 2011 Third Chinese Conference on*, pages 41–44, Dec 2011.

[195] Yonghong Xie and Qiang Ji. A New Efficient Ellipse Detection Method. In *International Conference on Pattern Recognition*, volume 2, pages 957–960, 2002.

[196] Lei Yan, Cheol-Woo Park, Sang-Ryong Lee, and Choon-Young Lee. New separation algorithm for touching grain kernels based on contour segments and ellipse fitting. *JOURNAL OF ZHEJIANG UNIVERSITY-SCIENCE C-COMPUTERS & ELECTRONICS*, 12(1):54–61, JAN 2011.

[197] Jun yan Huo, Yi lin Chang, Jing Wang, and Xiao xia Wei. Robust automatic white balance algorithm using gray color points in images. *Consumer Electronics, IEEE Transactions on*, 52(2):541–546, May 2006.

[198] Shulin Yang, Mei Chen, D. Pomerleau, and R. Sukthankar. Food recognition using statistics of pairwise local features. In *Computer Vision and Pattern Recognition (CVPR), 2010 IEEE Conference on*, pages 2249–2256, June 2010.

[199] Alper Yilmaz, Omar Javed, and Mubarak Shah. Object tracking: A survey. *ACM Comput. Surv.*, 38(4), December 2006.

[200] Qian Yu, ThangBa Dinh, and Grard Medioni. Online tracking and reacquisition using co-trained generative and discriminative trackers. In David Forsyth, Philip Torr, and Andrew Zisserman, editors, *Computer Vision ECCV 2008*, volume 5303 of *Lecture Notes in Computer Science*, pages 678–691. Springer Berlin Heidelberg, 2008.

[201] B. Zeisl, C. Leistner, A. Saffari, and H. Bischof. On-line semi-supervised multiple-instance boosting. In *Computer Vision and Pattern Recognition (CVPR), 2010 IEEE Conference on*, pages 1879–1879, June 2010.

[202] Chaoxin Zheng, Da-Wen Sun, and Liyun Zheng. Recent developments and applications of image features for food quality evaluation and inspection - a review. *TRENDS IN FOOD SCIENCE & TECHNOLOGY*, 17(12):642–655, 2006.

[203] CX Zheng, DW Sun, and LY Zheng. Recent applications of image texture for evaluation of food qualities - a review. *TRENDS IN FOOD SCIENCE & TECHNOLOGY*, 17(3):113–128, 2006.

[204] Xiaowei Zhou, Can Yang, and Weichuan Yu. Moving object detection by detecting contiguous outliers in the low-rank representation. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 35(3):597–610, March 2013.

[205] F. Zhu, M. Bosch, C. J. Boushey, and E. J. Delp. AN IMAGE ANALYSIS SYSTEM FOR DIETARY ASSESSMENT AND EVALUATION. *Proc Int Conf Image Proc*, pages 1853–1856, 2010.

[206] F. Zhu, M. Bosch, N. Khanna, C. J. Boushey, and E. J. Delp. Multilevel Segmentation for Food Classification in Dietary Assessment. *Proc Int Symp Image Signal Process Anal*, pages 337–342, Sep 2011.

[207] F. Zhu, M. Bosch, I. Woo, S. Kim, C. J. Boushey, D. S. Ebert, and E. J. Delp. The Use of Mobile Devices in Aiding Dietary Assessment and Evaluation. *IEEE J Sel Top Signal Process*, 4(4):756–766, Aug 2010.

[208] F. Zhu, A. Mariappan, C. J. Boushey, D. Kerr, K. D. Lutes, D. S. Ebert, and E. J. Delp. Technology-Assisted Dietary Assessment. *Proc SPIE*, 6814:681411, Mar 2008.

[209] Jacek M. Zurada, Aleksander Malinowski, and Shiro Usui. Perturbation method for deleting redundant inputs of perceptron networks. *Neurocomputing*, 14(2):177 – 193, 1997.

Vita

Robert DiBiano received dual Bachelor of Science degrees in Electrical Engineering and Computer Science at Lamar University in Beaumont, Texas in May 2004. He continued on to receive his Masters of Engineering in May 2008. Being highly interested in the field of image processing, he decided to continue on for a doctorate at Louisiana State University. There he had an opportunity to be involved in many projects advancing his knowledge in this area. These included the work discussed in this dissertation as well as aerial terrain analysis from satellite imagery, a robotic optical dynamic positioning system for navigation, detection and analysis of events on tracked vehicles and humans, laser materials analysis, laser diode stress testing, and robotic automation of experiments. Upon graduation, Robert will be one of the founding members in a startup specializing in deep learning based simulation and process improvement.