FussyFood: An Application to Navigate Food Allergies & Dietary Restrictions

Sameep Bagadia
Department of Computer Science
Stanford University
Stanford, California 94305
sameepb@stanford.edu

Rohit Mundra
Department of Electrical Engineering
Stanford University
Stanford, California 94305
rohitm92@stanford.edu

***Abstract*—This report addresses the design of an Android application with which the user can set dietary preferences, capture ingredient label images, and get feedback on the ingredients' alignment with his/her dietary preferences. We discuss the work flow of the application as well as the image processing routines used in this application to improve OCR accuracy. We also present a method to find the area of interest using F-score maximization allowing the automatic detection of the ingredient list bounding box. Finally, we evaluate how well this method works at minimizing false positive and false negative ingredient words.**

I. Introduction

In this project, we design an Android application which allows the user to make informed decisions about an edible product using an image of its ingredient list. For instance, it is estimated that 43–72 million people suffer from peanut allergies. Approximately 11 million people in America follow a vegetarian diet. Such instances demonstrate that large populations in the world need to make informed decisions regarding their diet due to a variety of restrictions. We make an application in which a user can take an image of the ingredients and receive a quick feedback informing them whether the product aligns with their dietary needs. Such an application is even more critical to users when they are unfamiliar with the ingredients listed – for instance, a person with dietary restrictions traveling to another country where the lingua franca is not one he/she is familiar with might find themselves in dire need for such an application.

II. Background

Optical Character Recognition engines have been shown to have a a very wide range of accuracy for a given image depending on the preprocessing done[1]. The diversity in results is rooted in the fact that the image given to an OCR engine can be rotated, warped, noisy and blurry among other distortions[2]. Thus, in our application it is important to be able to overcome many commonly found distortions in an automated manner before giving to the OCR engine.

In [1], the authors discuss many ways to preprocess images to improve OCR accuracy however many of these rely on image-specific processing. Many such techniques would require the user to themselves tune the parameters to counter the distortions. For instance, the authors describe an approach to recognize words in scanned textbooks that have noticeable distortions towards the spine; here, they hope for the user to be able to draw a spline along the text thereby demarcating the distortion. This would otherwise be harder to estimate efficiently and accurately.

In [2], the authors describe recognition of characters in natural images that are otherwise handled poorly by traditional OCR engines. The general approach is to extract features from the images such as shape contexts, SIFT descriptors, and Geometric Blur descriptors. They show such methods to work well with Nearest Neighbor methods as well as Support Vector Machines. We found that this approach would be more feasible if our application sent the image to a central server where the image processing was done rather than doing it on the mobile system itself.

While these studies gave us insight on how to tackle our problem, the problem definitions were different and thus, we relied more on empirically determining reliable strategies.

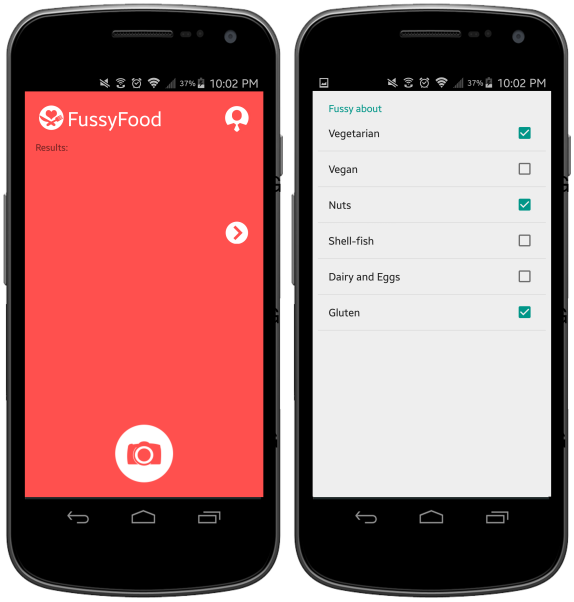
III. Application Usage

In this section, we discuss the various steps involved in our application. The process flow is shown in Figure 1.

Fig. 1: Process Flow

The user begins with selecting his dietary restrictions. In our application, clicking the user preferences button on the top right corner of the main activity screen opens the food preferences pane. This is shown in Figures 2a & 2b.

Once these selections are made, the user can click the camera button and the application will take him to camera screen where the user can click a photo of the ingredients. Once an image has been captured, the application will preprocess the image clicked before sending it to OCR engine. The image preprocessing steps are described in detail in the next section. From the text obtained from OCR, we select the list of ingredients found and query the database to find out whether each ingredient aligns with the user's dietary restrictions. The user is notified about each ingredient that is found to be non-compliant.



(a) Start Up Screen (b) Preferences Pane

Fig. 2: Application display on an Android phone

IV. Image Preprocessing

A major goal of this project is to improve the ingredient recognition accuracy by supplying the OCR engine with a cleaner image than the raw captured image. In this section we discuss the various preprocessing steps applied to the raw image before sending it to the OCR engine. We consider the image shown in Figure 3 as a running example to show the intermediate results after each step is applied on the image.

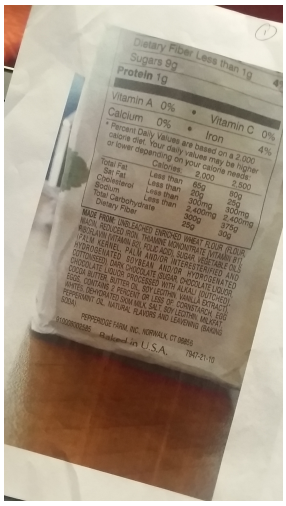


Fig. 3: Original image

A. Locally Adaptive Thresholding

We first convert the original RGB image to grayscale. This allows us to do thresholding efficiently based on a single value per pixel. In order to avoid incorrect thresholding due to lighting variations across a captured image, we apply locally adaptive thresholding where we divide the image into small windowed patches and apply Otsu's method in the blocks where the variance is found to be high (i.e. regions where text is likely to be present). Regions where variance is low are considered to be background. We found empirically that the window width and height of 96 pixels is appropriate for a 2.4 megapixel image (2048×1152). Of course, if the image dimensions are larger, we can linearly scale up the window width. Using this technique, we get the image as shown in Figure 4. Here we see that the text is successfully disambiguated from the background.

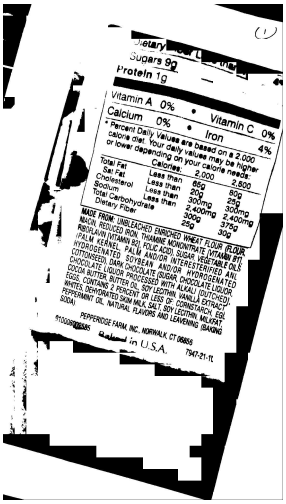


Fig. 4: Locally Adaptive Thresholding

B. Probabilistic Hough Transform for Auto-rotation

We found out that OCR performs very poorly if the image given to it is rotated. Therefore we auto-rotate the image to correct its rotation. We apply Probabilistic Hough transform to detect line segments in the image. This variant turns out to be computationally cheaper than the classical Hough transform and can be computed on a standard phone. For the Hough transform, we use distance and angle resolutions of 1 pixel and 1 degree respectively. Line segments shorter than one third of the image width are rejected since we expect that the image will be taken at a reasonable distance so that each line will take more than one third of the total width of the image. A maximum of 20 pixels of gap is allowed between points to be detected as part of the same line segment. After applying these parameters, only those lines are returned that have more than 100 votes. From these line segments, we filter out those lines which have an absolute angle of more than 45 degrees. Again, this assumes that the image will not be captured at extreme angles considering how most consumers naturally tend to keep images horizontally-axis aligned with only small rotation skews. We then take the median of the angles of line segments thus obtained as the skew angle of the image. Using the median allows to avoid extreme rotations that could potentially lead to errors.

Once we have the rotation skew angle, we rotate the image by that angle by applying the corresponding affine

transformation on the original image matrix and thus correct the image skew. The auto-rotated image is shown in Figure 5.

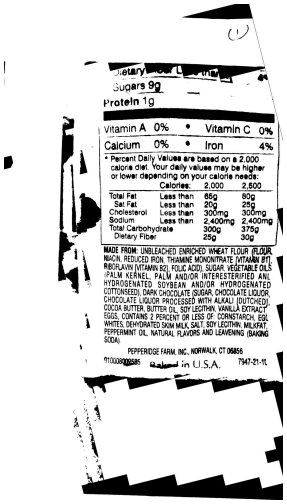


Fig. 5: Auto-rotation

C. Background Removal

As we see in the Figure 5, there is a lot of noise in the image. Particularly, there are regions corresponding to edges of the product and other markings that do not correspond to any text. In order to remove this type of noise, we first apply Canny edge detection method to detect edges in the image. This leads to high frequency of edges detected in regions where there is text. The image obtained is shown in Figure 6. We see that text edges, product edges, and all other markings now have a pixel wide Canny edge.

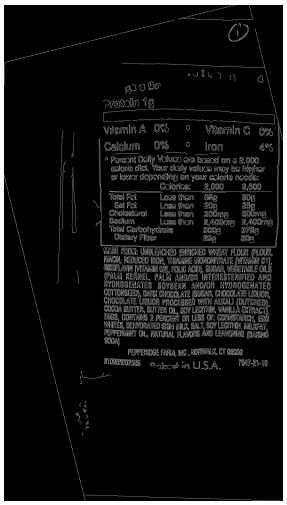


Fig. 6: Canny edge detection

We now need a way to keep only Canny edges that are close to other Canny edges. Thus, we apply an averaging box filter so that regions with many Canny edges will have a high average while those with low number of edges will get averaged with black pixels in the neighborhood. The rationale is that text regions will have a lot of white pixels and Canny edges nearby whereas non-text region will lack this property. We then threshold the resulting image after a global histogram equalization to remove the noisy regions. The resulting image is shown in Figure 7.

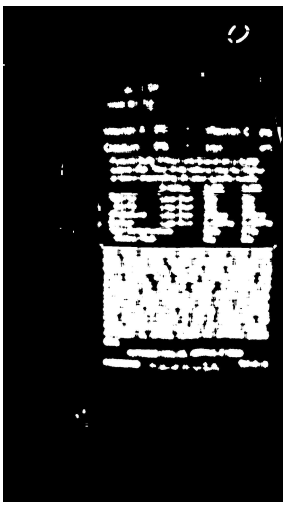


Fig. 7: Noisy Region Removal

D. Bounding Box Detection

At this point, we want to use image constructed so far to automatically identify the region that contains the ingredient list. Not all the text on the image is relevant to us since much of it contains other nutritional information and production details. We are only interested in text containing ingredients. So our aim is to detect a bounding box which contains the relevant textual region. The reason for finding this bounding box is two-fold:

1) The larger the image sent to the OCR engine, the slower the detection process with be; considering how OCR is a computationally expensive task, we wish to minimize the the input to solely textual information.
2) In many products we found text near the ingredients list signifying the absence of certain ingredients (such as "does not contain eggs" or "produced in a gluten free facility"). See Figure 8 as an example. If we perform a unigram database look up for words in such phrases, the database will incorrectly indicate presence of eggs or gluten.

Thus, it is important for our application to identify the ingredients list efficiently and accurately. An alternative approach would be to ignore bounding box identification and parse/process the natural language. As a design choice, we chose to handle this using image processing.

In order to find the bounding box, we realized that we need to keep dilating the image in Figure 7 till we find a central connected component that contains as much text in as little area as possible. The idea behind this is that words in an ingredient list are packed close together within a paragraphic form. Thus, we design an optimization problem around the image in Figure 7 where we want to maximize the fraction of white pixels in the central component while minimizing the

Fig. 8: Label indicates lack of wheat and gluten

area consumed by the rectangle containing that component. A quick inspection will make us realize that these are contentious goals. For instance, maximizing the number of white pixels in the central component would suggest that we should dilate the image so much that the entire image's white pixels are contained within it. This would result in a very large bounding box. But we also want to minimize this quantity. Conversely, the smallest bounding box would have no area and thus no white pixels in it. Thus, we create an F_β score optimization problem of the form:

$$\text{maximize} \quad (1+\beta^2)\frac{\text{precision} \cdot \text{recall}}{\beta^2 \,\text{precision} + \text{recall}}$$

Where precision is defined as the fraction of the image region not contained in the bounding box and recall is defined as the fraction of white pixels contained in the bounding box. β is the parameter which we can vary to get Pareto-optimal solutions to this optimization problem – for instance, a β value of 1 would lead to equal importance being placed on precision and recall while a lower value would emphasize precision over recall. For our application, we determined a value of $\beta = 0.5$ to be most accurate at finding the correct bounding box.

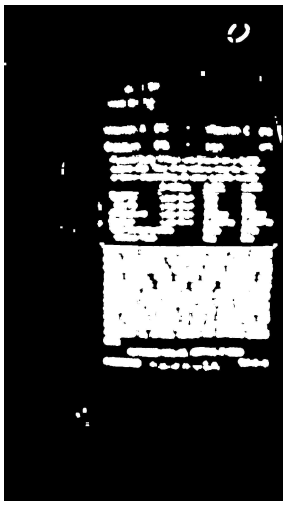


Fig. 9: F-score maximization

Thus, we iteratively dilate the image till we find the maximal $F_{0.5}$ score and then find the smallest rectangle encapsulating this connected component. We see in Figure 9 that the ingredient region is connected into a single component when our optimization problem is solved. Lastly, we find the minimum rectangle that contains this component and this problem can be solved trivially by observing the minimum and maximum x and y coordinates of the white pixels of the central component.

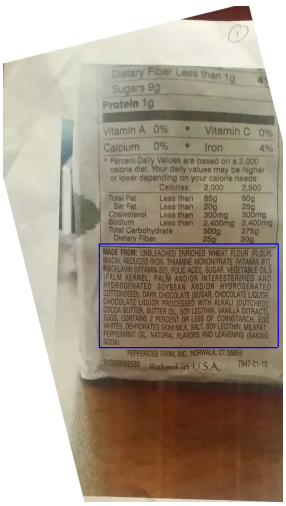


Fig. 10: Bounding Box

Overlaying this bounding box on the original image, the user can see the identified region of ingredients (Figure 10).

V. Optical Character Recognition & Database Lookup

Using the bounds we identified, we crop the locally adaptive thresholded image to get the final image to be fed to the Tesseract OCR engine [3] (Figure 11). Tesseract was found to perform very well in our application and its open-source library was available for Android. Given such an input image, the engine returns to us a text string that we post-process.

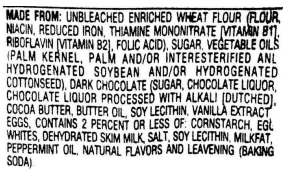


Fig. 11: Cropping

We split the string on all non-alphanumeric characters to get all the words in the ingredient list. We then query each word against a database and compare the user's preferences against the retrieved properties to check for alignment.

VI. Error Analysis & Results

In order to evaluate how well our system works in correctly identifying ingredient words and avoiding non-ingredient words, we designed three intuitive evaluation metrics:

1) Metric 1: Fraction of words detected to be as ingredients that were in fact ingredients.

TABLE I: Error Analysis

Metric	Scenario 1	Scenario 2	Scenario 3
Metric 1	0	0.760	0.742
Metric 2	1	0.240	0.258
Metric 3	∞	2.079	0.0

2) Metric 2: Fraction of words not detected to be as ingredients that were in fact ingredients.
3) Metric 3: Fraction of words detected to be as ingredients that were not ingredients.

The first two metrics assess and ensure that listed ingredients on the label are not missed since such cases could lead to false safety alerts to the user potentially leading to health and emotional damage. The third metric ensures that non-ingredient words are not detected; thus a low value would minimize false positives that would result from labels such as that shown in Figure 8.

We measure these metrics for three scenarios:

1) Scenario 1: Grayscale captured images fed to the OCR engine
2) Scenario 2: Auto-rotated images fed to the OCR engine
3) Scenario 3: Auto-rotated and auto-cropped images fed to the OCR engine

These scenarios illustrate how our system changes the values of different metrics across 10 sample images we tested on. Table I displays the results.

We see that giving a raw image to the OCR engine is insufficient to get desirable results. After auto-rotation, the OCR engine performs substantially better but without the appropriate cropping the system detects multiple non-ingredient words to be ingredients. The bounding box approach performs nearly as well at capturing the true ingredients but it outperforms at avoiding non-ingredient words.

VII. Future Work

Using the described system and image processing techniques, we were able to successfully design an assistive application for persons with dietary restrictions and/or food allergies. We found that many ingredient detection errors were the result of one or two characters being incorrectly recognized by the OCR engine. Thus, future work would be the integration of a spell-checker which can intelligently recognize words that are very close to ingredient words and handle them accordingly.

Furthermore, the application can be improved by also sharing a health report of the food item by analyzing the ingredients. This will increase the usability of the product and capture markets that care about general health but do not have any dietary restrictions.

We also hope to add more image processing routines to handle more types of warps that are commonly found, such as those on cylindrical objects. Perspective warp can also be integrated in the system. These will increase the general robustness of the system. [1]. [4]. [2].

References

[1] W. Bieniecki, S. Grabowski, and W. Rozenberg, "Image preprocessing for improving ocr accuracy," in *Perspective Technologies and Methods in MEMS Design, 2007. MEMSTECH 2007. International Conference on*, pp. 75–80, IEEE, 2007.

[2] T. E. de Campos, B. R. Babu, and M. Varma, "Character recognition in natural images," in *Proceedings of the International Conference on Computer Vision Theory and Applications, Lisbon, Portugal*, February 2009.

[3] R. Smith, "An overview of the tesseract ocr engine," in *Proceedings of the Ninth International Conference on Document Analysis and Recognition - Volume 02*, ICDAR '07, (Washington, DC, USA), pp. 629–633, IEEE Computer Society, 2007.

[4] R. Casey and E. Lecolinet, "A survey of methods and strategies in character segmentation," *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, vol. 18, pp. 690–706, Jul 1996.