Nutrition Facts

Final Report for ECE 1778.

Apper: Jamie Waese
Programmers: Sagun Bajracharya, Jacob Li & Guang Wei Yu

Word Count: 1996 words

Apper Statement: 357 words

Total Word Count: 2353 words

Introduction

Nutrition labels have been mandatory on all prepackaged foods in Canada since December 2007. These labels provide information on the amounts of 13 core nutrients and calories in an amount of food, along with a % Daily Value indicator to help people make informed decisions over food choices. This data is presented in the form of a standardized table, called the Nutrition Facts Table, which is text based and difficult to extract meaningful information from. Given that this information is important, it is unfortunate that the design specifications are so poorly suited for user comprehension.

The purpose of our app is to revisualize the data presented in the Nutrition Facts label in a more comprehensible manner. By scanning a nutrition label, users can automatically generate an intuitive and accessible chart of the data, along with a list of foods with equivalent nutritional values. The app also calculates personalized daily nutrition recommendations based on the user's sex, age, height, weight, and activity levels. Until label regulations are modified to reflect current thinking about data visualization, this app can help people make smarter nutritional choices.

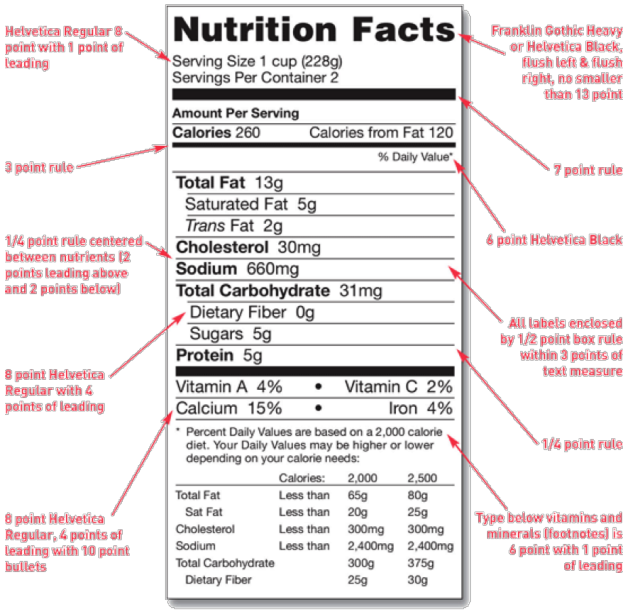

Figure 1. A sample nutrition facts label. Note how the data is presented entirely with text, requiring considerable cognitive processing to parse information. There is also a fair amount of "non-data ink" which further complicates visual analysis of the image.

Overall Design

General Design Notes:
The app opens directly to its main screen where the user can begin using it immediately. To maintain the familiar look and feel of the Nutrition Facts label, a similar black-text-on-white-background motif was used. We also matched the fonts and horizontal bars as page separators. For internal design consistency, all colors were chosen from this color palette:

Figure 2. Our color palette.

Data Visualization:
Our data visualization charts went through several rounds of design before we settled on the final layout. Here are two early incarnations:

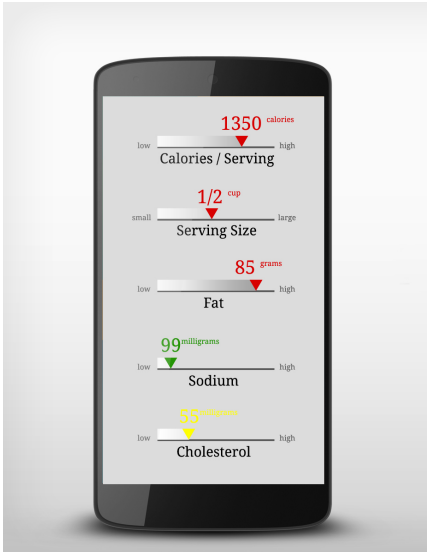

Figure 3. An early mockup, rejected because of the lack of hierarchical organization.

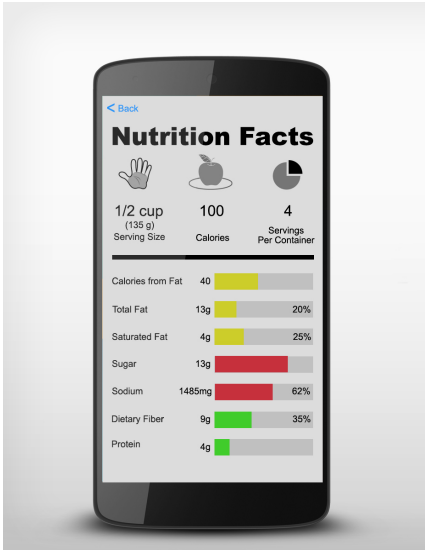

Figure 4. Another early mockup, rejected because of unnecessary infographics.

Our final design was influenced by the work of several prior designers, particularly Bradley Mu, who created this fine piece:

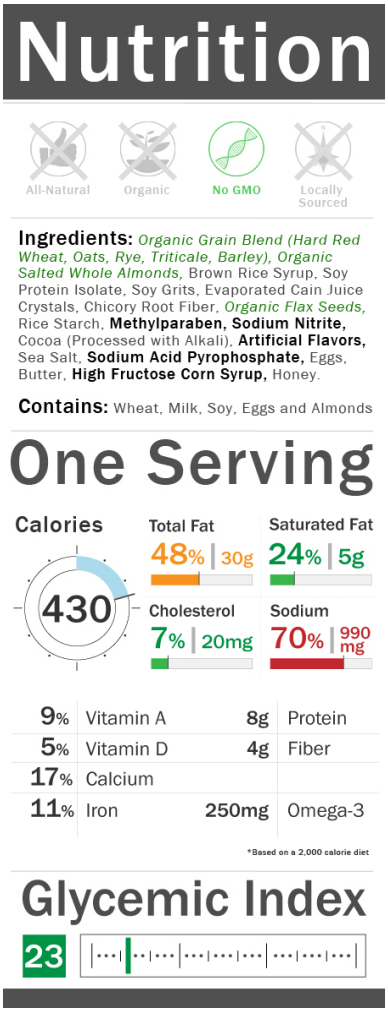

Figure 5. Bradley Mu's entry to Good Magazine's Redesign the Food Label contest. (http://www.bradleymu.com/newsite/nutrition-label/)

Our final design is presented here:

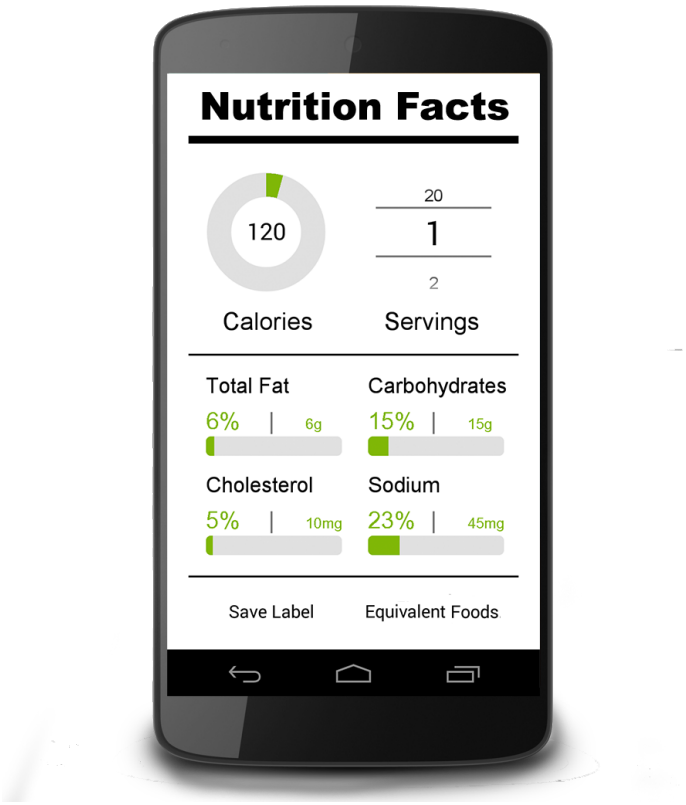

Figure 6. Our redesigned food label.

The minimalist aesthetic is consistent with current thinking in data visualization. Color-coded bar charts are easier to extract information from than text because they rely on features we preattentively recognize: size, placement, color, etc. For this prototype we chose to display Calories, Fat, Carbohydrates, Cholesterol and Sodium. A future version will allow users to select additional items they wish to track.

Block Diagram:

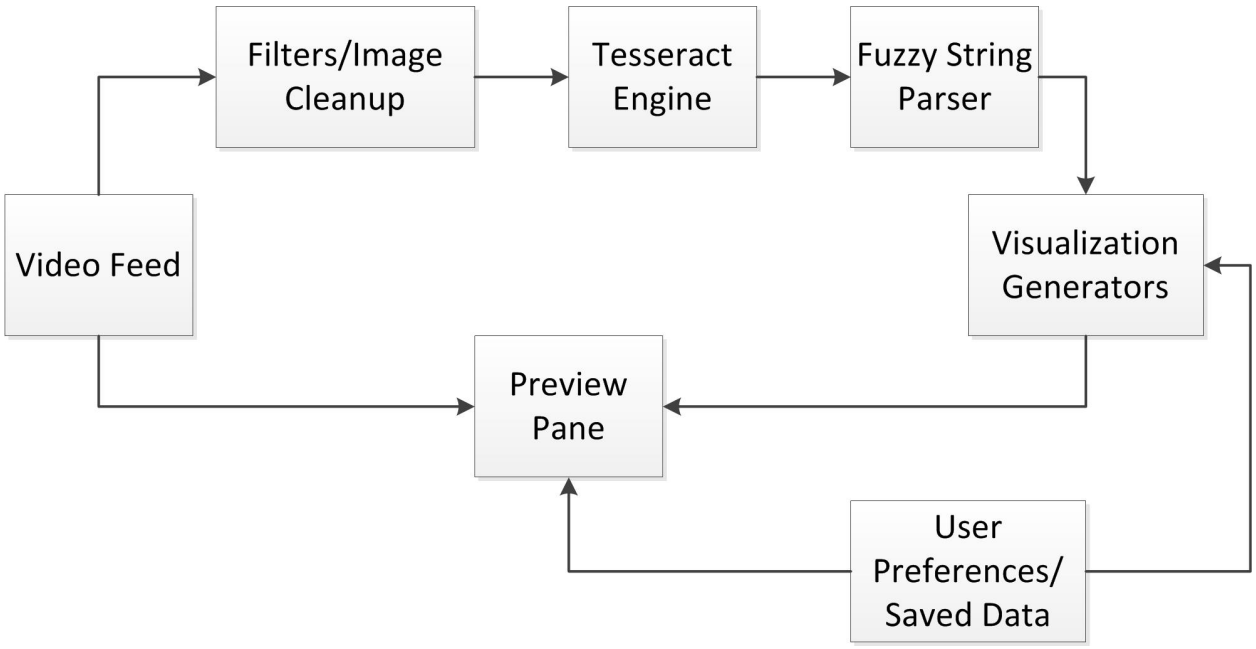

Figure 7. Block diagram of app.

Video Feed:
This block takes in continuous video data from the camera and sends it to the display screen. When the user clicks on the camera button, a picture is taken of the current preview, and gets passed onto the optical character recognition (OCR) engine.

Filters / Image Cleanup
This block pre-processes a bitmap grayscale image polled from preview video feed. The grayscale image is cropped and rotated to appropriate orientation, and passed through a binarization filter using Otsu's method. The output is 1-bit Leptonica Pix format binary image which the Tesseract (OCR) engine accepts.

Otsu's method is used to dynamically threshold the grayscale image to produce a binary image. The dynamic thresholding enable binarization of image under various lighting conditions. It is also ideal for nutrition fact type label as the foreground (the words) are highly contrasted from the background (usually white). We also add image smoothings to make OCR easier to perform. We use Leptonica image processing libraries to perform these filterings.

Tesseract (OCR) Engine

The filtered image is scanned by a port of the desktop Tesseract engine and a simple text file containing all of the characters that the engine recognized is output. We took a third party open source simple OCR program, pared it down, and used it as the basis for this block.

Fuzzy String Parser

The raw text file from the OCR engine is run through a fuzzy string parser which looks for specific substrings which correspond to nutritional fields (i.e., “Calories”, or “Sodium”). This is done by splitting every line in the input text file between text and numbers, then generating a Levenshtein distance between the input substring and a substring from the dictionary, and (if certain thresholds are met) finally choosing an interpretation of that input substring. The numbers are then taken as the value for the discovered field.

A filter to fix misidentified characters (ex. the OCR engine returned a “o 9” instead of a “0 g”) was also implemented in this block. The filter uses the context from the text before and after, as well as similar character shapes, to determine whether a character has been misidentified or not.

Visualization Generators

Two separate types of visualizations are generated from data returned by the fuzzy string parser. The first includes all of the charts and graphs and the second provides a set of pictures of comparable foods for each nutritional field. The first visualization page uses modified native progress bars to display fat, carbohydrates, sodium and cholesterol; a custom backport of the native number picker to display the serving size picker; and a custom version of the third party library Progress Wheel. The second visualization page has a fixed number of images stored for each nutritional field - each image represents an approximate ‘tier’ in its field (either low, medium, or high). It also uses the custom number picker.

Preview Pane:

This block is responsible for displaying anything the user can see while using the app. This includes the initial video feed as well as the revisualization that is returned by our app.

User Preferences:

This block allows users to personalize the daily recommended nutritional values that get passed to the graph generator.

The calorie counter is based on the Mufflin equation for Resting Metabolic Rates (http://www.caloriesperhour.com/tutorial_BMR.php):

- Men: (10 x weight in kg) + (6.25 x height in cm) - (5 x age in years) + 5
- Women: (10 x weight in kg) + (6.25 x height in cm) - (5 x age in years) - 161

This is multiplied by a factor determined by McArdle et al (1996) according to the user's activity level, as interpolated from the following table:

Activity Factor	**Category**	**Definition**
1.2	Sedentary	Little or no exercise and desk job
1.55	Moderately Active	Moderate exercise or sports 3-5 days a week
1.9	Extremely Active	Hard daily exercise or sports and physical job

This block also includes the saving and loading feature, which allows the user to save a label for future perusal or comparison.

Statement of Functionality

Below are screenshots detailing the various parts of our app:

Figure 8. Camera Screen.

Camera Screen

This is the main screen of the app. From here, the user can scan a Nutrition Facts label and access the Personal Settings or Load Image screens. User help tips are accessed by pressing the “Click me for help” line.

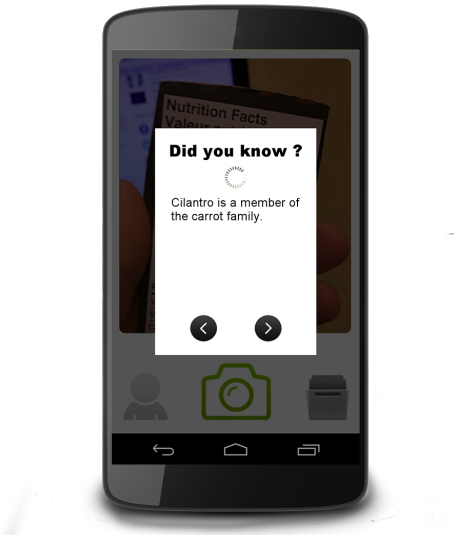

Figure 9. Processing Screen.

Processing Screen

Upon capturing an image, a series of random nutrition facts provides engagement while the image gets processed in the background.

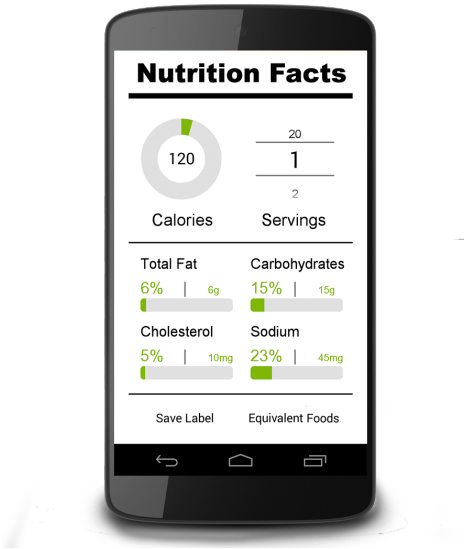

Figure 10. Visualization Screen

Visualization Screen

This screen uses status bars to display the captured data. A scroll wheel allows users to update the displayed values according to their desired number of servings. The buttons at the bottom of the page link to the Save Label function, and the list of Equivalent Foods.

Figure 11. Equivalent Foods Screen.

Equivalent Foods Screen

This page displays photographs of foods that have equivalent nutritional values for the field selected by the scroll wheel. (Ex. 1 serving of cookies has 6g fat, which 1 ice cream cone also has)

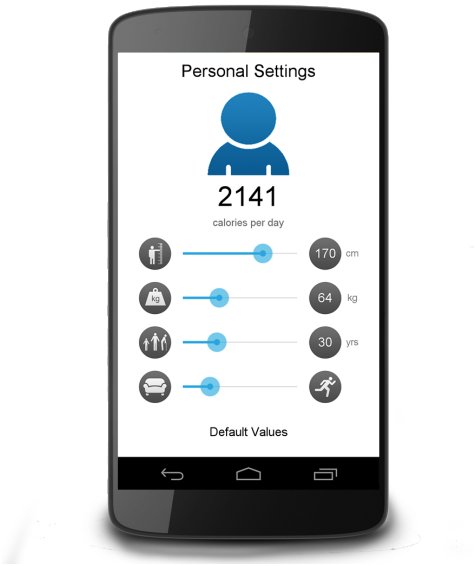

Figure 12. Personal Settings Screen.

Personal Settings Screen

Users can adjust their daily recommended nutrition values by customizing the sliders for height, weight, age, and activity level. Tapping the person icon changes gender.

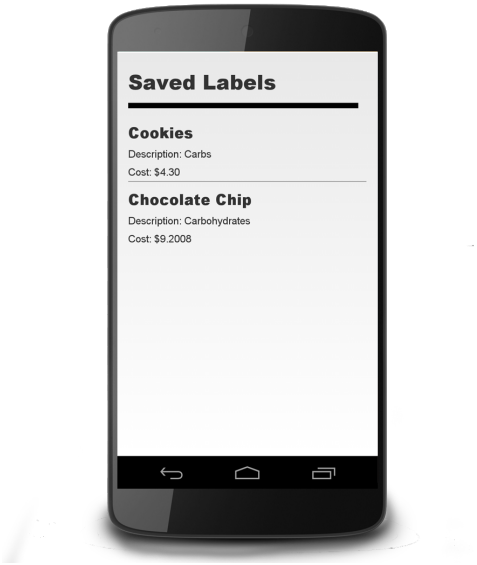

Figure 13. Load Label Screen.

Load Label Screen

From the main screen, touching the Open File icon brings the user to this page. Here, the user may view any previously saved Nutrition Label.

Considerations for Next Time

We have learned that OCR is a computationally expensive and challenging task. It is particularly difficult when nutrition labels are warped, the surface is laminated, or the lighting is not favourable.

Currently, all computations are done locally on the mobile device. This results in an overall processing time of 12 ~ 15s. If this app were to be remade, we would consider doing all the processing on a server rather than on the mobile device. The OCR engine, Tesseract, performs better on servers than on mobile devices. Even accounting for upload / download time, this should provide a performance boost.

Contributions by Group Members

Jamie:
I designed the look and feel of the app -- first using broad strokes with Moqups.com, then again with fine detail using Photoshop. I also prepared all the graphic elements and did a fair amount of research to support the daily recommended nutrition values and food equivalency comparisons. Finally, I contributed to the production of the app by providing usability and design notes at every stage of development and spearheading the preparation of this final report.

Sagun:
I implemented the main page of the app that displays a camera preview inside a bounding box to guide users in taking pictures of nutrition labels. I also set up the back end computations required to crop and downscale the image to a small portion of the entire image, reducing the overall computational power required for OCR. In addition to the main page, I also implemented the settings page that is capable of remembering the user's settings throughout various instances of the app.

Jacob:
I created the fuzzy string matcher and misidentification fixer which takes in raw junk text data from the OCR of the image and interprets words that would be on a nutritional label within that data. In addition, I also created the main visualizations page, which showed all of the nutritional fields in a graphical format. Finally, I implemented the saving and loading features which allowed users to save labels and view them later.

Guang:

I implemented the filters for image pre-processing. For pre-processing, I used Otsu's method to apply dynamic thresholding to a grayscale image to obtain a binary image, and a smoothing filter to close out edges in the letters. Instead of passing grayscale bitmaps, we pass the binary image which is only 1-bit of information per pixel to the OCR engine. Additionally, I implemented the visualization page which gives pictures of equivalent foods for each nutritional field. Finally, I implemented a randomized factoid popup feature during OCR computation to keep users occupied.

Apper Context

As a student of data visualization, I am interested in developing tools and techniques that help people grasp the significance of raw data through visual means. My expertise is on the design side, and my studies focus on applying what we know from a growing body of literature about data visualization to bioinformatic research tools that were not originally built with that in mind. Since my skills are multidisciplinary by nature, my future projects will span many fields, not just biology. Working with nutrition data has given me an opportunity to explore data visualization paradigms beyond the world of biology, and thus broaden the scope of my expertise. With that in mind, I am grateful to have had the opportunity to work on this project as part of my studies.

This app influences my research field in that it provides an example of how the rigorous application of current thinking about data visualization can be applied to commonplace subjects (such as the Nutrition Facts label) for non-educated users. There is an informal divide in the data visualization community between visualization apps intended for academia vs. visualization apps intended for the public. Apps intended for the public often over-decorate their data with unnecessary shading, 3D depth, outlines, and pictographs in order to make the data appear more "interesting". Apps intended for academia begin with the premise that the data is already interesting, and they focus on stripping away anything that might interfere with capturing the information they are designed to convey. This app demonstrates the positive effect of applying an academic design approach to an app made for the general public.

Given that I don't have a background in computer science, this experience has taught me several valuable lessons about about software engineering. I have already applied the notions of agile vs. cascade planning, wireframe mockups, and block charting to my dissertation project. While most of my studies are focused on building web applications, I am intrigued by the possibilities of applying multiple sensors and mobile computing to my future data visualization projects. Thank you for the opportunity to create this app as part of my graduate studies.

Future Work

It currently takes a long time to scan, process and display the data. A majority of this time is spent doing text identification. This is because Tesseract scans the image pixel by pixel to identify pixel clusters that potentially resemble letters. To speed this process up we convert the input image to grayscale and then threshold it so we are left with a binary representation of the image (where black pixels are 1s and white pixels are 0s). This helps find clusters of pixels quicker (since neighbouring pixels can be considered connected if they share the same value) and thus reduces the overall processing time of our app by about 10s.

However, the total processing time is still 12 ~ 15s. In the future, to reduce this time we would like to recognize entire words rather than individual letters. Because there are a limited number of words that can appear on nutrition labels, we expect this to reduce the overall processing time by another 4 ~ 6s since clusters of pixels representing words are much easier to identify than small clusters of pixels representing single letters.

Referenced Libraries and Projects:

Tesseract - https://github.com/rmtheis/tess-two
Sample OCR Application- https://github.com/rmtheis/android-ocr
ProgressWheel - https://github.com/Todd-Davies/ProgressWheel
NumberPicker - https://github.com/SimonVT/android-numberpicker

Source Code Release:

We would like to keep our source code unreleased for now, as we're hoping to do more work on it for a proper release into the Play Store.
Everything else (video, paper) can be posted online.

Referenced Nutrition Algorithm:

McArdle,William, Frank Katch, and Victor Katch. 1996. Exercise Physiology 4th edition Baltimore: Williams and Watkins