Grocery Shopping Assistant Using OpenCV

Article · April 2017

DOI: 10.21276/ijre

4 authors, including:

Divya Dsouza
NMAM Institute of Technology
7 PUBLICATIONS **26** CITATIONS

SEE PROFILE

International Journal of Research and Engineering
ISSN: 2348-7860 (O) | 2348-7852 (P) | Vol. 04 No. 03 | March 2017 | PP. 77-81
http://digital.ijre.org/index.php/int_j_res_eng/article/view/261

Grocery Shopping Assistant Using OpenCV

Author(s): Deekshith R. Shetty, Avinash D., Abhiram Uppoor, Divya Jennifer Dsouza
Affiliation: Dept. of Computer Science Engineering, NMAMIT, Nitte, Karnataka, India

Abstract*— In this paper we present an android mobile application that allows user to keep track of food products and grocery items bought during each grocery shopping along with its nutrient information. This application allows user to get nutrient information of products and grocery by just taking a photo. Product matching is performed using SURF feature detection followed by FLANN feature matching. We extract the table from the nutrient fact table image using concepts of erosion, dilation and contour detection. Classifying the grocery is done using Object Categorization through the concepts of Bag of Words (BOW) and SVM machine learning. This application includes three main subsystems: client (Android), server (Node.js) and image processing (OpenCV).

***Keywords*— OpenCV, SURF Feature Detection, FLANN Feature Matching, Erosion, Dilation, Contour detection, Object Categorization, Bag of Words, SVM Machine Learning, Android**

I. INTRODUCTION

According to sources from WHO around the world, 170 million children are underweight while 20 million children's suffer from overweight. Poor nutrition has caused nearly half (45%) of deaths in children under five each year. Moreover at least 300 million adults are clinically obese. Undernutrition as the name indicates is caused when people intake foods that do not have enough nutrients to keep them healthy whereas over-nutrition is the opposite caused due to increased consumption of energy-dense nutrient poor foods. Both of these causes health risks causing chronic diseases like heart disease, delayed physical and mental development etc. This clearly indicates that nutrition must be given greater attention in discussions about health, whether in rich societies or the developing world. Eating the right food in terms of quantity and quality is more important. Therefore, food recognition is emerging as an important research topic in object recognition because of the demand for better dietary assessment tools to combat under-nutrition and over-nutrition.

Various researches have been carried out in the field of nutritional management using computer vision. Researchers from Carnegie Mellon University have presented a paper on recognizing the foods from videos of people eating in restaurants which is recorded from a web camera [1]. Based on the results, the average calorie and nutrients consumed by people are estimated. Maruyama et al. [2] presented a method to classify food images by updating the model of Bayesian network incrementally after investigating a "food log" system. Yang et al. [3] proposed a food recognition by calculating pairwise statistics between local features computed over a soft pixel level segmentation of the image into eight ingredient types.

The proposed system that we demonstrate in this paper aims at developing a mobile and server vision-based nutrition management system for smartphone users using the concepts of image processing. This system makes use of many of the concepts of image processing such as table extraction, SURF feature detection and FLANN matching and object categorization using Bag of Words and SVM machine learning.

From client's or user's viewpoint, the application offers these two main features. First, it allows the user to store a library of his/her often purchased grocery items. During item addition the user takes a photo of the product followed by its barcode and "Nutrients Facts" label which is present in the package. The nutrient fact table image is sent to an OCR web service which extracts the nutrient information as a string. This string is then matched using a string matching program to obtain individual nutrient's information (Calorie, carbohydrates etc.). This data will be stored in a cloud database for future access. Next time the user purchases the same item he/she can find calorie and nutrition info by just taking the photo of the product. The app will recognize each existing item and list the corresponding calories and macro nutrient information to help the user plan a healthier lifestyle. The second feature allows the user to get nutrient information of grocery items (non products) like tomato, potato etc. by taking a photo of the item. In the following sections, we will explain in detail the implementation of the application with a main focus on the image processing algorithms employed.

II. SYSTEM ARCHITECTURE AND USER INTERFACE

A. System Architecture

The application makes use of the smartphone camera capabilities to capture the image of the grocery item or the product image and its nutrition fact table (NFT). After uploading the image, we use the image processing and cloud technology to process, analyse and store the useful information obtained. A client-server architecture is best suited for this type of application. We have used an Android smartphone as a client, and Node.js for the server. The image processing is done using OpenCV. Parse is the cloud database provider.

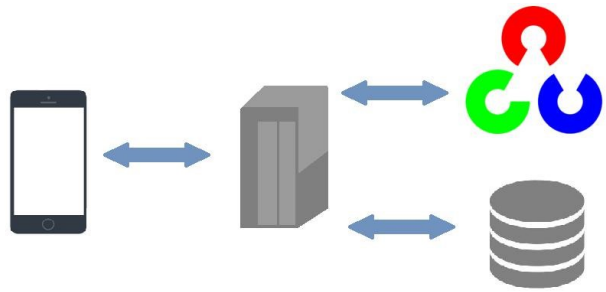

Fig. 1 System setup of the grocery shopping assist application

The cloud database acts as a server providing storage and retrieval services to the client. (Fig 1). Images taken by the phone is sent to the server running server script, as a byte stream. On the server side we make use of the popular OpenCV library to perform various operations on the image. The image uploads folder can be accessed by both server to store input images and by image processing server to process the image. After the information is extracted successfully we store the same in the cloud database.

B. User Interface

After the user opens the app and logs in to his account, the user interface for scanning contains four buttons as shown in the figure below (Fig. 2).

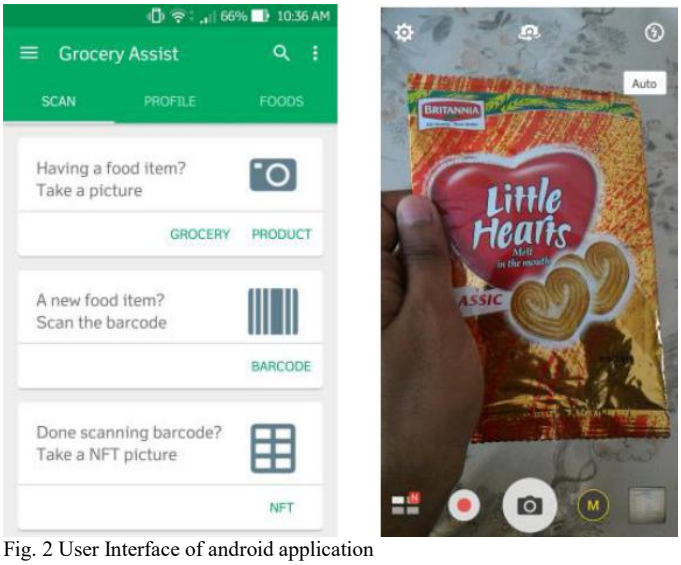

Fig. 2 User Interface of android application

1) Product button: This button allows user to take a photo of a product item and upload to the server.
2) Barcode button: If the photo of the uploaded product is new to user library, he needs to scan the barcode of the product item. Barcode is unique to each product and acts like a primary key in the cloud database.
3) NFT button: After scanning the barcode, user take the NFT photo using this button and uploads to the server.
4) Grocery button: This button allows user to take a photo of a grocery item and upload to the server.

III. FEATURE MATCHING PRODUCT IMAGES

The user can scan a food product by taking a clear photo of it and uploading to the server. The image is uploaded to the server and saved in "/uploads/user_products/userId/" folder which contains all the product images taken by the user "userId". The node.js server calls the Feature Matching OpenCV program to find the similar image in the library. SURF feature detection is performed on the input image and the samples in user library to extract prominent features and their descriptor. We then perform a loop through all the sample images to find the best match using FLANN matcher as shown in the FFBD diagram below (Fig. 3). The program outputs a message as not found if similar image isn't found in user library else it outputs the similar sample image name. Here, in this section we assume that the image of the food product was already uploaded by the user previously and is present in the user library.

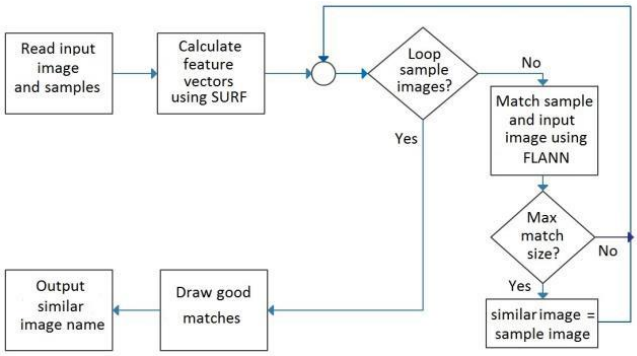

Fig. 3 Functional Flow Block Diagram for Feature matching sample images

A. SURF Feature Extraction

Speeded Up Robust Features (SURF) is a local feature detector and descriptor that can be used for tasks such as object recognition or classification. It is partly inspired by the scale-invariant feature transform (SIFT) descriptor. The standard version of SURF is several times faster than SIFT and is more robust against different image transformations than SIFT. Object detection using SURF is scale and rotation invariant which makes it very powerful. The algorithm has three main parts: interest point detection; local neighbourhood description; and matching. Square-shaped filters are used as an approximation of Gaussian smoothing. The descriptor provides a unique and robust description of an image feature by describing the intensity distribution of the pixels within the neighbourhood of the point of interest [4]. The features or point of interest detected using SURF on a food product is shown in the Fig. 4.a. By comparing the descriptors obtained from different images, matching pairs can be found.

B. FLANN Feature Matching

This matching is optimised by using Fast Approximate Nearest Neighbour Search (FLANN) based matcher. It contains a collection of algorithms optimized for fast nearest neighbour search in large datasets and for high dimensional features [5]. It works faster than Brute Force Matcher for large datasets. We find two nearest matches using knn match algorithm (k=2). It finds the 2 best matches for each descriptor in query with the descriptors in sample images as shown in Fig.4.b.

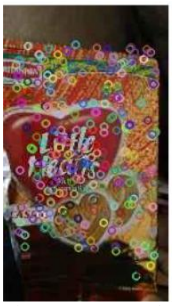
Fig. 4.a Features detected


Fig. 4.b Descriptors matched

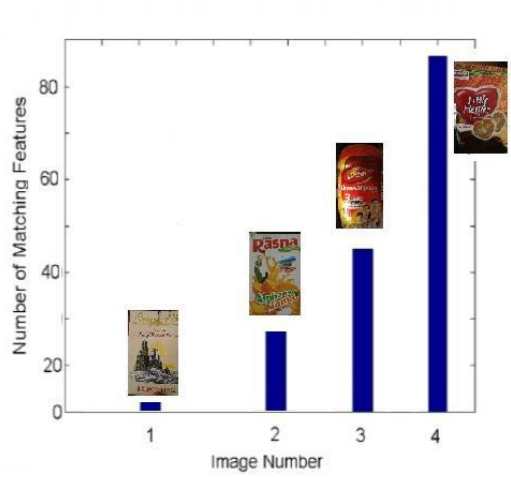

Fig. 4.c Features matching on user library images

We match the input image with each image in user library as shown in Fig. 4.c. The program outputs the name of the sample image which has the maximum matching features (min features to be matched is dependent on resolution of image, here we consider it to be 35). The server then fetches the nutrient information for that product from the Parse cloud database and sends to the client android phone.

IV. EXTRACTING TABLE FROM NFT

The user scans the barcode followed by photo of the nutrient fact table (NFT) when he gets a message that the product image which he took was not found in his user library. The reason the user scans barcode is because it is unique to each product and acts like a primary key in the Parse cloud database. The NFT image is uploaded to the server and saved in "/uploads/nft_images/" folder. The node.js server calls the NFT OpenCV program to crop the image to extract only the table. The table is later sent to an OCR web service which extracts the nutrient information as a string. This string is then matched using a regex string matching program to obtain individual nutrient's information (Calorie, carbohydrates etc.). The nutrient information is sent back to the client and also stored in Parse cloud for future use.

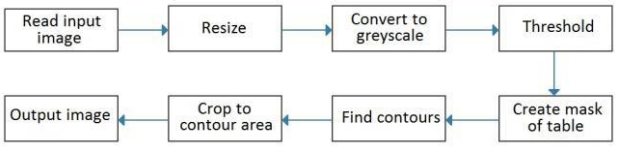

Fig. 5 Functional Flow Block Diagram for NFT processing

A. Thresholding

The NFT OpenCV program first converts the image to binary by applying a threshold. Since the image may be subjected to different lighting conditions in different areas (varying illumination) we go for adaptive thresholding. Using a global value as threshold value may not be good in conditions where image has different lighting conditions in different areas. In that case, we go for adaptive thresholding. So we get different thresholds for different regions of the same image and thereby giving us better results. Fig. 6 shows the output of adaptive thresholding on NFT image.

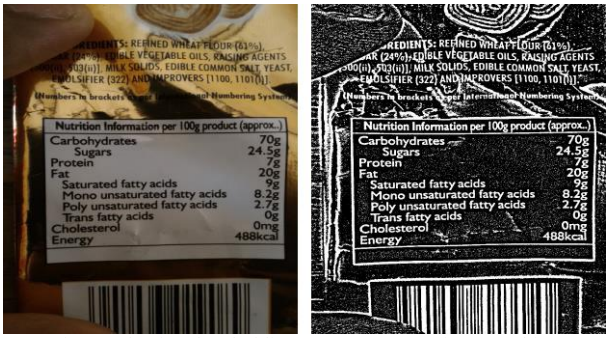

Fig. 6 Adaptive thresholding to convert RGB image to binary

B. Erosion and Dilation

We obtain horizontal and vertical projections of the table by applying morphology operations – erosion and dilation [6]. The kernel (structuring element) slides through the image. Erosion function erodes away the boundaries of foreground object. In erosion a pixel in the original image (either 1 or 0) will be considered 1 only if all the pixels under the kernel is 1, otherwise it is eroded (made 0). Dilation is just opposite of erosion. Here, a pixel element is '1' if at least one pixel under the kernel is '1'. Erosion removes white noises, but it also shrinks our object. So we dilate it after the noise is gone to increase our object area. We use a horizontal rectangle kernel to extract horizontal projection and vertical rectangle kernel to extract only vertical projections. The projections obtained are shown in Fig. 7.

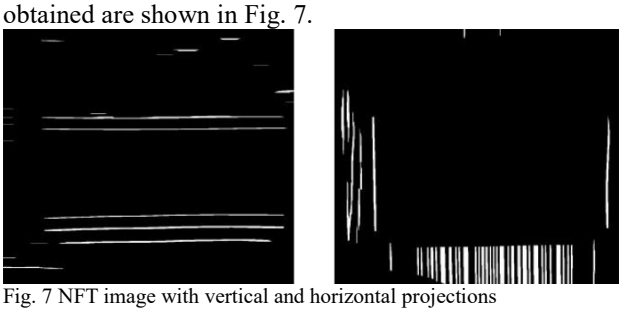
Fig. 7 NFT image with vertical and horizontal projections

C. Contour Detection

A mask is created by combining (OR operation) both horizontal and vertical projections. We apply the mask on the table and then detect contours of the table. Contours gives the boundary, which is then cropped to extract only table as output image. Fig. 8 shows these steps.

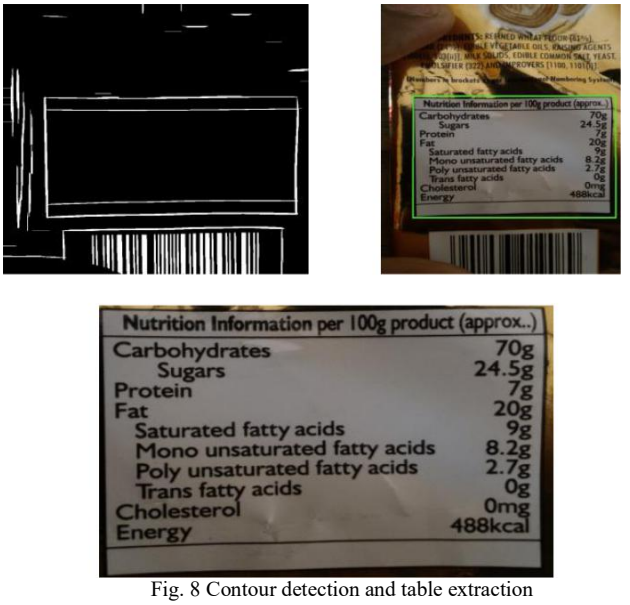

Fig. 8 Contour detection and table extraction

V. OBJECT CATEGORIZATION OF GROCERY

The user scans a grocery item by taking a photo of it and uploading to the server. The image is uploaded to the server and saved in "/uploads/grocery_images/" folder. The node.js server calls the Object Categorization OpenCV program to find the similar image in the grocery train library.

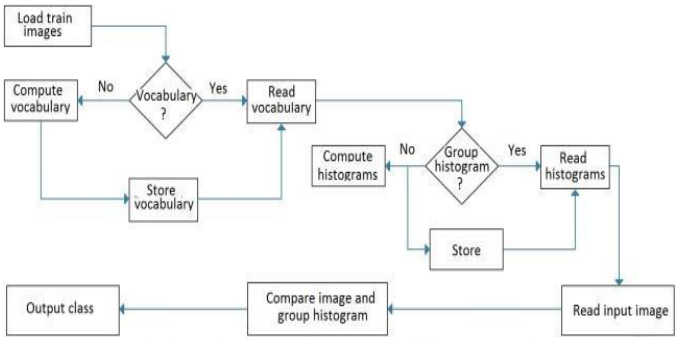

Fig. 9 Functional Flow Block Diagram of classifying grocery in Object Categorization program

The Object categorization program takes input grocery image and compares its features (words) with features of the train images in the grocery library using the concept of Bag of Words (BOW) and SVM Machine learning [7]. The training set of images contains all possible classes. Here classes are all possible groceries and class names are the grocery name. For the demonstration we have considered three classes of grocery – Tomato, Onion and Potato.

A. Building Vocabulary

We can use the bag of words model for object categorization by constructing a large vocabulary of many visual words and represent each image as a histogram of the frequency words that are in the image [8]. The figure (Fig. 10) illustrates this idea. First, we need to build a visual dictionary or vocabulary.

Vocabulary for the training set containing all classes is computed by taking a large set of training images and extracting descriptors (using SURF) [4] and clustering the set of descriptors (using k-means algorithm) of 1000 clusters (here k = 1000). Clustering is grouping a set of objects in such a way that objects in the same group (called a cluster) are more similar to each other than to those in other groups (clusters). The cluster centers act as our vocabulary's visual words.

B. Computing Group Histograms

In the second step, we need to scour the training set again to compute group histograms (the responses to the vocabulary). For each training image, we represent it using our bag of words model in the following manner. First we extract descriptors from the image around detected keypoints. Next, for each descriptor extracted we compute its nearest neighbour in the vocabulary. Finally, we build a histogram of length k where the i'th value is the frequency of the i'th vocabulary word [8]. This part requires vocabulary computed. We group the histograms of the same class as a labelled training data, where the label is the class name (grocery name). We make use OpenMP multi-threading to make the calculation parallel, and hence faster on multi-core machines, thereby reducing the execution time significantly.

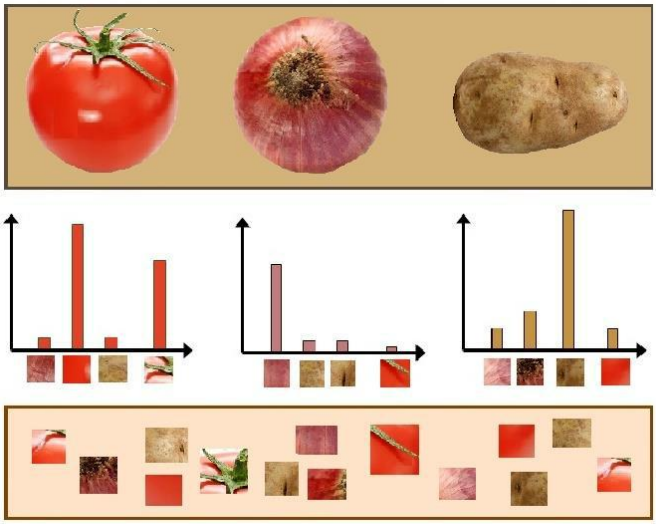
Fig. 10 Bag of words – representing object as histograms of words occurrences

C. Training SVM

In the third step we start training the Support vector machine (SVM). SVM is a technique for building an optimal binary (2-class) classifier. In other words, given labelled training data (supervised learning), the algorithm outputs an optimal hyperplane which categorizes new input example image i.e. the class which the input image belongs to. For a linearly separable set of 2D-points, the image has two types of data, red and blue. We can see that there exists multiple lines that offer a solution to the problem (Fig. 11.a). We need to find the best line among these solutions. A line is bad if it passes too close to the points because it will be noise sensitive and it will not generalize correctly. Therefore, our goal should be to find the line passing as far as possible from all points. Because there can be noise in the incoming data, this data should not affect the classification accuracy. Taking a farthest line will provide more immunity against noise [10]. The operation of the SVM algorithm is to find the hyperplane that gives the largest minimum distance (margin) to the training examples. So to find this Decision Boundary, we do not need all the training data, we just need the ones which are close to the opposite group. In our image in Fig 10.b, they are the one blue filled circle and two red filled squares. We can call them

Support Vectors and the lines passing through them are called Support Planes. They are adequate for finding our decision boundary [10]. Here in the figure (Fig 11.b) first two hyperplanes are found which best represents the data. The decision boundary is defined to be midway between these hyperplanes.

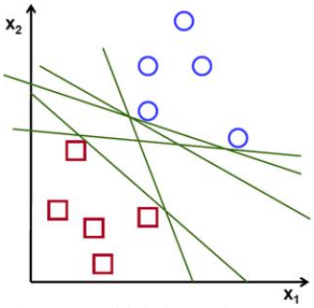

Fig. 11.a Multiple hyperplanes

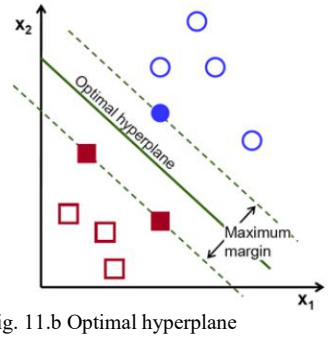

Fig. 11.b Optimal hyperplane

SVM builds a binary (2-class) classifier while we need multi-class classifier as we have 3 classes. We will train the 2-class SVM in a 1-vs-all kind of way. Meaning we train an SVM that can say "yes" or "no" when choosing between one class and the rest of the classes, hence 1-vs-all. SVM also needs a labelled training data. During building of labelled training data (sample histograms along with the labels) for each of the classes, we put in the positive samples and mark the labels with '1', and then put the rest of the samples and label them with '0' [9]**.** In order to reduce the execution time for future runs, the vocabulary and histograms for the training images are computed during the first run are stored in a YAML file and read from it directly during the future runs. We also use OpenMP multi-threading to run the loop parallel.

D. Testing SVM for Prediction

In the last step, given an input image, we represent it using our bag of words model as in the second step to build a histogram. We predict the response for the input image with classifiers obtained from training. The output class is the class of the classifier for which the output decision function value (signed distance to margin) is minimum. When the program gets an image it runs the classifier and gives the right class i.e. the grocery name as output. Note that this program will output a class name, no matter what the input image is. The server then fetches the nutrient information for that grocery from the Parse cloud database and sends to the client android phone.

VI. CONCLUSION

Giving attention to the nutrient information of the food products is very important to prevent under-nutrition and over-nutrition in a human being. Therefore food recognition has become an important research topic in object recognition. In this paper we presented a way in which an android mobile application can allow users to keep track of nutrient information of the food products and grocery items bought during each grocery shopping. We made use of some of the important topics of image processing in building the application. This system has numerous potential applications. We could combine this application with a checklist system to remind users of the items they have to buy. This application could likewise track the client's shopping habits and make diet recommendations, hence prevent certain diseases, for example, obesity and diabetes.

REFERENCES

[1] Wen Wu, Jie Yang, "Fast food recognition from videos of eating for calorie estimation," IEEE International Conference on Multimedia and Expo, July 2009.

[2] Maruyama, Y. de Silva, G.C. Yamasaki, T. Aizawa K, "Personalization of food image analysis," Virtual Systems and Multimedia (VSMM), Oct 2010.

[3] Shulin Yang, Mei Chen, Pomerleau, D, Sukthankar, R., "Food recognition using statistics of pairwise local features," IEEE Computer Vision and Pattern Recognition (CVPR), pp.2249-2256, 13-18 June 2010.

[4] H. Bay, A. Ess, T. Tuytelaars, L. van Gool, "Speeded-up Robust Features (SURF)", Computer Vision and Image Understanding (CVIU), June 2008.

[5] Marius Muja and David G. Lowe, "Fast Approximate Nearest Neighbors with Automatic Algorithm Configuration", in International Conference on Computer Vision Theory and Applications (VISAPP'09), 2009.

[6] Vladimir Kulyukin, "Vision-Based Localization and Text Chunking of Nutrition Fact Tables on Android Smartphones", Utah State University, Logan, UT, USA.

[7] G. Csurka, C. Dance, L. Fan, J. Willamowski and C. Bray, "Visual categorization with bags of keypoints", in ECCV Workshop on Statistical Learning in Computer Vision, 2004.

[8] Bag of Words Models for visual categorization, [Online]. Available: http://gilscvblog.com/2013/08/23/bag-of-words-models-for-visual-categorization/

[9] A simple object classifier with Bag of Words using OpenCV, [Online]. Available: http://www.morethantechnical.com/2011/08/25/a-simple-object-classifier-with-bag-of-words-using-opencv-2-3-w-code/

[10] Machine learning – OpenCV python tutorials [Online]. Available: http://opencv-python-tutroals.readthedocs.org/en/latest/py_tutorials/py_ml/py_table_of_contents_ml/py_table_of_contents_ml.html