UNIVERSITAT POLITÈCNICA DE VALÈNCIA

etsinf

Escola Tècnica Superior d'Enginyeria Informàtica

Escola Tècnica Superior d'Enginyeria Informàtica

Universitat Politècnica de València

Aplicación Android para clasificar setas presentes en España mediante procesamiento de imagen

TRABAJO FIN DE GRADO

Grau en Enginyeria Informàtica

Autor: Àngel Cubertorer Segarra.
Tutor: Dr. Salvatore Filippone
Dr. Francesc Muñoz Escoí

Curso 2015-2016

Aplicación Android para clasificar setas presentes en España mediante procesamiento de imagen

Resumen

La recolección de setas es una afición con una larga tradición. Las setas pueden ser utilizadas para medicina o como alimento. Dentro del mundo de las setas hay muchas especies diferentes y algunas de ellas guardan similitud en cuanto a forma y color. Esta similitud puede dar lugar a errores.

En este proyecto se propone un programa para clasificar setas a partir de una imagen. Este software utiliza diferentes procesos de Visión Artificial para conseguir este resultado. Estos procesos, pueden ser agrupados en tres niveles principales: Bajo Nivel, Nivel Medio y Nivel Superior. El programa propone un algoritmo para eliminar el fondo de una imagen mediante: la ayuda de un usuario que pulse las regiones de interés de dicha imagen, la computación del histograma de colores y comparándolo con otros histogramas en nuestra base de datos. Y de esta manera podamos encontrar a qué clase pertenece la seta.

La tecnología empleada para completar este proyecto son: librerías de Visión Artificial de Código Abierto “OpenCV” y herramientas relacionadas con el desarrollo para Android.

La puesta a prueba del programa muestra que funciona adecuadamente y que logra: identificar el área perteneciente a la seta, computar y comparar el histograma de colores y clasificar la seta seleccionada, en imágenes tomadas bajo las condiciones apropiadas. Sin embargo, quedá pendiente la obtención de suficientes resultados para evaluar el funcionamiento del programa con grandes volúmenes de datos. Como se comenta en el artículo, otros aspectos del sistema deberán ser también objeto de mejora.

Palabras clave: procesamiento de imagen, setas, OpenCV, Android.

Acknowledgements

Maria Vicenta Segarra Berges. Mamà, for teaching me that no matter the problem I should never back down and do my best. Thanks for all the support during the MSc and the thesis since the beginning until the end.

Dr. Salvatore Filippone, for your guidance and supervision in every stage. Thanks for having the ambition to accept this project.

Dr. Francesc Muñoz Escoí, for your support and your willing to help through the whole process not only in the thesis but all the paperwork in the UPV so I get recognised the work I completed in this project. It was not the easiest thing but you accepted anyway and I am glad you did so.

Pau Cubertorer Segarra, my beloved brother. Through your support and experience you made me feel that I could always rely on you when I was not able to see the path to follow.

Anna Audiovisuales, my girlfriend. I could not have asked for you to do anything more during these months. You supported me through the MSc and now during the thesis. Thanks for understanding what I was going through at every step and doing everything you could for helping me.

Contents

List of abbreviations

Android: Android operating system

App: Application program

SDK: Software Development Kit

IDE: Integrated Development Environment

NDK: Native Development Kit

GUI: Graphical User Interface

XML: eXtensible Markup Language

RGB: Red Green Blue colour space

HSV: Hue Saturation Value colour space

H: Hue

S: Saturation

V: Value

1D: one dimension

2D: two dimension

UPV: Universitat Politècnica de València

OOP: Object Oriented Programming

List of figures

1. Introduction

Mushroom collection is a dangerous hobby if those who practice it are not capable to differentiate between classes of fungi. Until recent years, when the phone and, thus, apps for these devices were included in our daily life, inexperienced people would rely on some people who, through studying books and years of experience in the field, were able to classify a mushroom when they would find one.

Nowadays, mobile phones have improved the availability of information, as we can reach it from almost any point on the planet earth. This is why some phone's apps that aim to help people to identify mushrooms have been published in Spain as *Setas Bolets MushTools* or *Buscar Setas*.

However, they require the user to input a lot of information about the mushroom's characteristics. I proposed this project for my MSc thesis at Cranfield in order to simplify and semi-automate this process, with a minimum input by the user required. In order to perform the classification, the mushroom's colour has been used as feature to differentiate between classes. In the following sections of this paper we will define how object classification in computer vision is performed and the requirements found in the literature to implement it will be described.

1.1 Definition of object classification

In imaging science, image processing is defined as the field where, having an image as an input, an output is retrieved. This result can be another image or some useful information.

The field of image processing is a broad field where object classification is one of the main processes applied. Object classification in image processing is the technique of labelling an object appearing in an image. The classification is made by means of some object's features that can define the different classes.

1.2 Industry fields

1.2.1 Chain production

Many companies use computer vision system at some point on their manufacturing in order to determine the quality of their products and then discard those which do not achieve the minimum specifications. Duan, Wang, Liu, & Li (2007) proposed a system for detecting damages on beer bottles using histograms for determination of inspection area and two neural networks for defect detection. González, Villamizar, & Lopez (2014) suggested another system for determining the level of liquid in a soda bottle and exclude those below the acceptable limit, for doing so they used conversion from RGB to HSV, removed V component and find contours, for later on obtaining the filling ratio.

1.2.2 Security

Object detection and classification is widely used in the security industry, mostly for

detecting and tracking threats, this means that the object is detected and then classified as a threat or not. Using image processing like applying a threshold, Canny edge detector and find contours helped for detection of high speed flying objects such as bullets or rockets (Cho, Kim, Kim, Lee, & Kim, 2016).

1.3 Object classification in Agriculture & food

A system for identification of fruits and vegetables, using RGB to HSV conversion, Canny edge detection, histogram comparison and using texture as a feature, has been successfully tested (Chowdhury, Alam, Hasan, & Khan, 2013), (Ahluwalia & Karani, 2014).

Measuring quality of fruits for public consumption within four different categories was performed using background removal by colour thresholding and average colour coordinates (Blasco et al., 2009).

Using a computer vision system that involves background removal, applying Gaussian filter and white colour thresholding allows user to detect and count number of tangerine flowers in an image for estimate tangerine crops (Dorj, Lee, & Lee, 2013).

Another system for automatic apple harvesting based on computer vision using contour tracing and colour pattern detection was proposed (Jeet Sharma & Kaur, 2014).

1.4 Mushroom classes

There have been discovered many fungi classes in the world. It has been said that there are more than 5.1 million species (Blackwell, 2011). Not all of these species are with the typical mushroom shape nor edibles. Because of this, I have narrowed the species to classify to some of the most common species that can be found in Spain.

Figure 1: Miniatures of the mushroom classes in the system

The species selected and shown in Figure 1 are:

1- *Cantharellus cibarius*.

2- *Boletus edulis*.

3- Tricholoma terreum.

4- Agaricus campestris.

5- *Craterellus cornucopioides.*

6- Marasmius oreades.

7- Amanita caesarea.

8- Hygrophorus Latitabundus

9- Lactarius sanguifluus.

2. Methodology

From each of the following stages, information gathered was used to keep researching on the next or previous stages. As an example, the validation of the mushroom isolation resulted in coming up with new features such as the user tapping on the image for cap and stem positioning.

2.1 Identify needs

There were three main needs from where the program took its structure.

First, I myself am a fan of collecting mushrooms, so I was aware since the beginning that the major need was to be able to identify the class a mushroom belonged to.

Second, parting from the first need, we asked ourselves, what is necessary in order to classify an object appearing on an image? And the solution is to first find the object in the image.

Finally, once we have detected the object in the image, what is left in order to classify this detected object? To compare it somehow with other objects

This way, we were able to identify the needs of our program and started exploring the methods in order to satisfy these needs.

2.2 Exploration of methods

The first sight into the methods to use was given by the literature review and the lectures at Cranfield University. After this, we were able to recognise which were the transformations that should be performed and, thus, the methods to be applied. Of course, not all of the initial methods thought were applied. During the development step the output of these methods were analysed and those that did not give the desired results were left apart since we wanted to apply only those that suited the best in our system. Then, individual execution time of the methods was performed and some were removed since the cost of a little improvement in the solution was too high. In Figure 2 we can see the methods that were used in the final solution separated by the level of processing it belonged to.

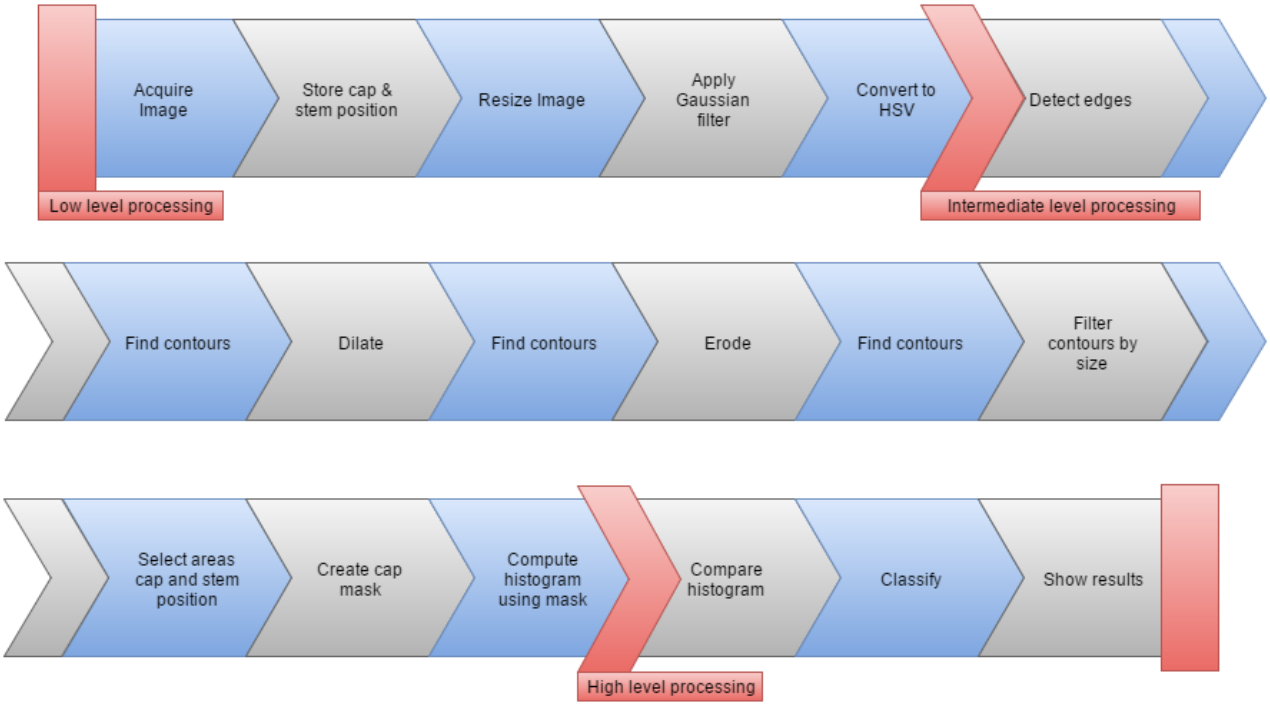

Figure 2: Program workflow with methods used

2.3 Development and validation

The information used in order to develop the final solution was gathered from relevant Computer Vision and Programming sources. During the development of the system a method for debugging is needed so in case of an unexpected error occurring we are able to identify the cause of this error and quickly fix it. For this case, the default debugger for Visual Studio was used since it was the environment used for carrying out the development of the computer vision part. Furthermore, since we were working with images, it was also needed during the development, to know what were the values an image had on the different steps of the program. For this purpose, the Visual Studio extension Image Watch was used. This extension is merged into the default debugger and lets you see the value of an image at the desired step.

Regarding validation, it was clear from the very beginning that the objective of this thesis was to validate this program in a relevant environment, if the conditions were met. However, these conditions were not met since it is not possible to find mushrooms in their real environment within the time constraints for this thesis. That is why the validation of the technology had to be made using a desktop version of the program and with images already captured. The validation was carried out by feeding the system with previously unseen images and then checking if the program was able to successfully label them.

3. Computer vision software for classifying mushrooms

Image processing and image analysis are the main components of computer vision with many algorithms known as successful methods for achieving results. Furthermore, image processing and image analysis implies several steps that can be separated into three different levels: low level processing (acquiring the image and pre-processing), intermediate level processing (image segmentation and description) and high level processing (recognition and interpretation) (Sun, 2000). Image processing is not supposed to obtain any information from the image but to remove noise and prepare the image for our system.

3.1 Technology

3.1.1 Android SDK

Android is a mobile operating system developed by Google, it is mainly designed for touchscreen devices. Android is the most installed operating system of any type.

The SDK includes among other things a set of software libraries, a debugger, an emulator and a IDE known as Android Studio.

Android has third-party apps that can be downloaded from an app market named Google Play. We built one of these apps using the SDK and Android Studio. For Android Studio we needed to develop the GUI using XML language and Java

language for the activities.

3.1.2 Visual Studio

Visual Studio is an IDE developed by Microsoft and accepts different programming languages. One of its usages is to develop computer programs.

In our case, we decided to use Visual Studio for debugging purposes, as it allows including the OpenCV software libraries and the compilation time was shorter than on Android Studio. So at the beginning the part of the program dedicated to image processing was developed in C++ using Visual Studio and later on transported into Android and Java.

3.1.3 OpenCV

OpenCV is a set of computer vision and machine learning software libraries, developed firstly by Intel, now counts with more than 2500 optimised built-in functions that recognise objects, faces, help with robot navigation, etc.

We chose OpenCV as our computer vision libraries because of the ability to be included into Android projects, the broad online user community, and the possibility to work both in Android and Java.

3.1.4 OpenCV4Android

OpenCV4Android is an SDK made available by OpenCV after the necessity of using OpenCV functionalities in Android apps. It allows users, both, to use the OpenCV libraries in Java and reuse C++ language code on our app by using a wrapper and native development.

In this case, we first decided to reuse the C++ code we had for computer and use a wrapper and the NDK in order to that. However, the instructions on how to do that are out of date and not clear, so after a long time trying to configure it and failing we decided that was better and would save us time to include the OpenCV libraries and translate the C++ code into Java language.

3.2 User interface

The app's GUI has been written in XML and rendered using Android Studio. The app has 4 different GUI one for each activity that the user must perform. In Figure 3 we can see the workflow the GUI follows.

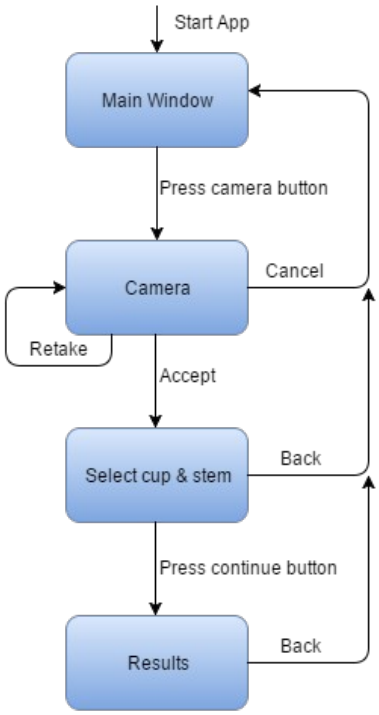

Figure 3: App workflow

3.2.1 Main view

The Main view is the very first window that the user has from the app. It is the view that prompts when the app is initialized. In this window we can find a button, that will lead us to the next step, opening the camera for capturing the mushroom, and the app's logo.

The possibility of loading an image from the gallery is not given, the reason is that this app is intended to be eco friendly. Even when the mushrooms are toxic it is beneficial for the environment to let them grow, so if the user takes a picture of the mushroom, collects it and later on checks whether it is toxic or it is not, thus throwing it away and having harmed the environment. In Figure 4 it can be seen the design of this window.

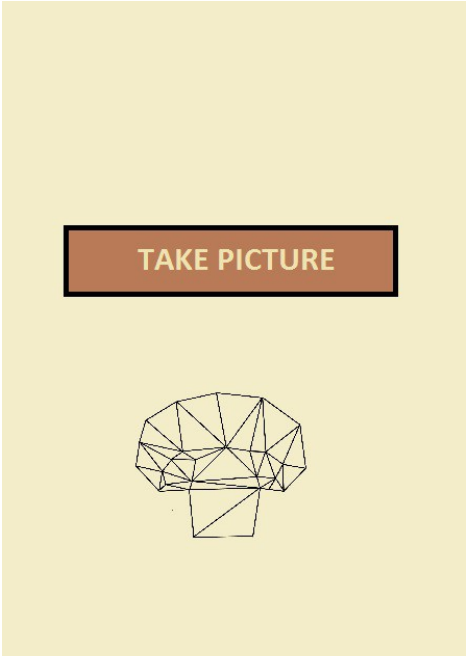

Figure 4: Main view design

3.2.2 Camera

The default Android camera appears, the user then can take the picture or go back into the main view. After taking the picture the user is given the possibility of retaking it, cancel the action or accept it as the image for querying. If the last option is selected the user advances into the next step that is marking the cap and stem of the mushroom in the picture.

3.2.3 Select cap and stem

After the image has been taken and the user selected it as valid for classification, it is shown again in a new view, where now, the user will have to tap on the cap and the stem of the mushroom and then click the button 'Continue' in order to get the results. The first tap must be on the cap and the second one on the stem for the program to work properly. In Figure 5 we can see the view after the cap and stem are selected. In this view, if the user makes a mistake when taping and desires to fix it there is no other solution than going back to the main view, take another picture and start the process again. This is one of the point to improve on future work.

Figure 5: Mushroom with cap and stem touch-marked by user

3.2.4 See results

Finally the classification has been performed and the results are shown. The result consists of the image pending of classification, the model image of the class and a brief description of the species of mushroom in case of being successfully labelled. Otherwise, the input image and a cross will be prompted.

Figure 6 shows the brief description and the model image that the user will see. Therefore, it is the user who, at the end, can decide whether the app performed a correct classification or not.

Figure 6: Result window for a correct classification

3.3 Low level processing

Once we had the picture we could start working with it. However, raw images from a camera present some noise that could make our system not to work properly. For this reason, we needed to apply some preprocessing to our image in order to try and minimise the noise and make the image more suitable for our app. In the following lines, the steps this preprocessing consisted of are explained.

First of all, to reduce the execution time of the program, make it easier to work with the image captured by the user and standardize the image size our app works with, we had to resize the captured image. The standard size which the images are resized to was stated as (480x450), this size was chosen after several processes of trial and error over our set of images.

After resizing we had to get rid of the noise present on the image, there are different filters to use like median filter, normalized or Gaussian filter. By applying one of these filters we will “smooth” the image. For this, we chose to use a 5x5 Gaussian filter because is the only filter with weights for the influence of neighbours on the new value of the pixel. In this case weights means how much a pixel influences when calculating the new value. This filter convolves each pixel, and its neighbours, at a time in the image with a Gaussian kernel and modifies its value to the addition of all the results in the kernel. In Figure 7, it can be observed how the weights of the neighbours are not the same through the whole kernel, it increases when getting closer to the centre pixel. We can appreciate the smoothing that results after applying the Gaussian filter to the image in Figure 8.

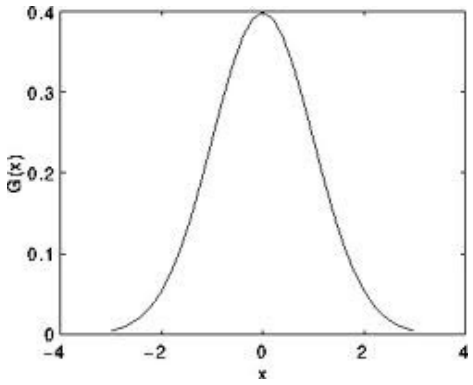

Figure 7: Gaussian distribution

Figure 8: left: normal image, right: same image after applying Gaussian filter

Then, the next step was to convert the image from RGB colour space to HSV colour space. The reason for this conversion lies on how HSV expresses the colours. In Figure 9 we can appreciate that HSV colour space is composed of three variables: Hue indicates the pure colour; Saturation expresses how bright the colour is; and Value reveals how dark the colour is. It can be deduced that the V component is very sensitive to change in lighting conditions making a colour based system vulnerable. And the opposite way, if we change to HSV colour space and remove the V channel from the image we obtain a robust system against changing lighting conditions. Because of this, HSV is the colour space preferred for edge detection and histogram comparison (Jeon, 2013). In Figure 10 we can appreciate the difference between colour spaces for the sight of the image.

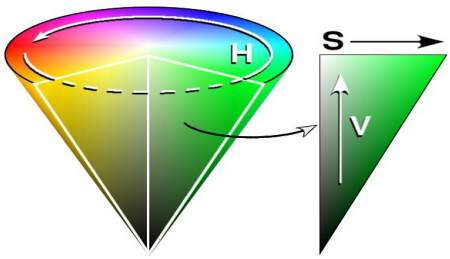

Figure 9: HSV distribution scheme

Figure 10: left: image in RGB space; right: same image in HSV space

3.4 Intermediate level processing: Mushroom isolation

Having completed the preprocessing, we then had an image with reduced noise and we were ready to move forward.

The second process to conduct for the classification was to identify where the mushroom was located on the image. By isolating the mushroom, we removed the information that was not needed that caused our system to not be stable, this was because we wanted to compare the mushroom by its colour since it is something constant on every mushroom from the same class while the background of the images are different from one to another. Following are the steps taken in order to achieve the isolation.

First of all we needed to recognise shapes on the image since this would help us to find objects, thus, being able to distinguish the mushroom among these. What describes a shape is its contours so first we needed to find the edges. For edge detection the Canny edge detector method implemented in the OpenCV libraries was used, with the following thresholds: low threshold was set to 70 and high threshold to 150. The high level threshold defines the limit where, if the gradient result of the possible edge is above the threshold, this edge will be marked as a pure edge. Those possible edges between the two thresholds will be marked as an

edge or not depending on whether they are in contact with a pure edge or not, and the ones below the low threshold will be discarded. A threshold too high will miss information and one too low will include irrelevant information. No tried and tested approach for finding the proper thresholds exists yet. The usual approach is the so called try and error, and is the one followed during this project.

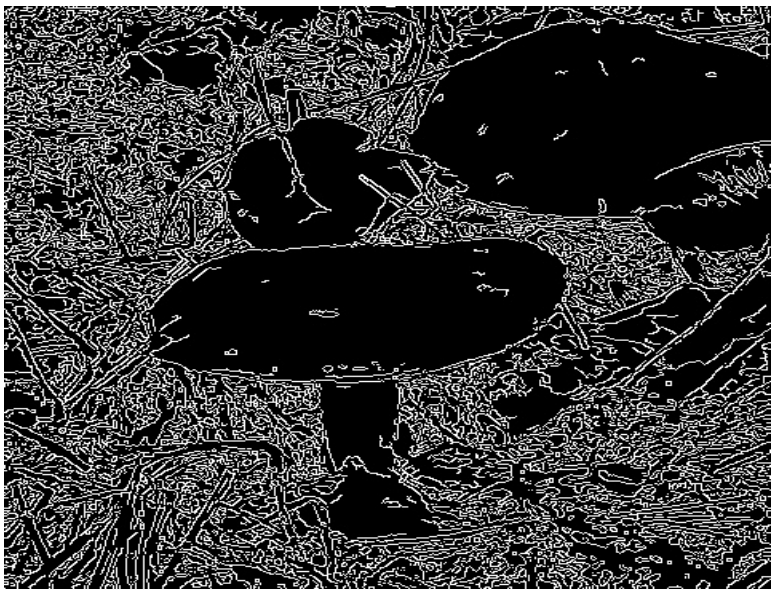

Figure 11: Result image after applying Canny edge detector

In Figure 11,as a result of the edge detection, we were able to detect most of the edges as well as some noise.

The edges are just lines on the image so we needed to join together those edges that would form the contour of an object. For doing so, we used the findContours() method implemented in the OpenCV libraries. This method retrieves contours from a binary image using the algorithm suggested by Suzuki & Abe, 1985 that distinguish between parent and child contours in the image.

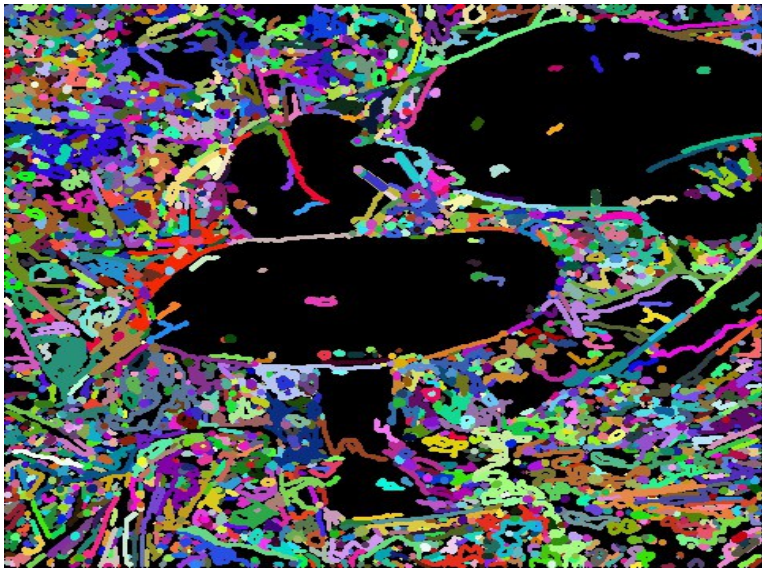

Figure 12: Result image after finding contours

As it can be seen in Figure 12, where each contour got assigned a different colour, we identified the contours of the image, however, the true contours got broken down into small unconnected contours and the background appeared to have too much noise. We needed to join the short contours in order to get closer to the real-life contours and unify the background in order to facilitate the mushroom isolation.

For joining the small contours we used dilation and erosion. The dilation method modifies the value of the pixels in an image. The new value is determined by its neighbours. This process overlaps a kernel through the whole image substituting the value of the pixel positioned on the anchor, which is the centre position of the kernel; by the maximum value pixel in the kernel. This causes the bright region within an image to “grow”. So we dilated the image. By doing so, we removed some of the black holes appearing on the mushroom. In the example in Figure 13 the anchor pixel would get 255 as its new value.

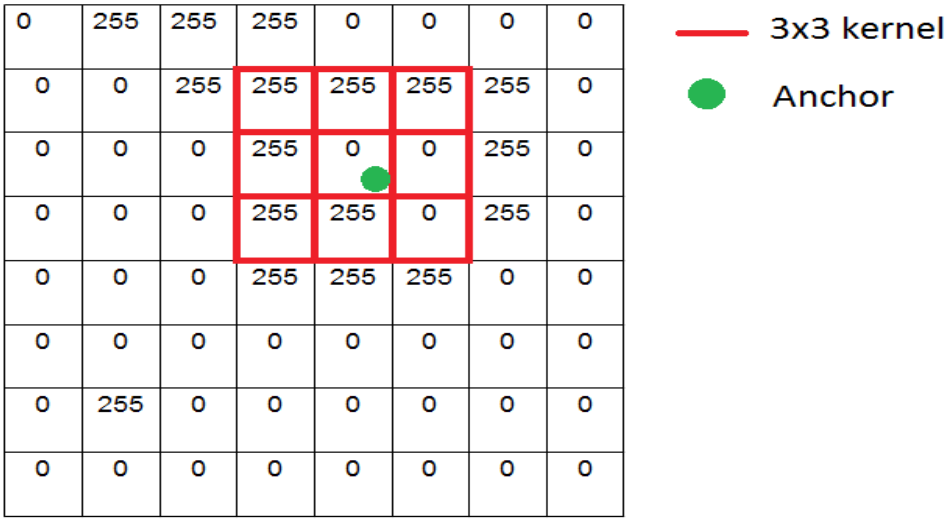

Figure 13: Representation of a kernel overlapping an image

We then found the contours again, with the previously explained findContours() method, in this case we obtained a much clearer image of contours. However, Figure 14 still had its contours open and we wanted to close them so we could achieve more realistic shapes.

We closed the contours by applying erosion, this method can be seen as the opposite of the dilation. Unlike in the dilation, the anchor pixel is substituted by the minimum pixel value inside the kernel. In the example in Figure 13 the anchor pixel would remain with a 0 value. Even though, they are opposite processes, the appliance of one after the other does not result on the same initial image, since one 0-value pixel surrounded with all its neighbours being 255-value pixel when applying dilation will be modified to 255 but if we apply then erosion it will stay in 255. The result of this process can be seen in Figure 15 .

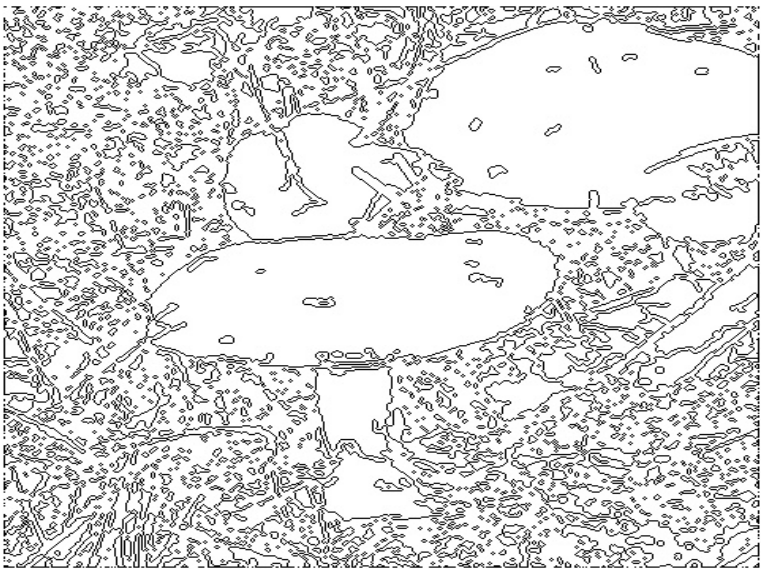

Figure 14: Result image after finding contours on a dilated image

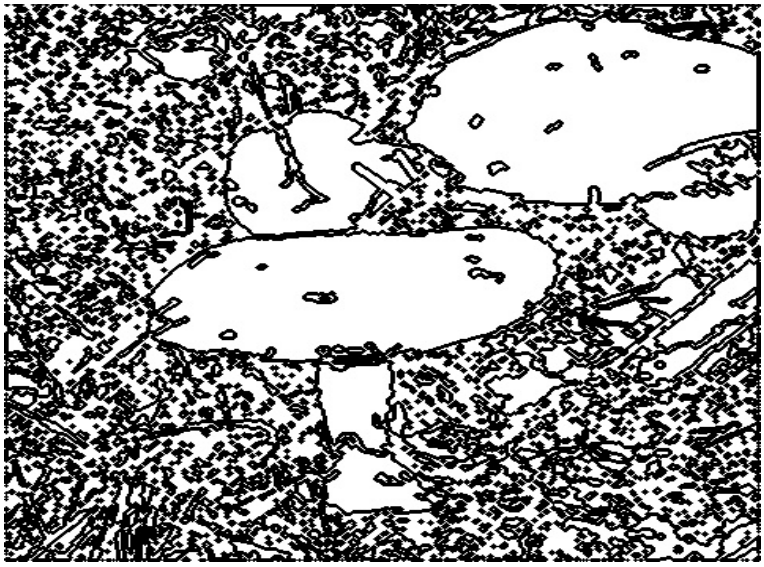

Figure 15: Result image after eroding the image on Figure 14

Once we had most of the contours closed, we needed to remove those that are too small, meaning it could have been produced by the processing of some noise in previous steps, and taking into account that the object of interest in the image is the mushroom what implies that its components have a big size, we decided to filter the contours in the image by its area size. The result of this filtering can be appreciated in Figure 16 where we obtained an image with few big contours.

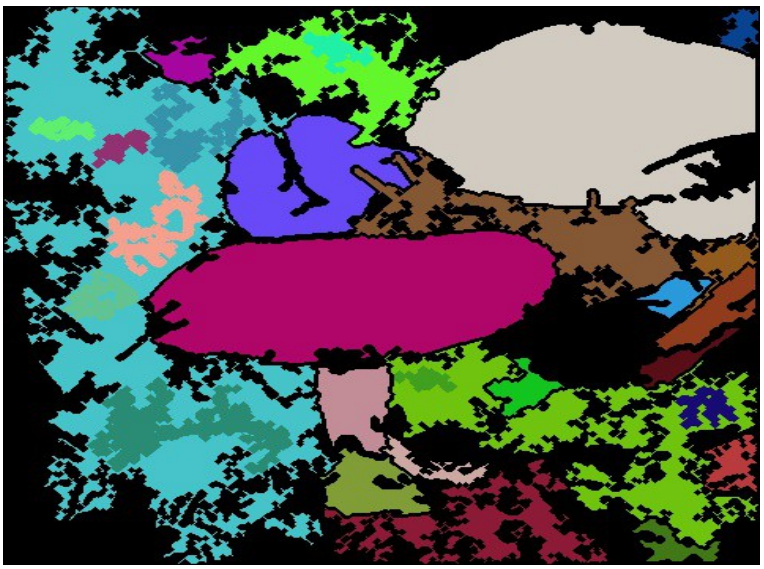

Figure 16: Result image after filtering contours by area size

The last step for the isolation was to be able to select the areas that constitute the mushroom. In this step we retook the points entered by the user as where the cap and the stem are, we then computed which areas these points belonged to. So then we recorded those areas as the mushroom's components. This step is the critical step on the program, since it influences highly the correctness of the system and is more prone to errors. These errors can happen due to either the user marked the cap and the stem in the wrong places on the image or there was some noise on the image that made the mushroom shape to not be recognised properly. In both cases we would compute the wrong histogram masks, thus, computing the wrong colour histograms what would cause a wrong classification.

3.5 Intermediate level processing: Extract colour information

The method used for computing how similar two images are is to compute the colour histogram of both images and obtain the distance between histograms.

A colour histogram is a representation of the distribution of colours in an image. It represents the number of pixels of an image that falls within a range of intensity. In Figure 17 we can see a 1D histogram, in our case we worked with 2D histograms. In the case of OpenCV the dimension of the histogram is not the variables represented on the axes of it but the number of parameters whose pixel distribution are being computed on the histogram.

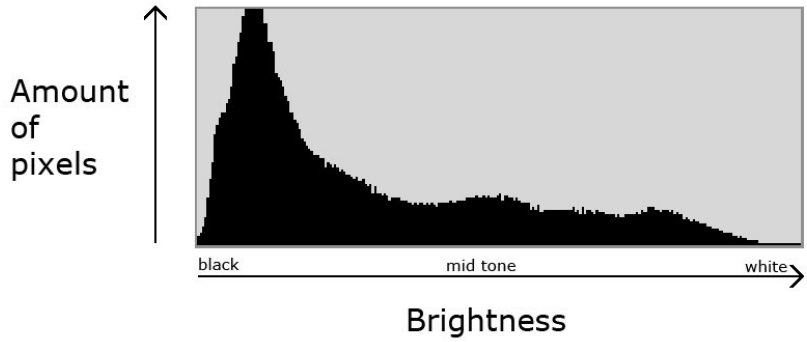

Figure 17: Example of 1D histogram

Coming from the previous section, having identified the mushroom's components we proceeded to extract the colour information needed. As previously mentioned, the colour was the feature selected for performing the classification as it is the main classification criteria used by humans.

As stated on previous sections, the Value channel of the HSV image is sensitive to changing lighting conditions so before calculating the histogram we drop the V channel from the image.

For computing the histograms we used the function calcHist() from the OpenCV libraries. This method allows us to compute the histogram of just a part of the image by passing a mask as a parameter when calling the function. We, then, computed two different binary masks, for the cap and stem, from the areas obtained in the previous step and we passed these masks as parameters when computing each histogram, but before that we made applied a process called closing, that removes some holes and sharpens the shape of the mask by turning into bright those dark regions surrounded by white pixels. Both the masks (Figure 18, Figure 20) and the result of this process (Figure 19, Figure 21) can be appreciated below these lines.

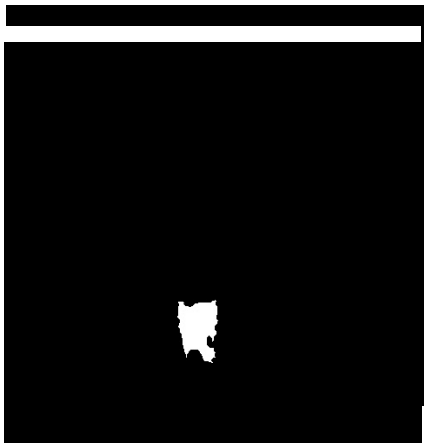

Figure 20: Stem histogram mask

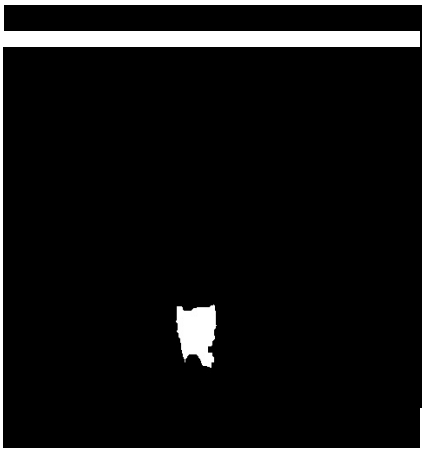

Figure 21: Stem histogram mask after applying closing process

We finally obtained the histograms for the cap and the stem and we were ready to get into the final part of comparing it to the models of each class. In this project, since there are only 9 classes, using only the cap histogram for performing the classification was enough. However, the program already computes the stem histogram in view of future work, because if we increase the number of classes more parameters for classification will be needed. In the future work section can be read a deeper explanation.

3.6 High level processing: Mushroom classification

From previous steps we had computed successfully the histogram of the cap and the stem from the user input. It was only left then to compare these histograms with the models.

In Figure 22 can be seen a code snippet where we:

1. Loaded the class histograms that are stored on a local file.
2. Calculated the similarity between these and the query image histogram. The higher the similarity, the higher the probabilities of the query mushroom belonging to that class.
3. However, we still needed to check if this most likely class is the true class, where the mushroom belongs to, since mushrooms that are not in any class would still show some similarity. This is the reason why we needed to define a set of limits independent to each class that would determine if the input mushroom belongs indeed to that class or not. If the similarity of the mushroom with the class was above the limit then we could label it as belonging to that class and thus being edible, otherwise it was being labelled as non-edible.

```
double maxComparison = 0.0;//var for storing maximum similarity
int maxPosition; //var for storing which model mushroom is the most similar
//open file with histograms for reading
FileStorage fs2("histograms.yml", FileStorage::READ);

//run over all ten type of mushroom histrogram
for (int i = 0; i < 10; i++)
{
    Mat modelCup;
    //load histogram of a model for comparing
    fs2[mushrooms[i]] >> modelCup;

    int compare_method = 2;// intersection method

    //obtain comparison's result
    double similarity = compareHist(hist_cup, modelCup, compare_method);
    //compare if it is more similar than the current most similar
    if (similarity>maxComparison)
    {
        maxComparison = similarity;//set new max similarity
        maxPosition = i;//set new max position
    }

}
```

Figure 22: Example code of loading and comparing histograms

Once we checked if it was above the limits of the most probable class then we could give the results to the user as explained in previous section about GUI.

3.7 Mushroom classes definition

In previous sections we stated that the histograms of each mushroom class were loaded from a file for later comparison with the query image. The procedure we conducted for computing those histograms and storing them on a local file was the same as when the program receives a new query image but with a slightly change of the code, instead of loading histograms from a local file and comparing we were writing the histograms into a local file with the name of each class.

The process above mentioned is unavailable for the user as the file containing the classes histograms is not intended for the user to modify it.

By keeping the histograms on a file instead of in a web server we allow users to use the app correctly whether they are connected to the net or not, and having in mind that mushroom collection usually happens in places with connection difficulties, as mountains or forest, we saw it as the best choice for our app.

4 Discussion

4.1 Results

Regarding the results of the app, we faced a big problem through the whole project and was receiving quality images taken with Android phones. As the usual season for the mushroom collection is Autumn and this work finished in August we could not gather together the images by ourselves or ask any other people to do so because there are no mushrooms at the time. No database with images of enough quality existed on the web even though we asked some experts clubs from Spain.

On the other hand, we still obtained local results for some images, depending on which conditions they were taken under, the system performed properly. We were able to achieve our objective of labelling mushrooms as different classes starting from an image and removing the background and comparing histograms. Figure 23 shows the result of computing the histogram of two images where it appears a mushroom in both of them, these two mushrooms belong to the same class, while Figure 24 is the result of calculating the histogram of the same two images, however, this time, we successfully removed the background. It can be seen that the images in Figure 24 are more similar than those in Figure 23, even though both cases contain mushrooms from the same class. With this we can see that the background was successfully removed with our steps and we improved the colour histogram comparison by doing so.

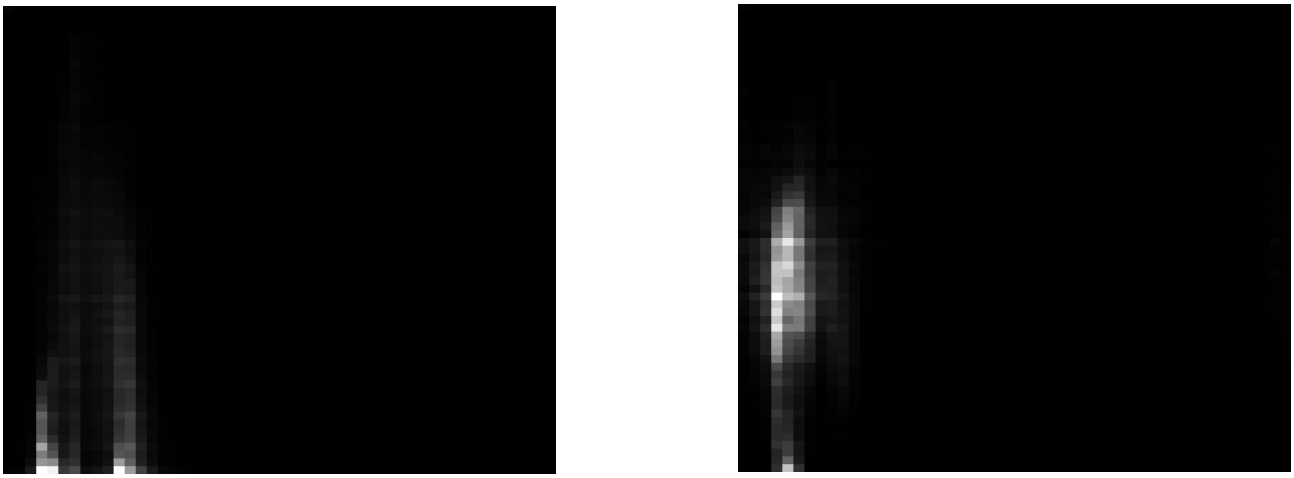

Figure 23: Two 2D histograms of images with same mushroom class and no background removed

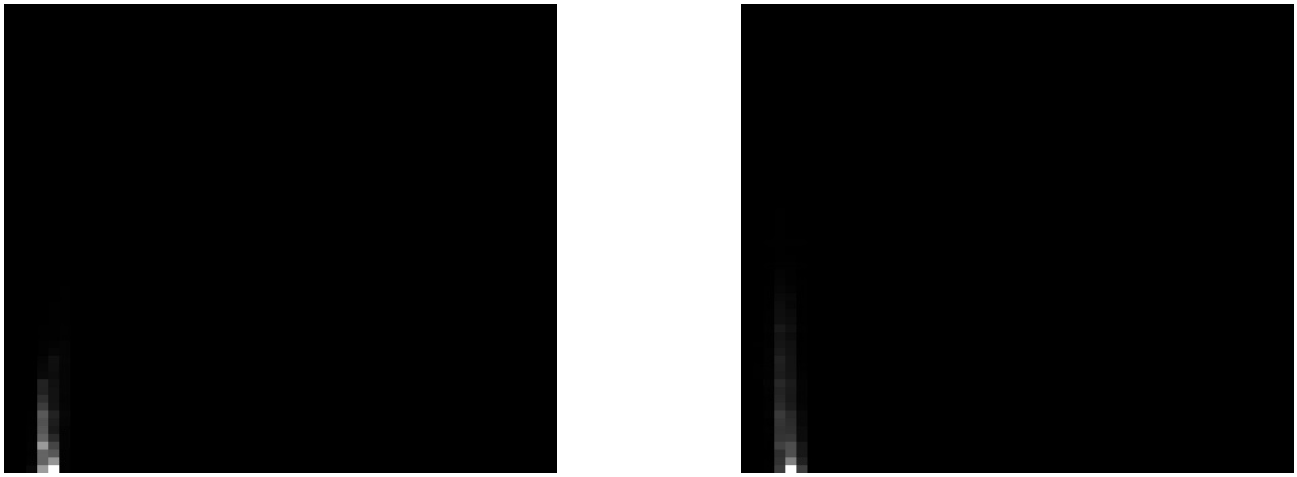

Figure 24: Two 2D histograms of image with same mushroom and background removed

4.2 Test plan

The solution we decided for gathering enough quality pictures captured by an Android phone is that before the mushroom season starts and experts go to the mountains to collect them, we will get in touch with mycology clubs around Spain and distribute them our app so, they will be able, at the same time, to offer it to their members with the condition that those members will not be making decisions based on the app results but based on their experience. So the app will only be available for a group of experts until enough feedback has been collected.

The expert users will provide us feedback on whether the app worked properly or not and in which cases it did fail. After executing our app and looking at the results given by it, the experts will take note of: the expected classification, the result classification, the date and time of the picture, and if the mushroom was placed in

the shadows or in the light. With this form we will be able to take record of the classification accuracy, which classes present more error, thus adapt our model class to more realistic values; and under which lighting conditions the picture was taken.

With this system we would be able to obtain enough objective reviews about the correctness of the app and then prepare statistical results for deciding if the program is accurate enough for being published on the market for users to use and trust it and if it is robust against lighting conditions because using the HSV colour space was the right choice in this case as a barrier against these changes.

5 Conclusion

Although, as stated in the previous section, we could not gather enough pictures for obtaining a reliable accuracy rate, we did successfully tested the program on different images obtaining the desired result of classifying mushrooms appearing on an image by removing the background and comparing colour histograms. This means that the program worked well for pictures captured under the right conditions of both the environment and the camera used.

In conclusion, we can say that the program proved that it is possible to classify different mushroom species and that the techniques used were the appropriate for the mushroom isolation and classification. It has been also proved that the colour histogram comparison is a reliable classification method when the background is removed and the histogram is computed for the mushroom shape only.

As a result of the work completed for this thesis I can say that I have learned several things that extend the knowledge I had from the lectures.

The main one is to plan a project like this for 3 months, with the aid of my supervisors, managing my time and resources in order to deliver a good result. In this case putting to work what I learned in the UPV course Project Management.

Then, being able to code both in Java and C++, as learned in the Introduction to Programming on my first year of studies at UPV and the course C++ at Cranfield University. As both of them are OOP languages my whole degree at UPV was useful because they taught me a way of thinking into the OOP world and I found small differences when translating code from C++ into Java.

This project, as any other, had stages of debugging and it was needed to keep the code in a clear structure and commented in order to save the work from day to day and still be able to remember what every line of your code does. All of these good practices were learned on the UPV course Software Engineering.

The courses Image Processing and Image Analysis at Cranfield University introduced me into the OpenCV libraries and their main functionalities that helped me through the whole project. However, I could have never imagined how wide were the possibilities that computer vision in general and OpenCV libraries in particular had. At this point I have more knowledge about the difference between colour spaces, being able to reduce the noise present on an image and being able to reconstruct shapes.

Finally, I learned a lot about the Android environment. I followed a Udacity and Google course for Android developers, which taught me how to connect to the cloud, build different activities on the same project and how to communicate among them; create a background thread where to place the majority of the workload in order to not stall the GUI.

6 Future work

6.1 Features

There are several new features for the program that could be implemented in the future. The first one is to add more species to the initial nine. If we want to increase the number of classes and keep the reliability of our system we should add new comparison parameters like the colour histogram of the stem, that is already being computed but not used as explained on previous sections, or shape features that are unique to some classes.

The second is to include GPS location and the possibility to save the location where a mushroom was found with the class information to a private map. This will help users to locate where the mushroom was found in previous seasons; since mushrooms, if not damaged, tend to grow on the same place every year.

Then, as previously stated, improve the user experience when selecting the cap and the stem from the image. It should be allowed for the user to slide the crosses around the picture, if needed, for marking the region of interest instead of having to start the whole process again.

Finally, expand the app to other mobile operating systems like Windows Phone or IOS for reaching those users that do not have Android on their devices.

6.2 Expert users

For the expert users, the same users that gave us the feedback will be given the possibility of signing in our app with an expert profile once the app has been released and is available in the market.

These expert users will keep the ability of giving feedback to the administrators of the system and will have new abilities.

One of this new abilities will be to upload pictures of misclassified mushrooms, so the model class can be modified in order to approach more realistic values. In this case, it will be needed a minimum number of modification requests made by different expert users in order for the administrators to modify the class histogram.

Another one is to set a new mushroom class that they think should be available for the rest of the users. In this case, as before, it will be needed more than one request in order to be taken in to account by the administrators. After the minimum number of requests have been reached but before setting the new class, the administrators will have to verify the class data.

Finally, in both cases the experts what they do is to make a request for a change to

be made by the administrators, together with the request the image captured is sent. After this, the administrators of the system will be the ones deciding if that change is needed or not, since the problem behind the request may be affecting a few isolated cases.

Bibliography

Ahluwalia, A., & Karani, R. (2014). Review Paper on Vegetable Identification and Detection using Image Processing. *International Journal of Current Engineering and Technology*, *4*(6), 4260–4262.

Blackwell, M. (2011). The Fungi: 1, 2, 3 ... 5.1 million species. *American Journal of Botany*, *98*(3), 426–438. http://doi.org/10.3732/ajb.1000298

Blasco, J., Aleixos, N., Cubero, S., Juste, F., Gómez-Sanchis, J., Alegre, V., & Moltó, E. (2009). Computer vision developments for the automatic inspection of fresh and processed fruits. *First International Workshop on Computer Image Analysis in Agriculture*, 21–24.

Cho, C., Kim, J., Kim, J., Lee, S., & Kim, K. (2016). Detecting for high speed flying object using image processing on target place. *Cluster Computing*, *19*(1), 285–292. http://doi.org/10.1007/s10586-015-0525-x

Chowdhury, T., Alam, S., Hasan, M. A., & Khan, I. (2013). Vegetables detection from the glossary shop for the blind . *IOSR Journal of Electrical and Electronics Engineering*, *8*(3), 43–53.

Dorj, U. O., Lee, K. K., & Lee, M. (2013). A computer vision algorithm for tangerine yield estimation. *International Journal of Bio-Science and Bio-Technology*, *5*(5), 101–110. http://doi.org/10.14257/ijbsbt.2013.5.5.11

Duan, F., Wang, Y., Liu, H., & Li, Y. (2007). A machine vision inspector for beer bottle. *Engineering Applications of Artificial Intelligence*, *20*, 1013–1021. http://doi.org/10.1016/j.engappai.2006.12.008

González, M., Villamizar, J., & Lopez, J. (2014). LIQUID LEVEL CONTROL OF COCA-COLA BOTTLES USING AN AUTOMATED SYSTEM. In *CONIELECOMP 2014 - 24th International Conference on Electronics, Communications and Computers* (pp. 148–154).

Jeet Sharma, D., & Kaur, J. (2014). Automatic Apple Harvesting Using Computer Vision Based On Shape & Colour-Based Analysis and Object Positioning. *International Journal of Science, Engineering and Technology Research*, *3*(9), 2278–2281.

Jeon, G. (2013). Measuring and Comparison of Edge Detectors in Color Spaces. *International Journal of Control and Automation*, *6*(5), 21–30.

Sun, D. (2000). Inspecting pizza topping percentage and distribution by a computer vision method. *Journal of Food Engineering*, *44*, 245–249.

Suzuki, S., & Abe, K. (1985). Topological Structural Analysis of Digitized Binary Images by Border Following. *Computer Vision, Graphics, and Image Processing*, *46*, 32–46.