SANTA CLARA UNIVERSITY
DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

Date: May 7, 2016

I HEREBY RECOMMEND THAT THE THESIS PREPARED UNDER MY SUPERVISION BY

Nate Matsunaga
Rick Sullivan

ENTITLED

Image Processing for the Extraction of Nutritional Information from Food Labels

BE ACCEPTED IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE DEGREE OF

BACHELOR OF SCIENCE IN COMPUTER SCIENCE & ENGINEERING

Thesis Advisor

Thesis Reader

Department Chair

Image Processing for the Extraction of Nutritional Information from Food Labels

by

Nate Matsunaga
Rick Sullivan

Submitted in partial fulfillment of the requirements
for the degree of
Bachelor of Science in Computer Science & Engineering
School of Engineering
Santa Clara University

Santa Clara, California
May 7, 2016

Image Processing for the Extraction of Nutritional Information from Food Labels

Nate Matsunaga
Rick Sullivan

Department of Computer Science & Engineering
Santa Clara University

May 7, 2016

ABSTRACT

Current techniques for tracking nutritional data require undesirable amounts of either time or manpower. People must choose between tediously recording and updating dietary information or depending on unreliable crowd-sourced or costly maintained databases. Our project looks to overcome these pitfalls by providing a programming interface for image analysis that will read and report the information present on a nutrition label directly.

Our solution involves a C++ library that combines image pre-processing, optical character recognition, and post-processing techniques to pull the relevant information from an image of a nutrition label. We apply an understanding of a nutrition label's content and data organization to approach the accuracy of traditional data-entry methods. Our system currently provides around 80% accuracy for most label images, and we will continue to work to improve our accuracy.

Table of Contents

List of Figures

Chapter 1

Introduction

1.1 Problem

Dietary tracking in today's world requires a constant updating of numerous nutritional metrics such as calorie and fat contents of foods being consumed. Traditionally, this process necessitated keeping a handwritten or electronic log of eating habits and performing tedious calculations to keep one's progress up to date. Recently, health and fitness applications have arisen that provide an automated means of tracking nutritional data, but many of these programs still require the user to input all of the necessary information. This manual data input requires tedious repetition on the part of the user. Other applications have emerged that provide alternative, but flawed, methods of nutritional data gathering.

1.2 Current Solutions

Applications have a few different ways of simplifying user input of nutritional information. One solution used by popular apps such as MyFitnessPal leverages crowd-sourced information. Users can search for their product by name to find relevant statistics, but crowd-sourced data may lack information, may have confusing duplicates, or be completely inaccurate. Other solutions, such as the effort by Open Product Data, revolve around constructing a database of known reliable nutritional data. Database-centric solutions require all data to be gathered in a single location, and changing data must be updated constantly. This approach requires a large amount of infrastructure, collaboration, and maintenance. If the data is not maintained or new products are not added to the central repository, data that a user is looking for may be absent or outdated.

1.3 Our Solution

We propose a computer vision solution to extract data directly from the nutrition label on the food item itself. Our solution will use computer vision and character recognition technologies to pull relevant nutritional data (calorie, fat, protein, carbohydrate, and other nutrient amounts) from an image of the USDA nutrition label on the food's packaging. Using data directly from the food package ensures that data is never absent, stale, or inaccurate. Data will not have to be maintained, as the information presented on the food's label is required by law to be accurate. Our tool can be leveraged by mobile or web applications to provide a simple and accurate means of tracking diet. These applications do not have to connect to a central database of information; furthermore, mobile applications would provide the user a simple interface to connect with the web application.

1.4 Motivation

We have both used nutrition-tracking applications, and have found data entry to be unnecessarily tedious for the user. We have an interest in the field of computer vision, so we wanted to see how effectively a computer-vision approach can address an existing problem. We decided on image data extraction as a project due to its inherent difficulty and potentially widespread applications. Domain-restricted computer vision is currently only used in a few specific cases, such as depositing a check by taking a photo with your phone, but we think it could be applied to numerous significant use cases, starting with nutritional data.

Chapter 2

Requirements

2.1 Overview

Requirements for the system we described in the introduction can be divided into two categories: functional and non-functional requirements. Functional requirements define what must be done by the system, while non-functional requirements describe the manner in which the functional requirements need to be achieved. Furthermore, each set is divided into three tiers of importance: a critical requirement is absolutely necessary, a recommended requirement is highly desirable, and a suggested requirement could possibly be done but is not essential. Lastly, the design constraints are also included. These restrict the system design by limiting the means at our disposal to create the final product.

2.2 Functional Requirements

Functional requirements are non-negotiable functionalities that the system must have to be considered completed. We have identified seven functional requirements.

2.2.1 Critical

1. The system will analyze an image of a USDA nutrition label, identify the macronutritional information located in the top portion of the label, and return the relevant data in an object or text file.

2. The system will provide a C++ library that defines the necessary code for easy integration with other programs.

3. The system will work for most common image formats including .jpg, .bmp, .png, and .tiff.

4. The system will provide configuration functionality for data logging and filtering.

2.2.2 Recommended

1. The system will run on modern versions of Mac, Windows, and Linux operating systems.

2.2.3 Suggested

1. The system will support additional compilers other than modern versions of gcc.

2.3 Non-functional Requirements

Non-functional requirements describe the manner in which the system achieves the required functionalities. These requirements are measured on a continuum, not in a black-and-white fashion. We have identified four non-functional requirements.

2.3.1 Critical

1. The system will complete processing in a reasonable amount of time on modern desktop hardware.
2. The system will be easy to use and configure.

2.3.2 Recommended

1. The source code will be well-organized, cohesive, and thoroughly documented.

2.3.3 Suggested

1. The system will be portable across mobile, desktop, and server environments.

2.4 Design Constraints

Design constraints limit the resources and tools with which the project may be developed by setting technological boundaries on the project design. We have identified one design constraint.

1. The system must work correctly on Ubuntu 14.04.

Chapter 3

Use Cases

3.1 Overview

Use cases describe a series of steps to achieve a goal in a system. We have restricted our project's scope to a very narrow focus, so we are only presenting two use cases. Because we are creating a software development tool, not an end-user application, the actor in each use case is a developer using our system in an application. Figure 3.1 shows the use cases we have identified.

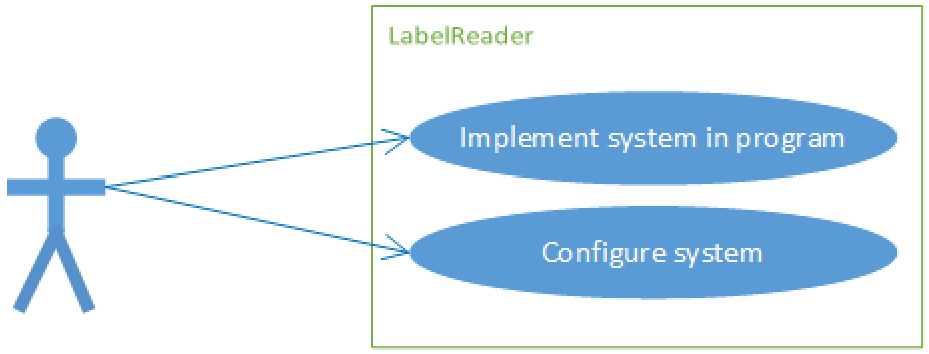


Figure 3.1: Use Cases for Nutrition Label Data Recognition.

3.2 List of Use Cases

3.2.1 Implement System in Program

Goal: Integrate software tool functionality in independent code.

Actor: User (developer using our library).

Precondition: User has OpenCV, Tesseract, and a C++ compiler installed.

Postcondition: The API has been successfully integrated, allowing the user to extract data from a label image.

Steps:

1. User includes our library header in their application or system.
2. User makes appropriate function call to run program.

Exceptions:

1. User gives invalid image input.

3.2.2 Configure System

Goal: System changes configuration for data logging and information filtering.

Actor: User (developer using our library).

Precondition: System is correctly integrated.

Postcondition: API uses new configuration for each subsequent image processed.

Steps:

1. User provides configuration parameters.
2. User calls our system code as they would normally.

Exceptions:

1. Defined configuration parameters are invalid.

Chapter 4

Architectural Design

4.1 Overview of Program Function

Our program's goal is to analyze nutrition label images and identify the textual information present on them. We are choosing to focus on the macro-nutritional data present in the top portion of most standard labels. Specifically, we look for...

- Calories
- Total Fat
- Saturated Fat
- Trans Fat
- Monounsaturated Fat
- Polyunsaturated Fat
- Cholesterol
- Sodium
- Potassium
- Total Carbohydrate
- Dietary Fiber
- Sugars
- Protein

To accomplish this goal we are using a combination of image pre-processing and post-processing techniques in conjunction with the optical character recognition engine Tesseract.

4.2 Dataflow Model

Our system follows a dataflow architectural model, which describes a system in which a defined type of data undergoes various transformations in sequence. In this case the data is an image file and the transformations are the pre-processing, optical character recognition, and post-processing stages it pipes through to determine the nutritional information located within it.

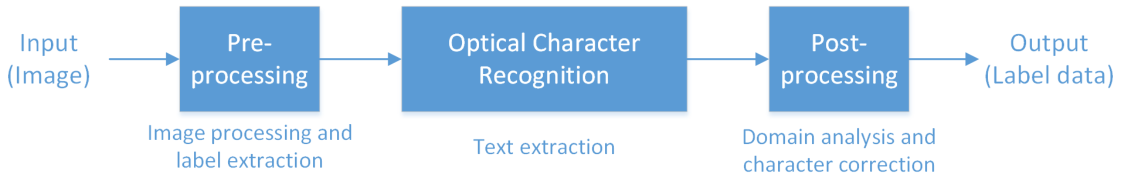
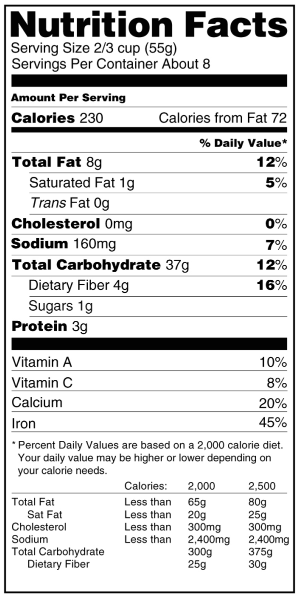

Input (Image)
Pre-processing
Image processing and label extraction
Optical Character Recognition
Text extraction
Post-processing
Domain analysis and character correction
Output (Label data)

Figure 4.1: Architectural Model.

4.3 Module 1: Pre-processing

The input image first passes through the pre-processing module. This stage is responsible for doctoring the image into a clean, organized format, in which the English text and numerical values present are easily identifiable. We designed the pre-processing stage to return a black-and-white, oriented image containing just the label portion of the input image. Figure 4.2 below shows an example of the ideal pre-processing output. The closer the image gets to that ideal form, the more effective the text identification and reading will be in the next step. Because our software focuses on reading the macro-nutritional information located in the upper portion of the label (Calories, Total Fat, Cholesterol, etc.), the pre-processing output may only contain that top part rather than the entire label.



Nutrition Facts
Serving Size 2/3 cup (55g)
Servings Per Container About 8

Amount Per Serving

Calories 230	Calories from Fat 72
	% Daily Value*
Total Fat 8g	**12**%
Saturated Fat 1g	**5**%
Trans Fat 0g	
Cholesterol 0mg	**0**%
Sodium 160mg	**7**%
Total Carbohydrate 37g	**12**%
Dietary Fiber 4g	**16**%
Sugars 1g	
Protein 3g	
Vitamin A	10%
Vitamin C	8%
Calcium	20%
Iron	45%

* Percent Daily Values are based on a 2,000 calorie diet. Your daily value may be higher or lower depending on your calorie needs.

	Calories:	2,000	2,500
Total Fat	Less than	65g	80g
Sat Fat	Less than	20g	25g
Cholesterol	Less than	300mg	300mg
Sodium	Less than	2,400mg	2,400mg
Total Carbohydrate		300g	375g
Dietary Fiber		25g	30g

Figure 4.2: Example of ideal output.

4.4 Module 2: Optical Character Recognition

The optical character recognition (OCR) module takes as its input the image generated by the pre-processing module. The OCR engine then attempts to identify the actual English words and numerical values contained in the image, and write them to a basic text file. To accomplish this task we implemented an open-source OCR engine called Tesseract, which is maintained by Google.

Tesseract works by matching all of the image data to English words and characters. It will attempt to find a match for everything that it sees, even if the contour does not relate to an alphanumeric character. For example, the horizontal black bars separating parts of the label are not letters or numbers, but Tesseract will match them to something anyway, usually a "-" character.

Furthermore, Tesseract analyzes the image data in chunks, meaning that it looks at how contours relate to one another to help determine what each contour represents. While this approach can increase word accuracy, any errant or invalid data (such as the horizontal lines) can negatively impact Tesseract's output. As such, we expect the Tesseract output to contain some inaccuracies in spelling as well as extraneous information such as lines of dashes.

4.5 Module 3: Post-processing

The post-processing module serves to correct inaccuracies in the Tesseract output, clean up the data, and report it in a usable format. As explained above we can expect that the Tesseract output may contain errors such as misspellings or misidentified characters, and that Tesseract might identify additional information that is not present on the actual label. The post-processing module analyzes the text file and searches for the important information (i.e. the macro-nutrient names and values) while ignoring any irrelevant data. Lastly, the module organizes the nutrient information into name-value pairs and outputs that data as a text file.

Chapter 5

Conceptual Model

The purpose of this section is to give a more detailed explanation of the techniques we used in our pre-processing and post-processing modules.

5.1 Pre-Processing Stage

As stated above in section 4.3, the objective of the pre-processing stage is to transform the input image into a simple, clear, and formatted output. To help accomplish this goal, we implemented some image manipulation tools from the OpenCV open-source library. Our pre-processing module can be broken into 5 distinct stages (listed below) that run sequentially. Each stage is accompanied by an example of a sample image that we processed with our program.

5.1.1 Graying and Blurring

The first pre-processing step is to convert the image from a multi-color space to grayscale. Most color images are represented using three values for each pixel, a red hue, a blue hue, and a green hue. Converting to grayscale transforms the pixel data to only depend on a single value ranging from white to black with shades of gray in between. Once the image has been grayed, we apply a subtle blur to help smooth out any outlying pixel values. The blur looks at each pixel and alters its value so its intensity more closely matches the relative intensity of the neighborhood of pixels around it.

Nutrition Facts
Serving Size 8 fl oz (240mL)
Servings Per Container 12

Amount Per Serving
Calories 150 Calories from Fat 20

	% Daily Value*
Total Fat 2.5g	4%
Saturated Fat 1.5g	8%
Trans Fat 0g	
Cholesterol 15mg	5%
Sodium 120mg	5%
Total Carbohydrate 26g	9%
Dietary Fiber 2g	8%
Sugars 23g	
Protein 8g	16%

Vitamin A 10% • Vitamin C 0%
Calcium 30% • Iron 0%
Vitamin D 25%

* Percent Daily Values are based on a 2,000 calorie diet. Your daily values may be higher or lower depending on your calorie needs:

Calories:		2,000	2,500
Total Fat	Less than	65g	80g

r your
Trade
d
nes.

Figure 5.1: Grayscale Image.

5.1.2 Contrasting and Filtering

The next step takes the image and applies a histogram equalization function to help increase the contrast among areas of different intensities. This process helps darken the nutrition label border to a black or near black value all along its contour to create a noticeable distinction against the lighter intensity background. Once the contrast between the label border and the background has been established, we use a bilateral filter to create more uniform pixel regions in the background of the image. The filter is particularly helpful in ensuring that the background area directly around the label border evens out to a lower intensity than the border itself.

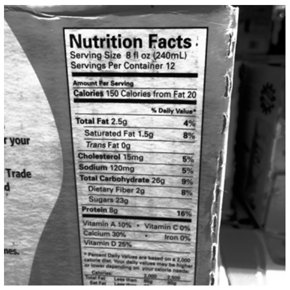


Figure 5.2: Contrasted Image.

5.1.3 Image Thresholding

Now that a stark contrast exists between the label border and the background we use a version of thresholding called Gaussian thresholding to set very dark pixels to black and lighter pixels to white. The Gaussian version takes into account the given pixel intensity relative to a neighborhood of pixels around it. This step produces a black and white image and isolates the black label border within a distinctive white background.

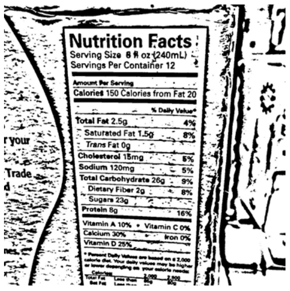


Figure 5.3: Thresholded Image.

5.1.4 Contour Identification

Once we generate the black and white image, we implement a contour identification process to find all shapes present in the image. This initial analysis records a large number of contours as each character, line, and any enclosed white space is usually identified as a contour. From this list, we filter all the contours based on their size and location in the image. We know that we are looking for a centrally located and rectangular contour that takes up a majority of the original image space. Using these qualifications we isolate the contour along the label border.

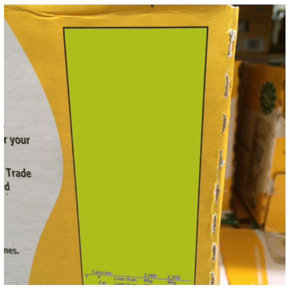
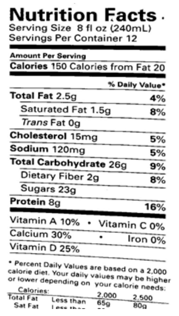


Figure 5.4: Contour Image.

5.1.5 Label Extraction

The final pre-processing step involves applying a perspective transform to map the identified label space on the original image to an oriented, rectangular space in a new image. The new image contains only the nutrition label, with the borders of the label on the original image becoming the outside border of the new rectangular image. The oriented label is then run through steps 2 and 3 again to clearly bring out the text information as black pixel data on a white background.



Nutrition Facts
Serving Size 8 fl oz (240mL)
Servings Per Container 12

Amount Per Serving	
Calories 150 Calories from Fat 20	
	% Daily Value*
Total Fat 2.5g	**4%**
Saturated Fat 1.5g	**8%**
Trans Fat 0g	
Cholesterol 15mg	**5%**
Sodium 120mg	**5%**
Total Carbohydrate 26g	**9%**
Dietary Fiber 2g	**8%**
Sugars 23g	
Protein 8g	**16%**

Vitamin A 10% • Vitamin C 0%
Calcium 30% • Iron 0%
Vitamin D 25%

* Percent Daily Values are based on a 2,000 calorie diet. Your daily values may be higher or lower depending on your calorie needs:

Calories:		2,000	2,500
Total Fat	Less than	65g	80g
Sat Fat			

Figure 5.5: Label Image.

Ideally the end result of the pre-processing stage is a vertically oriented, rectangular image of just the nutrition label with a white background and black text. While that goal may not be fully achievable, the closer we can get the image to that ideal, the more accurate the Tesseract output. Because we are not making any OCR modifications ourselves, successful pre-processing is critical in obtaining the most accurate result.

5.2 Post-Processing Stage

As mentioned above in section 4.5, the purpose of the post-processing stage is to identify and correct any errors present in the Tesseract output. This module depends heavily on the particular domain of the problem we are addressing. Because we are analyzing food labels, we can expect a restricted list of characters and words to appear on the label. Furthermore we can make use of the label structure itself to organize the OCR output. Using this domain knowledge greatly increases the effectiveness of the post-processing module and consequently the accuracy of our final results. Our four main post-processing stages are listed below.

5.2.1 Clean OCR Output

The first post-processing step is to read through the Tesseract output and decide which information is important and which information should be ignored. Tesseract tends to generate extraneous newlines and errant "-" and "*" characters that complicate the information we want. Specifically, this step analyzes the OCR output line by line and removes all lines that do not contain a close name match with one of the 13 macro-nutrients we are trying to identify. The important lines are then extracted from the file with their ordering maintained.

5.2.2 Construct Key/Value Pairs

Once we have identified the important lines, the next step is to create a name (key) and a value from each line. The name usually consists of the set of letters read before a number is found, and the value becomes the first sensible number read on the line. Sensible means that the number is located in an expected position. Some OCR reading errors identify numbers at the beginning of the line, but we know that the numerical value should be after the nutrient name. The key/value pairs are stored in a list of pair objects.

5.2.3 Keyword Matching

Next, we must determine the mapping from the keys to the macro-nutrient names that gives the best overall match for the whole list. To accomplish this we use a bipartite graph matching algorithm, which works by first determining the Levenshtein distance between each key and each nutrient name, and second finding the mapping that both matches each key to a nutrient and minimizes the total edit distance among all the matches. The Levenshtein (edit) distance is a metric that reflects how closely a given string of characters matches another given string.

The algorithm we use creates a best-fit mapping for the list of keys as a whole, rather than simply identifying the best match for each individual key. Furthermore, at this stage we account for the possibility of having multiple occurrences of the same nutrient name on the label. If the key chosen in the mapping has a line number outside the range of the expected number of nutrients then we know that we should re-run the matching algorithm, ignoring the line(s) that fall outside the determined range.

5.2.4 Construct Label Object

The post-processing module concludes by generating a simple text file containing the key names and their corresponding values. If a macro-nutrient was not present on a given label (e.g. several do not contain ”Potassium”) then the field for that nutrient gets a value of null.

5.3 Activity Diagrams

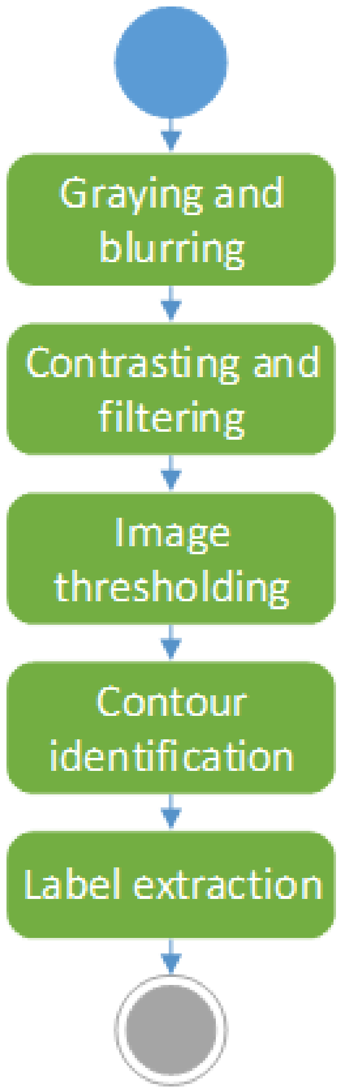


Figure 5.6: Activity Diagram for Pre-processing.

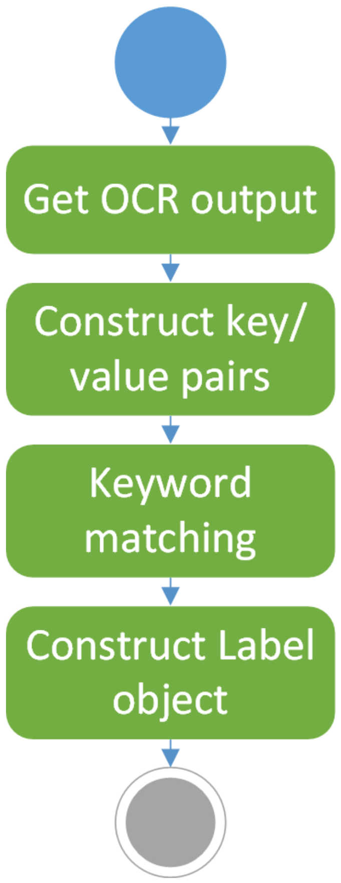


Figure 5.7: Activity Diagram for Post-processing.

Chapter 6

Design Rationale

6.1 Justification for Technologies

6.1.1 Primary Technologies Used

1. OpenCV
2. Tesseract
3. C++ programming language
4. Python programming language

6.1.2 OpenCV for Image Processing

The Open Source Computer Vision Library, better known as OpenCV, is an open source computer vision library. OpenCV provides common infrastructure for all manner of computer vision projects; it provides tools for object recognition, image processing, machine learning, and much more. OpenCV has some other advantages in that it is multiplatform, and provides APIs for five programming languages.

Support ecosystem

OpenCV is used and supported by companies of all sizes, including Google, Yahoo, Microsoft, and Intel. Because it is used so widely, it is very well documented, and we are unlikely to run into any problems that have not been previously encountered.

Unneccessary features

OpenCV does bring many features that we are unlikely to use in our small-scoped system. Features of OpenCV that are not needed could potentially bloat our system. However, we can circumvent this issue by building OpenCV with only the modules we need to use in our process. OpenCV has a comprehensive build system using CMAKE, which will let us customize our build to minimize bloat in our system.

Designing for change

OpenCV's huge number of modules could provide functionality for any design changes that occur over our project's lifetime. If we decide to branch out into different areas of the computer vision field to improve our library's accuracy, OpenCV will likely already provide some functionality we need.

OpenCV vs. custom code

An alternative to using OpenCV would be to create our own functions and libraries for image manipulation. However, this adds another likely point of failure to our project; we would be implementing algorithms that have been implemented and used many times in OpenCV. Using OpenCV gives us confidence in the correctness of the computer vision algorithms we use, as the stable features of OpenCV have been tested and used many times before.

6.1.3 Tesseract for OCR

Tesseract is an open source optical character recognition (OCR) library. It is now maintained by Google, and is considered on of the most accuracte open source OCR libraries.

Commercial OCR tools

We considered some commercial OCR tools as alternatives to Tesseract. The most popular and most accurate commercial OCR system is ABBYY's FineReader. FineReader and competing commercial OCR tools have license costs in the thousands of dollars for developer software development kits (SDKs). While FineReader is much more accurate than Tesseract out of the box, some studies have shown that finely tuning Tesseract can lead to performance near that or better than commercial products.

6.1.4 C++ Language

We plan to use C++ to implement the majority of our system. This design choice is highly influenced by two major factors:

1. Integration with chosen technologies
2. Team familiarity

Technology integration

Both OpenCV and Tesseract are written in C++, and natively provide C++ APIs. These APIs will allow us to directly integrate with these systems.

Team familiarity

We are both more familiar with C++ than any other programming language. Using this language will require minimal time devoted to understanding how to use new libraries.

6.1.5 Python Language

We will use Python for scripting to automate tests of our system. Python is much more flexible and user-friendly than shell scripting for automated tests. Python is aimed at increasing developer productivity by being understandable and easy to use, so it enables us to quickly automate running our system and any other included technologies. Python provides easy integration with C++ function calls as well.

6.2 Pre and Post Processing

Optical character recognition (OCR) accuracy can be greatly improved by pre-processing the image in certain ways. Studies have found that OCR accuracy is negatively influenced by factors such as image skew, low contrast, low lighting, and geometric distortion. Image pre-processing aims to eliminate these negative factors through intelligently altering the provided image.

After raw OCR output is gathered, we can apply post-processing to identify and fix OCR errors. If output is restricted by a certain lexicon, or list of words possible in an image, we can use this information to adjust erroneous output. Many OCR systems use the english language as a lexicon for correcting OCR, but we can use the much more limited set of expected nutrition label terms to improve accuracy.

6.3 Choice to Implement as API

We decided to implement our system as an application program interface (API) that can easily be included in applications of all types. We provide a C++ API through header files so that users can quickly include our system without having to install or build anything.

We considered creating a fully-fledged application for our system. However, many nutrition and fitness applications already exist, and our software would serve to complement those other systems rather than act as the basis for a multi-featured application. This approach allows us to focus on the functionality and accuracy of our system and more effectively explore the applications of computer vision to our particular problem.

Chapter 7

Testing and Results

7.1 Test Plan

We took two major approaches to testing the effectiveness of our implementation. The first approach focused on testing each of our three modules that make up our solution. We attempted to isolate each of these modules in our architecture and manually verify whether our approach at the time was effective. These tests occurred primarily during the early implementation phases. Our second testing approach analyzed end-to-end accuracy of our entire system. For obvious reasons, this testing begin once we had a fully-functional prototype completed, and continued throughout the rest of our implementation. For both types of testing, we constructed ground truths for each image in our data set, which contain the data we expect to see from a correctly working system. All accuracy testing was verified against these truths.

7.1.1 Module testing

Pre-processing

In order to isolate improvements in pre-processing, we pre-processed images in our data set and passed them through Tesseract OCR's default configuration. We manually compared this raw output (without post-processing) to our ground-truth files. Our project was heavily research based–we implemented around 10 to 15 different pre-processing approaches that had been used in various computer vision projects and applications. Testing continuously allowed us to verify whether or not a certain approach would be effective for our problem. Once our pre-processing step was producing relatively clear images, we used an edit distance calculation to determine how closely our output came to the expected ground truth. This allowed for some early automation of testing before our entire system was in a working state.

Optical Character Recognition

Similarly to pre-processing, isolated Tesseract in testing, feeding clear images to various configurations. Unfortunately, Tesseract proved very difficult to correctly configure to a restricted-domain problem, and only restricting the engine's word dictionary provided any realistic results.

7.1.2 Testing Post-processing

The ability of post-processing to improve our system's accuracy was highly dependent on the first two components delivering some level of accuracy, so post-processing testing was delayed until later in our project. We identified some common mistakes from our OCR module, so we created test cases to ensure that our post-processing approach accounted for them. For example, the 'g' for 'grams' after a number value in a label is often misidentified as the number '9', so we created tests to verify our handling of the issue.

7.2 End-to-end testing

After our system reached a functional state, we tested the system in its entirety by evaluating accuracy across a variety of test sets. Labels come in many different types, and we needed to verify how well our system handled each type individually.

Accuracy was evaluated by finding the percentage of nutritional amounts correctly identified across the label set.

$$Accuracy = \frac{\sum_{i=1}^{n} \# \ values\ correct\ in\ label_i}{\sum_{i=1}^{n} \# \ values\ expected\ in\ label_i}$$

We constructed and analyzed the following data sets.

7.2.1 Standard

Standard labels are images that have the label centered and vertical, with the label taking up the majority of the image. Labels are black and white and are on a two-dimensional surface. These test images should be the easiest to process.

7.2.2 Skewed

These test cases contain images of nutrition labels that are skewed (rotated) in either direction. This set also includes images taken from above or below a label, which results in perspective distortions. Our system should be able to handle angled perspectives of nutrition labels.

7.2.3 Lighting

Our lighting subset contains labels with uneven lighting, or shadows appearing across the image. These labels require our system to provide more complex intensity thresholding.

7.2.4 Curved

Curved labels refer to labels on the side of cylindrical objects. These labels test how well our system handles nonlinear geometric distortion.

7.2.5 Color

Different color schemes can have negative effects on OCR accuracy. This test set contains images with color schemes outside of the regular black and white. This also tests how well our system deals with low-contrast label images.

7.2.6 Horizontal

Certain food items have nutrition labels that have all information presented inline in a horizontal field. This data set challenges our system with more complex layouts.

7.3 Test Results

7.3.1 Module testing

Our module testing was primarily manually-verified, with some later applications of edit distance calculations. We do not have concrete test results for this primary testing stage.

7.3.2 End-to-end testing

Our results across our constructed data sets can be see in figure 7.3.2. Most of our data sets are processed with approximately 80% accuracy. Our approach sees a slight decrease in accuracy in our curved data set–our current pre-processing approach relies on linear transformations, which can result in distorted final images for curved labels, reducing our final accuracy. We had initially planned to address horizontal labels, which list values in a comma-separated list. Due to time constraints, however, this has not been fully implemented in our system, resulting in the 35% accuracy for that category.

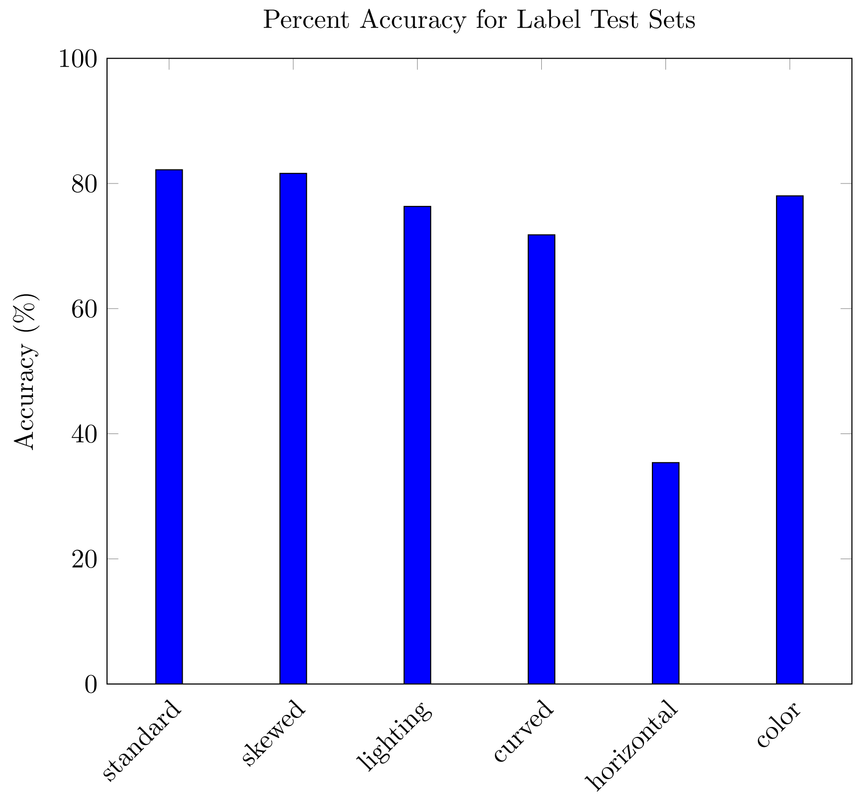


Figure 7.1: Accuracy across label test sets. Data set sizes are approximately 20 labels each.

We also noticed varying accuracy levels for each of the nutritional values we chose to identify on the labels. Figure 7.3.2 lists accuracy levels for each of these values in the order that they appear on labels. Most of the nutritional keywords have a consistent accuracy near or above 80%. However, keywords that appear toward the bottom of the label, such as sugars and protein, have much lower accuracies. This is due again to our method of geometric distortion correction–certain distortions become more exaggerated toward the bottom of an image, requiring further pre-processing.

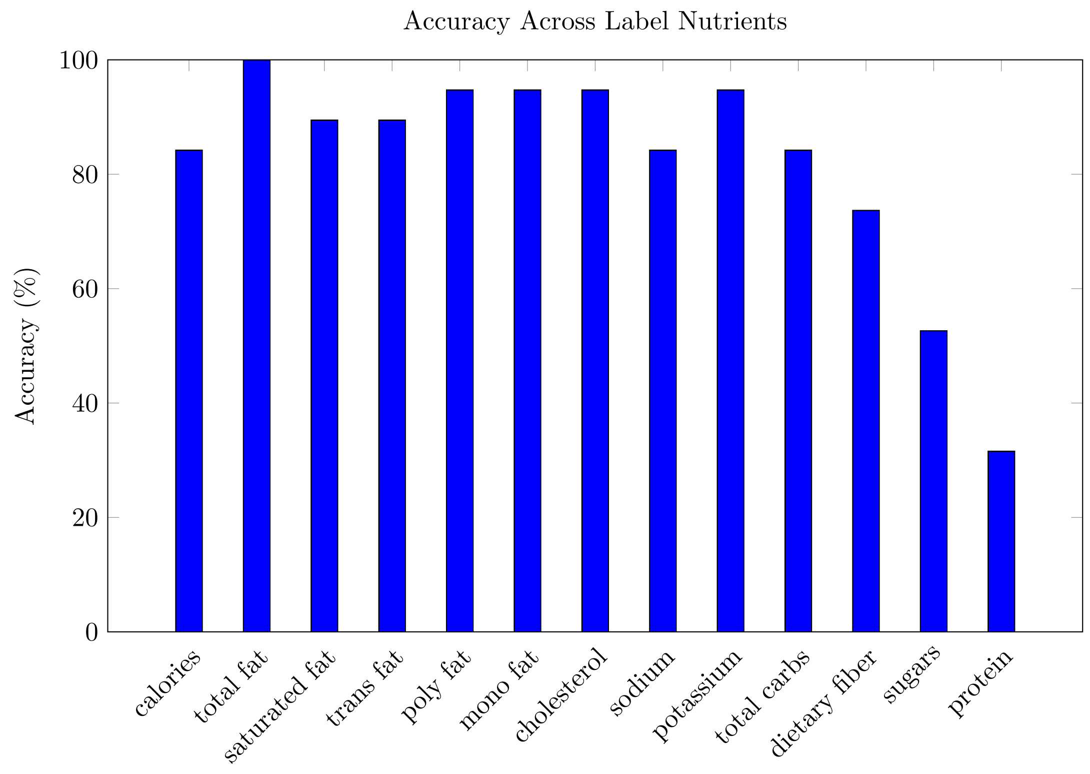


Figure 7.2: Accuracy across label nutrients for standard labels.

A histogram of the success levels for each of our labels can be found in figure 7.3.2. Each color represents one of our constructed data sets. Most of the labels we have in our test set result in over a 60% accuracy rate. However, there are still quite a few outlying cases that our approach does not adequately handle.

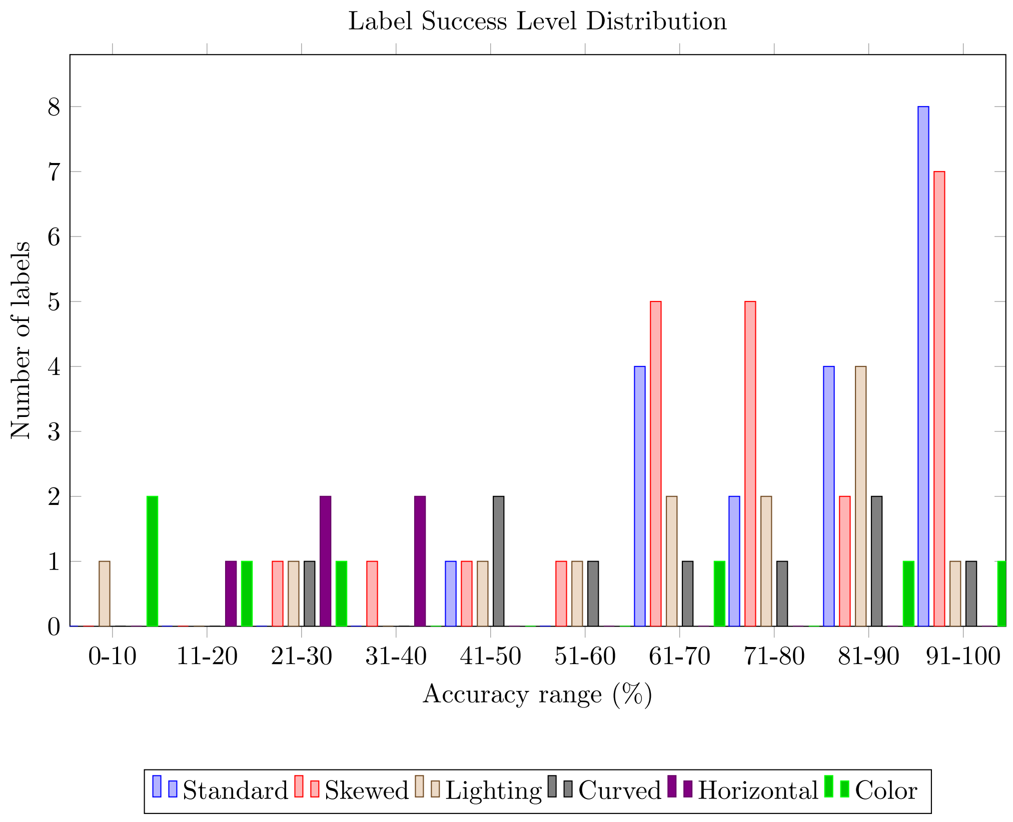
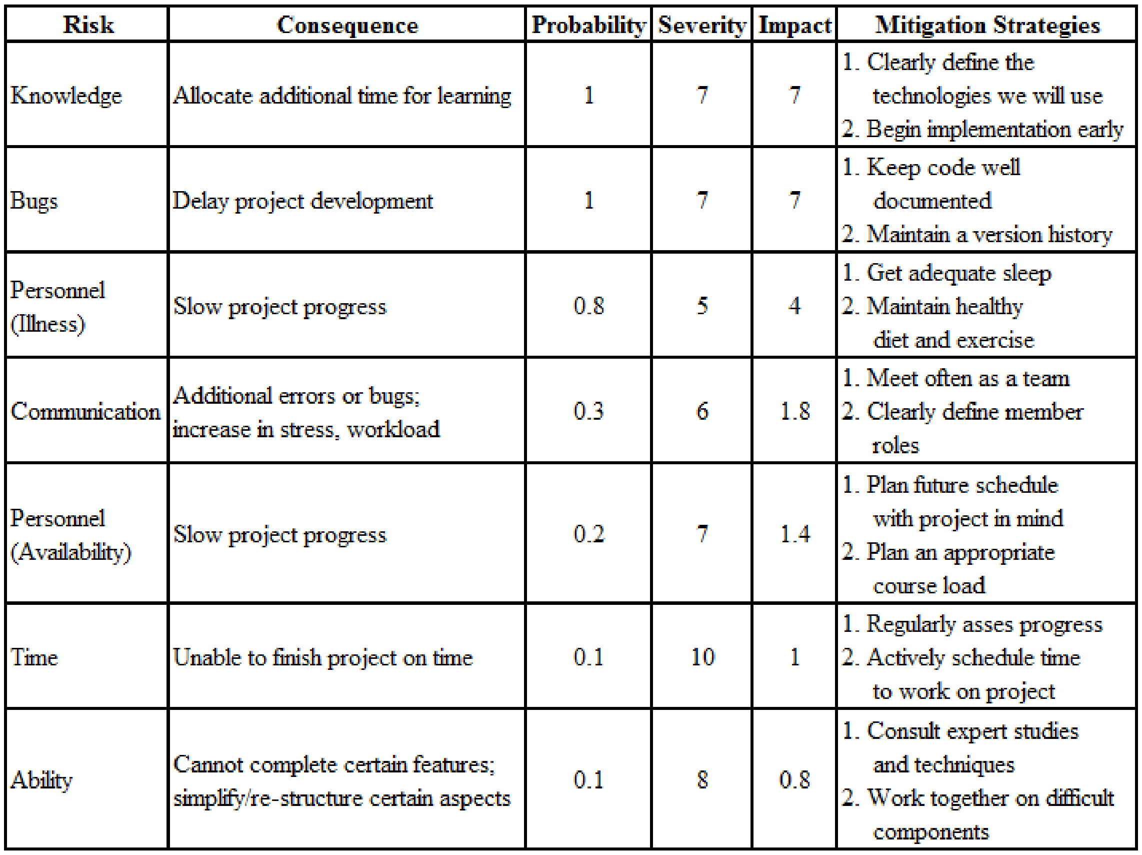


Figure 7.3: Label Success Level Distribution.

Chapter 8

Risk Analysis

This section outlines the risks associated with the development of this project. The probability of a risk occurring is assessed between 0 and 1. The severity of the risk is measured between 1 and 10. The risk's impact is the probability it will happen multiplied by its severity. The mitigation strategies are actions that lessen the impact of a risk.



Risk	Consequence	Probability	Severity	Impact	Mitigation Strategies
Knowledge	Allocate additional time for learning	1	7	7	1. Clearly define the technologies we will use 2. Begin implementation early
Bugs	Delay project development	1	7	7	1. Keep code well documented 2. Maintain a version history
Personnel (Illness)	Slow project progress	0.8	5	4	1. Get adequate sleep 2. Maintain healthy diet and exercise
Communication	Additional errors or bugs; increase in stress, workload	0.3	6	1.8	1. Meet often as a team 2. Clearly define member roles
Personnel (Availability)	Slow project progress	0.2	7	1.4	1. Plan future schedule with project in mind 2. Plan an appropriate course load
Time	Unable to finish project on time	0.1	10	1	1. Regularly asses progress 2. Actively schedule time to work on project
Ability	Cannot complete certain features; simplify/re-structure certain aspects	0.1	8	0.8	1. Consult expert studies and techniques 2. Work together on difficult components

Figure 8.1: Risk Analysis Chart.

Chapter 9

Conclusion

9.1 Project Summary and Evaluation

As stated at the beginning of this paper, our goal for our senior design project was to apply the field of computer vision to the existing problem of dietary tracking. We integrated the domain knowledge associated with the standard structure and form of a nutrition label to create an innovative solution to improve on the existing manual entry and database models. This project was very research heavy for a development project, as we had no prior experience working with computer vision applications. As such the final system still has some room for future accuracy improvement. We are pleased with the work we have done over the last nine months, and we think that we have set the stage for further advancements both with this project in particular and the increasing number of computer vision applications.

9.2 Future Progress

Although senior design has concluded, we plan to continue working on this project, testing out new implementations, and exploring more OCR options to see if we can obtain better results. The project is located in a gitHub repository, so anyone who wishes to review or experiment with what we have done is welcome to do so. We do not plan to make this a marketable item; even if further improvements are made we intend to keep it open source as it is now.

9.3 Ethical Considerations and Project Impact

The field of computer vision has widespread applications. We believe that advancements in this field could allow for near endless applications that would allow machines to automate tasks that previously required direct human involvement. As computer vision essentially seeks to teach machines how to see and glean understanding from images and video just as people do, there is a potential for great benefit, but also great harm. Our particular project has a positive focus that could not directly be used an unethical application, such as a program that reads credit card numbers from public video recordings. Additionally, we have not contributed to the underlying theory of computer vision techniques, so our project has not enabled the creation of any harmful products.

Primarily, we see our senior design project as an early effort to expand the applications of computer vision and combine existing techniques with the specific domains associated with various problems to improve on current solutions. Anything that requires systematic data entry, pattern recognition, object identification, and any other observation based task could theoretically be automated with the proper integration of computer vision techniques with the given problem domain. Our efforts give testament to, if nothing else, the real possibilities that exist in developing more efficient and automated solutions to these problems utilizing the field of computer vision.

Appendices

Appendix A

Install Guide

A.1 Package Manifest

The installation folder should contain the following files:

- categories.py
- contours.py
- end_to_end.py
- eng.label-patterns
- generate_json.py
- keywords.py
- label.py
- post_process.py
- requirements.txt
- text.py

A.2 Installation Instructions

Product installation is relatively straightforward. Once the install package has been downloaded, extract the files to the desired directory. In order to run properly the user must have OpenCV (3.0.0 alpha or higher), the Tesseract OCR, and Python (3.4 or higher) installed properly. Once everything is set up, simply call the end_to_end process located in end_to_end.py to run the software.

Appendix B

Bibliography

Bhattacharjee, Disha, et al. “A Novel Approach for Character Recognition”. *International Journal of Engineering Trends and Technology* 10.6 (2014): 271–275. Print.

This source deals with the goal of English word recognition using computer vision. While the source “Scene Text Recognition Using Structure-Guided Character Detection and Linguistic Knowledge” centers on detecting words within structured images, this source gives a description of identifying any and all printed words in an image for the purpose of language translation. As with any optical character recognition project accuracy is the paramount concern, and this article focuses on a new method of increasing OCR accuracy using image pre-processing or “cleaning up” the image to make characters more easily recognizable. By smoothing out lighting and shadow effects, perspective distortion, and confusing background colors OCR programs can more effectively identify English characters. This article is helpful for our project because we are developing our own method of preparing images for OCR reading, and this source provides some useful techniques we will want to consider. The source is peer-reviewed and published in a scholarly engineering journal. I found this source using the Santa Clara University library database.

Bieniecki, Wojciech, Szymon Grabowski, and Wojciech Rozenberg. “Image preprocessing for improving ocr accuracy”. *Perspective Technologies and Methods in MEMS Design, 2007. MEMSTECH 2007. International Conference on.* IEEE, 2007. 75–80. Web.

Bieniecki and his co-authors discuss methods for correcting geometric distortion of text pages. Correcting these distortions can drastically improve the accuracy of OCR systems. The essay explains methods for detecting and correcting both linear and non-linear distortions. These explanations can be extremely useful for our approach in correcting geometric distortion of nutrition labels in images. Nutrition labels exist on both flat and round objects, as well, so we can improve our design by considering both linear and non-linear correction.

Borovikov, Eugene. “A survey of modern optical character recognition techniques”. (2004). Web.

The purpose of this article is to provide a broad summary of current optical character recognition techniques. It focuses on techniques used to analyze both printed and hand-written words in images. The article begins by giving an overview of OCR engines and their general trend of development. It continues with discussing various OCR techniques for different image types and scenarios. Lastly it talks about pre-processing and post-processing strategies and how they apply to optical charater recognition. The article was written in 2004 so some of the information is not current regarding the present state and direction of OCR engines; however, it does provide useful backgroud information on OCR methods and overall strategies for computer vision analysis. We used this article to help plan the basic structure of our program and to guide our initial testing efforts to determine OCR accuracy. While the article was not published in a scholarly journal it was written by a research scientist who works in the field of computer vision and hosted in an academia database making its information trustworthy.

Chang, Loh Zhi and Steven Zhou ZhiYing. "Robust pre-processing techniques for OCR applications on mobile devices". *Proceedings of the 6th International Conference on Mobile Technology.* ACM Digital Library, 2009. Print.

This source discusses two specific techniques for accomplishing the goal of image pre-processing with the added consideration of implementation on a mobile platform. Pre-processing is especially important in mobile applications using computer vision because they have less resources to work with than larger, more powerful machines. Although mobile devices have developed rapidly in the last decade they remain at a vast disadvantage in memory and processing power when compared to desktop machines, two aspects necessary for detailed image analysis. The two techniques described in this article utilize different algorithms that employ object isolation and shape detection. These algorithmic techniques seek to improve the accuracy of optical character recognition programs while minimizing the demand on mobile hardware. Ideally our project would culminate in a mobile application which would provide the easiest medium for customers to use our label analysis program. Implementing on a mobile device however means that we will have to adjust some of our methods for image analysis due to the decreased capability when moving from a computer to a mobile device, and this source provides a helpful starting point for making that transition. This source is both peer-reviewed and published in a scholarly journal. I found this source when researching general pre-processing techniques.

"How we tuned Tesseract to perform as well as a commercial OCR package". (2014). Web.

This blog post discusses the efforts of a group of developers to increase the performance of Google's Tesseract OCR engine by focusing on processing and cleaning an image before Tesserat analyzes it. They identified several factors such as lighting and contrast which can negatively impact an OCR engine's ability to process an image. By implementing a image cleaning program to smooth contrasts and remove lighting and shadow effects, the group saw an impressive increase in the performance of Tesseract compared to high-end commercial products, suggesting that focusing on effective pre-processing rather than the OCR engine may be the more beneficial endeavor when trying to increase the accuracy of computer vision software. While this source is neither peer-reviewed nor published in an academic journal it provides useful information regarding the choice between using an open source OCR like Tesseract or a commercial product such as ABBYY FineReader. This report helped us decide that we wanted to work with the Tesseract engine rather than purchase an expensive license for a commerical product.

Kim, Daehyn and Hong Yu. "Figure text extraction in biomedical literature". *PloS one* 6.1 (2011). Web.

In this article, Kim and Yu analyze and test methods to most effectively extract text from biomedical figures. Biomedical figures often have important and helpful text contained within, but out-of-the-box optical character recognition (OCR) software is not equipped to handle these more complicated figures. Kim and Yu focus on three components to improve OCR performance: image preprocessing, character recognition, and text correction. Throughout each of these steps, the authors apply knowledge of biomedical figures to make their tool more capable of text extraction. For example, text in biomedical figures often features words truncated or hyphenated at unexpected points; the authors found that standard OCR tools were ineffective at accurately extracting text from biomedical figures because of these unique features. Kim and Yu found that text localization and image up-sampling were the most effective preprocessing approaches to improve OCR accuracy. Their research identified five main sources of OCR errors: high image complexity, thick stroke, low image contrast, small font size, and non-standard font types. Kim and Yu's objective shares many similarities with our goal of extracting and processing text from nutrition labels. Most importantly, the authors identify the most effective areas of improving out-of-the-box OCR, as well as the most problematic sources of OCR errors.

"Macronutrients: the Importance of Carbohydrate, Protein, and Fat". University of Illinois at Urbana-Champaign. 2014. Web.

This source is an informative report from the McKinley health center at the University of Illinois at Urbana-Champaign that discusses estimating the caloric content of a food from its macronutrients. It gives the number of calories found in fats, carbohydrates, proteins, and alcohol, the four main sources of calories in any food. The report goes on to describe the health benefits of fats, carbohydrates, and proteins. To help improve the accuracy of our software, we plan to check the number of calories read against the number of fats, carbohydrates, protein, and alcohol read to determine if the relationship makes sense, as the number of calories is roughly dependent on those four metrics. We used this brief report to verify this relationship so we could correctly check these values in our program. This source gives mostly common knowledge information, but it is pertinent data that we did not know before-hand. This source is neither peer-reviewed nor published in an academic journal, but it is supported by a nationally accredited university which gives it validity. I found this source through researching relationships among nutritional metrics found on a USDA food label.

Marcin, Helinski, Kmieciak Milosz, and Parkola Tomasz. "Report on the Comparison of Tesseract and ABBYY FineReader OCR Engines". Poznan Supercomputing and Networking Center, Poznan Supercomputing and Networking Center. 2012. Web.

As the title suggests this report details a comparison of two different optical character recognition programs, Google's Tesseract and ABBYY FineReader. The report focuses on comparing the accuracy of the two engines in analyzing scanned images of historical documents. Currently, Google's Tesseract is the best open-source OCR engine while ABBYY FineReader is the best commercial software. When using each as a standalone system the ABBYY engine vastly outperforms the Tesseract; however, licenses for the ABBYY software cost thousands of dollars while the Tesseract is completely free. This article describes efforts to train the Tesseract engine and improve its accuracy to analyze how close the performance could get to commercial software levels. In terms of the test cases (reading historical documents) the study found that with adequate preparation the Tesseract gave results equitable to those of the ABBYY FineReader.

We used this source to help us determine the OCR engine we wanted to work with for our project. While purchasing an ABBYY license was never realistic we wanted to gauge the effectiveness of the free Tesseract engine against commercial products. Along with additional research this source helped us decide that we could work with Google's Tesseract and obtain accurate results. The source is peer-reviewed but was published online only not through a scholarly journal. I found this source when researching the pros and cons of different OCR engines.

Ranjini, S. and Dr. M. Sundaresan. "Extraction and Recognition of Text From Digital English Comic Image Using Median Filter". *International Journal on Computer Science and Engineering* 5.4 (2013): 238–44. Web.

Ranjini and Sundaresan analyze and explain the problems with extracting text from images of comics and provide a solution for some types of comics. The authors focus on comics that contain a speech bubble. In order to recognize the text from within the speech bubble of a comic, Ranjini and Sundaresan apply a median filter to remove noise from the image and then use connected-component labeling to identify the location of a speech bubble within the image. They then assume that the speech bubble will be a certain size relative to the entire image, and they are able to isolate the speech bubble. Once the speech bubble is located, the actual character recognition is trivial. Ranjini and Sundaresan neglect to provide any justification for why their method is preferable over another, and do not provide any information about how accurate their approach actually is. One component of our project focuses on finding a nutrition label from within an image, and then extracting text from inside of that image area. Identification and isolation of a comic's speech bubbles is a similar task, although I am hesitant to believe that Ranjini and Sundaresan's method can be generalized for real-world images because of their lack of supporting evidence.

Rice, Stephen V. "Measuring the accuracy of page reading systems". Diss. University of Nevada, Las Vegas, 1996. Print.

In his dissertation, Rice analyzes methods for determining the accuracy of optical character recognition (OCR) systems. He thoroughly explains solutions that have been used as the computer vision develops. In order to measure character accuracy, Rice covers solutions to the string editing problem, as well as improved versions of classic algorithms. The dissertation also covers other forms of accuracy, including phrase accuracy and non-stopword accuracy. Rice's research and clear explanations give us a basis to determine how accurate our system is. This will enable us to measure the effects of changes to our system, and to determine where we can most effectively improve our system's accuracy.

Sawaki, Minako and Norihiro Hagita. "Text-line extraction and character recognition of document headlines with graphical designs using complementary similarity measure". *IEEE Transactions on Pattern Analysis and Machine Intelligence* 20.10 (1998): 1103–9. Web.

Sawaki and Hagita approach the problem of headline recognition from a very low-level. They explain an approach to identify text-lines and then apply the complementary similarity measure to analyze each individual character in the identified line. For their testing, the authors focus on recognition of Japanese headlines. Japanese characters are much more complex than Arabic characters, and are likely more difficult to distinguish between. This means that Sawaki and Hagita's approach to character identification can be directly applied to English or other languages, although it does not account for the likelihood of certain words occurring. For individual character recognition, however, Sawaki and Hagita provide a strong argument for their approach. Sawaki and Hagita tackle two problems that are directly applicable to our project: text-line identification and individual character recognition. They provide a large amount of detail, explanation, and justification for their methodology.

Shi, Cun-Zhao, et al. "Scene Text Recognition Using Structure-Guided Character Detection and Linguistic Knowledge". *IEEE Transactions on Circuit and Systems for Video Technology* 24.7 (2014): 1235–50. Web.

This scholarly article explores a new method of extracting scene text from real-world images. Scene text can be have different fonts, shadows, distortions, deformations, low resolutions, and occlusions; these attributes make scene text much more difficult to identify accurately. Shi and his co-authors give experimental results for each component of their character recognition system. They explain how each component was tweaked to improve performance of their final system. In the concluding sections, the authors also compare the performance of their solution against other commercially-available methods as well as alternate methods proposed in research. Their method generally outperforms previously implemented methods, but the authors are clear in showing the data sets and configurations where they are not the optimal solution. This article not only provides extremely detailed approaches to character and word recognition in real-world image, but it also emphasizes which approaches work best in which contexts. This research could be extremely valuable in directing our approach to character recognition in nutrition labels.

Tsai, Sam S., et al. "Combining image and text features: a hybrid approach to mobile book spine recognition". *Proceedings of the 19th ACM international conference on Multimedia.* ACM, 2011. Web.

Tsai and his co-authors approach the problem of identifying book spines by applying both character and object recognition. Book spines do not have many distinguishing features other than text, so the authors' solution uses text recognition to search a book spine text database, while book spine images are used to search a book spine image database. These results are intelligently combined to increase the accuracy of identifying a particular book. Their results show that this combined approach improves the text-based or image-based systems' 72% recall rate to approximately 91% recall. The authors' system does increase recognition latency from 0.22 and 0.57 seconds for text and image recognition, respectively, to a 1.24 second delay. However, the increase in system accuracy will often be worth the time tradeoff. Nutrition labels will always have text inside, and we could potentially merge character recognition with object recognition to more accurately identify the location of a label in an image.