Flower detection using features

Computer Vision, UPC

Xavier Martín Ballesteros and Adrià Cabeza Sant'Anna
UNIVERSITAT POLITÈCNICA DE CATALUNYA

June 18, 2019

Contents

1 Introduction

The aim of this assignment is to classify 12 different types of flowers using feature extraction. The different flower types are the following:

- BotodOr
- Crocus
- Fadrins
- Gerbera
- Hemerocallis
- Narcis
- Buixol
- DentdeLleo
- Fritillaria
- Girasol
- Lliri
- Viola

We have first implemented several ways to extract features that we believed could be key in order to classify correctly the flowers, we tested them, and using different classifiers (decisions trees, SVM or random forest) have tested the accuracy, specificity or sensitivity.

2 Descriptors

In the following sections we will introduce different descriptors we have used to find those valuable features.

2.1 Compactness

One of the first descriptors we came up with when observing the different species of flowers was the **compactness**, that is the ratio of the perimeter to the area of the region. Since the shape and the size of each flower really varies, we think it can be a good discriminator.

The measure takes a minimum value of 1 for a circle. However, objects which have an elliptical shape or a boundary that is irregular rather than smooth, will decrease the measure. On the other hand, objects that have complicated, irregular boundaries will have larger compactness.

To implement it we do need two things: **perimeter**(P) and **area**(A).

$$C = \frac{(P^2/Area)}{4 * \pi}$$

To get the area we have used Matlab's **regionprops** utility, and to get the perimeter we have made **an erosion and a substraction with the original image**. Then, we have counted the number of pixels in white.

In this example, we compare a *Hemerocallis* and a *Botó d'Or* flowers. As the *Botó d'Or* image is more similar to a circle than the *Hemerocallis* one, we should have smaller values for it.

(a) Compactness: **28.10**

(b) Compactness: **0.78**

Figure 1: Comparison of compactness

2.2 Color

Even though there are some types of flowers that can have different colors, most of them show always the same one or two colors. Thus, the color is really important in order to distinguish one specie from another. To do it we have used the Mathworks utiliy: **image2palette [1] which uses k-means color clustering**. We executed the k-means with 4 clusters (one is for the black, which is always present) to get the most important colors.

Moreover, we have also the percentage of pixels that belong to each cluster. Consequently, we will not only use the cluster colors as a feature but also its percentages. We have to remark that before adding the percentage feature to the feature vector, we delete the black percentage and then normalize the other three values.

This is an example with a *Viola* picture:

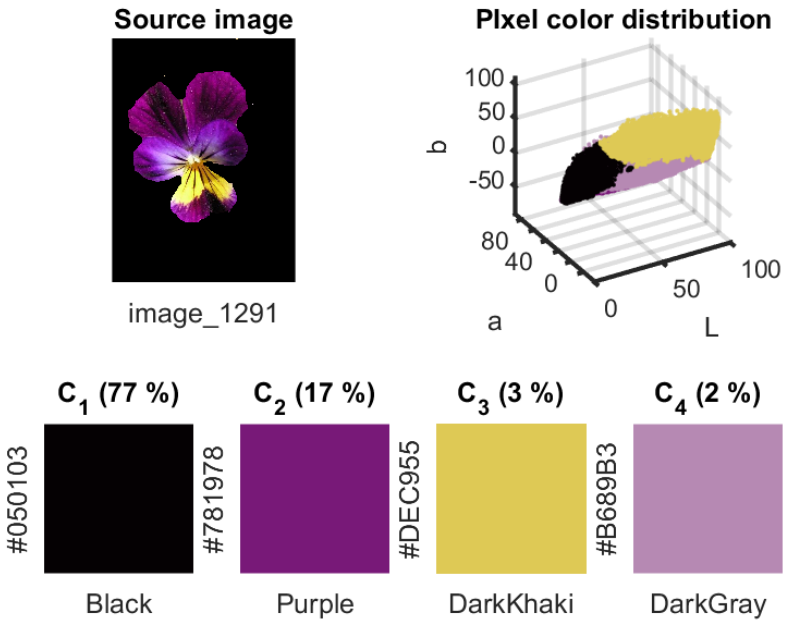


Figure 2: Example of our color feature extraction

2.3 Number of petals

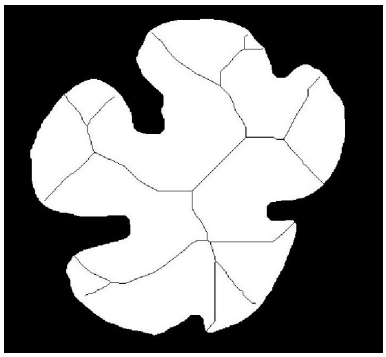
This descriptor was one of the most difficult to think of. It can be done in several ways so we first had to choose in which way we wanted to attack the problem. In our case we decided to **skeletonize the flower**.

To do so we used **bwmorph(image, 'skel', inf)** from Matlab. This creates the skeleton iterating as many times as necessary in order to not see any change between iterations (until it converges). Then we observed that sometimes the skeleton did not touch the contourn of the flower so we could not really count the petals. To solve it, we applied an small erosion using a disk of radius 4 as the structural element. The value of the disk structure was chosen based on several trials.

Finally, we use **bwlabel(image, X-connected)** to label the image in order to obtain the number of petals. In our case, we have computed it using a value of 4 as the connectivity.

In this example using *Boto d'Or* we get as a result 5 petals:

(a) Original

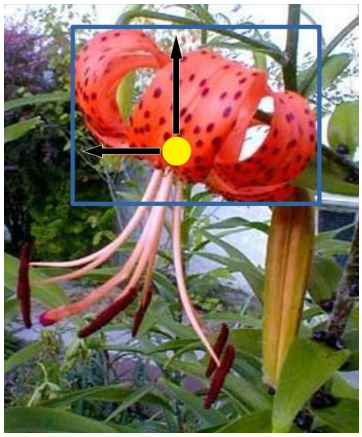


(b) Skez of the image

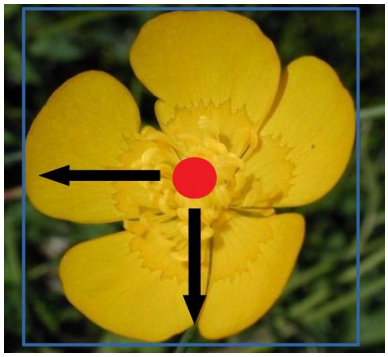
2.4 Relative position of the centroid

Just evaluating the position of the centroid would have been useless, it is not a feature of the flower per se. It can vary depending on the angle of the photo or the distance. However, if we calculate the relative position with respect to the flower's bounding box, we can get really powerful information.

This two flowers, for example, should have really different values for this descriptor and it could be key in order to differenciate them in our classifier:



(a) Centroid and bounding box of a *Hemerocallis*



(b) Centroid and bounding box of a *Girasol*

The way our descriptor is implemented is the following: we get the first and last pixels of X and Y axis in order to create the **bounding box**, then using **regions props** we get the **centroid of the flower** and finally we calculate the **relative position** of the centroid over the bounding box.

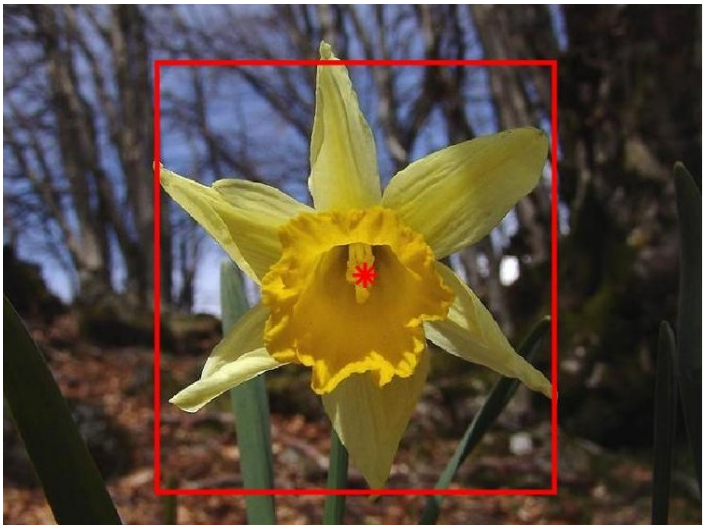
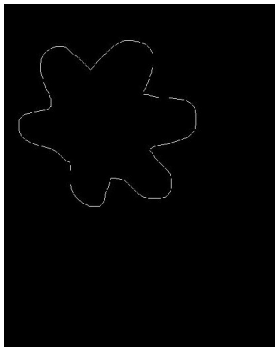


Figure 5: Example of centroid and bounding box

2.5 Fourier descriptor: Shape

It is known that we can use the Fourier transform to analyse and characterize the shape of a boundary. Since there are some flowers like the *Hemerocallis* or the *Girasol* that have a very specific shape, we believe that **the shape** can be a great descriptor. We have only got the first and last **N** values of the vector as features, where **N** marks the level of detail we want of the shape.

(a) Original



(b) Shape

Figure 6: Fourier descriptors

2.6 Hogs form: Orientation

Previously we have introduced a way to describe information about the shape. However, there are other ways to describe it.
As we wanted to have as proper features as possible, we have also implemented HOGs (histogram of oriented gradients), which basically describes **the shapes as the distribution of intensity gradient and edge directions**.

To do it, we have computed the first derivatives of x and y to obtain the gradient. Then, we have computed the histogram of the gradients using the number of bins specified in the input. In our case, we have used 16 bins.

(a) Original

(b) Hog features

Figure 7: HOG descriptors

3 Classifiers

Once we have obtained the feature matrix, where each column represent a feature and each row represent an image, we have used the **Classification Learner App** from Matlab to see the accuracy we get with different types of classifiers. For the classifiers, we have chosen the **Cross-Validation** method with 50 folds to examine the prefictive accuracy of the fitted models.

The Cross-Validation method gives a good estimate of the predictive accuracy of the final model trained with all the data. It is very recommended for small datasets, which is our case. It partitions the dataset into k disjoint folds. For each of them it trains the model and finally it calculates the average test error over all folds.

We have used a Random Forest as our classifier. To obtain it, we have used **fitcensemble** with Method *Bag*, which stands for "bootstrap aggregation", and Tree as Learner. It has as parameters the number of learners (trees) and the maximum number of splits per tree.

The results with this App can be found in Section 7.

4 Experiments

4.1 Feature variables

The first thing we had to do was to obtain the characteristics that seem suitable for describing each type of flower (see Section 2 to see what features we used). Nevertheless, all features (except the compactness and the bounding box) have some variables that need to be fitted.

In this first experiment we have focused on giving the best possible values to those variables. To do it, we have worked on each feature "isolated" from the rest so that the predictions could only vary because of the change of the feature we were working at. We have started by setting a random value two times to a variable and comparing the results. Afterwards, we set again another value (greater or lower depending on the results obtained) and compared the result with the best result of the previous two results. This has been done until the results have been very similar. Thus, the value for the variable has been the one that has given us the best result.

For example, in the number of petals we have been modifying the radius of the disk and observing the number of petals from the output. Depending on that value, we have increased/decreased the radius.

Feature	Variable
Color	nClusters = 4
Compactness	-
Number of petals	SE radius = 4
Fourier	N = 6
HOG	bins = 16
Bounding box	Take contour = NO

Even though we though that the contour would be crucial for the bounding box (especially for the *Hemerocallis*) we have seen that we got worse results when using also the contour. We concluded that this happens due to the fact that some segmented images have contour in its borders. This means that there is contour in places where there is no flower.

4.2 The dataset and experimental procedure for the classification

We have used a dataset consisting of 681 images divided into 12 flower classes. Each class consists of between 46 and 75 images. The dataset is divided into a training set and a validation set. The training set consist of the 80% of images per class (539 in total). The validation set consist of the remaining 20% of images (142 in total). Figure 8 shows the distribution of the number of images over all the classes.

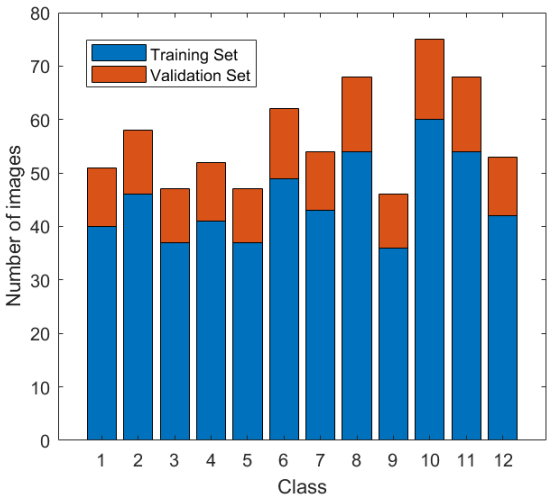


Figure 8: The distribution of the number of images over the 103 classes.

We have trained all the classifiers with the training set and we have selected the one with the best accuracy. Then, we have used it to classify the images from the validation set. We have computed the accuracy for each class. The final performance is the classification averaged over all classes (not over all images).

4.3 Segmentation

Some of the pictures that were given did not have a proper segmentation. Moreover, if we wanted to use different pictures from another sources, we needed a nice and automated way to segmented them since most of our descriptors use the segmentation.

To do so we tried several methods because segmentation can be really difficult depending on the image. Some of the methods we tried were **Otsu**, **k-means with 3 or 4 clusters**, **superpixels and region growing** or **fast marching**. We also considered another type of algorithms called **graph-based segmentation** but they are not easy to automate so we discarted them. In the following examples, you can see some of the bad segmentations those methods gave us:

(a) Original

(b) Segmentation using k-means

(a) Segmentation using Fast Marching

(b) Segmentation using Otsu

Figure 10: Examples of bad segmentation with different methods

Finally, we did realize that it was nearly impossible to get a perfect segmentation method, it really depends on the image; so we decided to use the **active contour** method which seems to get us the better results on average. This method segments an image into foreground and background using the active contour algorithm, also called *snakes*. During an interative process, those *snakes* move to find object boundaries.

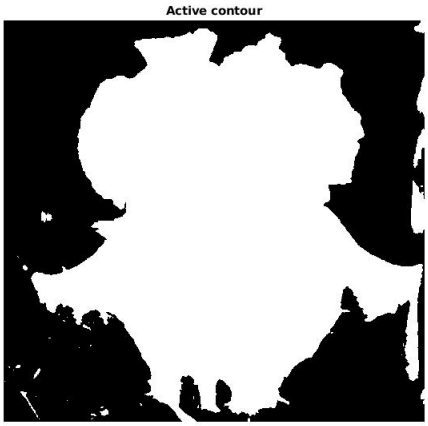


Figure 11: Segmentation using active contour method

Furthermore, we also saw another method [7] that used the **superpixel** technique and then, apply the **region growing** method. The results are very good, even with difficult images such as the *Hemerocallis* ones. However, the computational time is very high, so we decided not to use it.

4.4 Significance of our features

We have already introduced the features we are using. However, it is important to see which are the most important ones and how they are actually able to distinct different types of flowers. To do so we have used the **predictorImportance** utility from Matlab which computes the importance of each predictor based on the splits and the number of branch nodes; and the **Classification Learner App** which is able to gives us the relationship between two features and plot the classes.

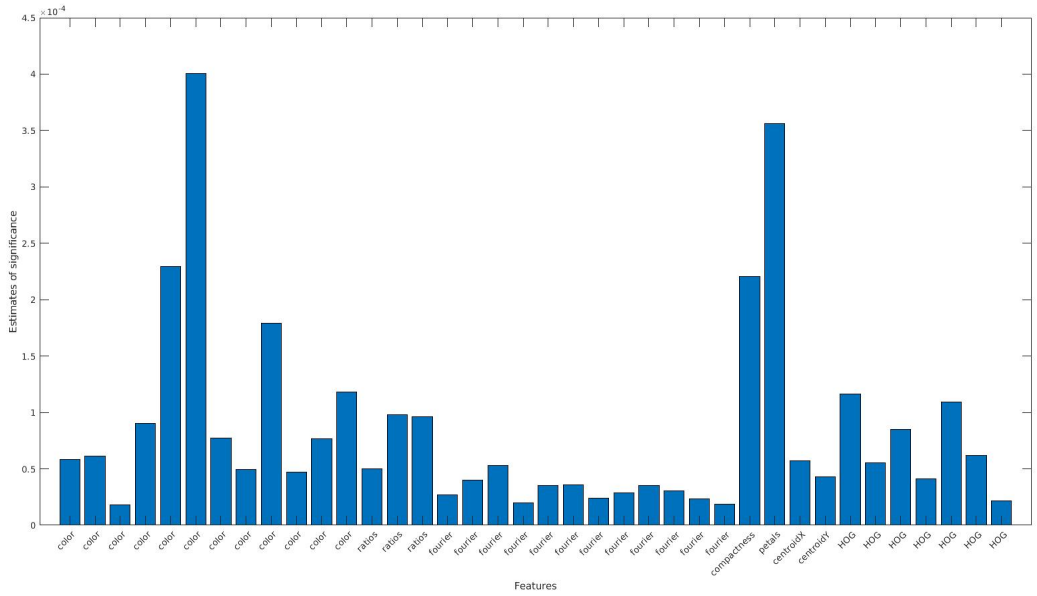
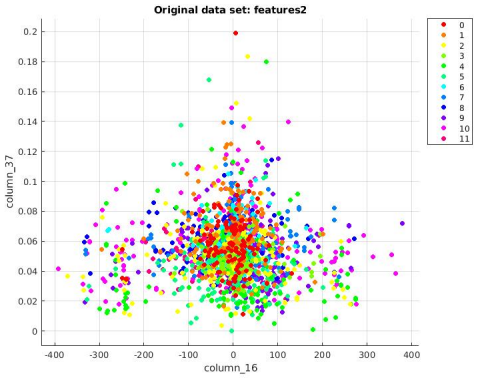


Figure 12: Significance of our features

Although it was expected, we have confirmed that our most valuable features are the color and the number of petals since those features vary a lot depending on each type of flower.

Also, we were a bit surprised about the tiny significance of the fourier descriptor or the relative positions of the centroids in our classifier. To confirm it we used the Classification Learner to see some of its relationships between other features. Using those features we did not find any combination that gave us a good classification distribution. Here we can see some examples:



(a) HOG-Fourier Relationship

(b) CentroidX-CentroidY Relationship

That is why we conclude that those features are not as good as the others to classify. If they were good, we should have found some distribution looking like this:

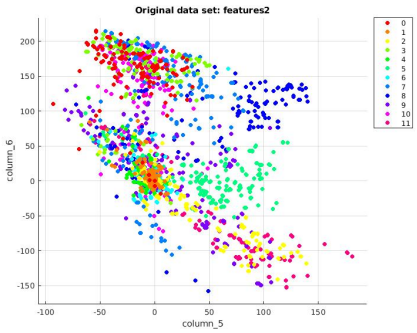


Figure 14: Colors relationship

5 The *zero* class

In order to increase the precision of our model, we have added a new class called *zero*. This class represents all the flowers that are not from any of the 12 species that we are observing.

To implement it we have written a script that **crawls google images and downloads photos of other species of flowers**. This will be very useful to create our dataset of *zero* flowers. Morover, since these images are not segmented, we have looked for another source of flower pictures. In this case we have used **the Flowers Dataset from the Visual Geometry Group** (University of Oxford) [3]. This approach will help us to save time because we will have some flowers segmented by us (reviewed manually afterwards) and some already segmented (which we suppose are already correct).

Our approach will consists in not training our classifier with that photos: we will assign a picture to that class if the confidence is less than a threshold. To test if actually works we will use the already mentioned test.

Some of the flowers that can be seen in our *zero* flower dataset are:

(a) Azalea

(b) Roses

(a) Gazania

(b) Tulip

Figure 16: Zero class flowers

6 Data augmentation

In order **to strenghten our descriptors** we have also used the following public repository: *Albumentation: fast image augmentation library and easy to use wrapper around other libraries* [5], which proporcionates facilities to augment the dataset including several transformations like: flipping, blurring, RGB Shifting, Random contrast, Random brightness, etc...

To decide which transformations we wanted to apply, we looked first to our descriptors and the importance of each feature. For example, we discarted the Channel Shuffle because we believe that the color is very important for our implementation. Then, based on trial and error, we picked some of them which gave us an overall better result. Finally we are using random variations of:

- Brightness and Contrast
- Blur
- Rotations
- Flips

For example let's suppose that we are working with this *Boto d'Or* image:

Figure 17: Original picture

The script we have implemented would apply the following transformations:

(a) Blur transformation

(b) Horizontal Flip transformation

(a) Brightness and Contrast transformation

(b) Scale transformation

Figure 19: Data augmentation transformations

7 Results

Before starting this section, we must remark that the training and validation sets have always been the same to be able to replicate the same training for the same model. The unique randomness is the one generated by the Cross-Validation method, with its 50 splits of the data.

The first time we used the **Classification Learner App** was without using the Data Augmentation technique. We trained All model types to see which one gave us the best accuracy. The results are shown below. As you can see, the best model is **Bagged Trees**. So from that moment on we have only used this model.

Model	Accuracy
Fine Tree	72.5
Medium Tree	71.2
Coarse Tree	39
Linear Discriminant	75.3
Linear SVM	78.1
Quadratic SVM	79.8
Cubic SVM	79.6
Fine Gaussian SVM	16.3
Medium Gaussian SVM	75.3
Coarse Gaussian SVM	64
Fine KNN	68.5
Medium KNN	67.7
Coarse KNN	49.2
Cosine KNN	69.4
Cubic KNN	64.9
Weighted KNN	70.9
Boosted Trees	76.3
Bagged Trees	**84.2**
Subspace Discriminant	74.8
Subspace KNN	30.4
RUSBoosted Trees	76.1

Then, we used the validation set to see the accuracy we could get when using images that the classifier had never seen. We got an accuracy of 74.6479%.

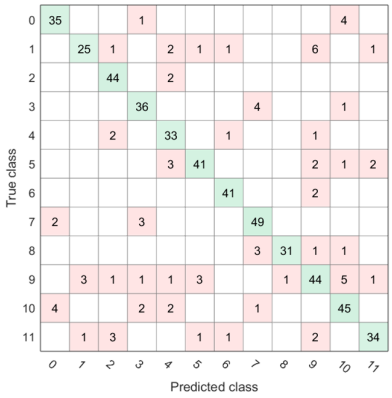


Figure 20: Confusion matrix of the **Classification Learner App** without using the Data Augmentation images.

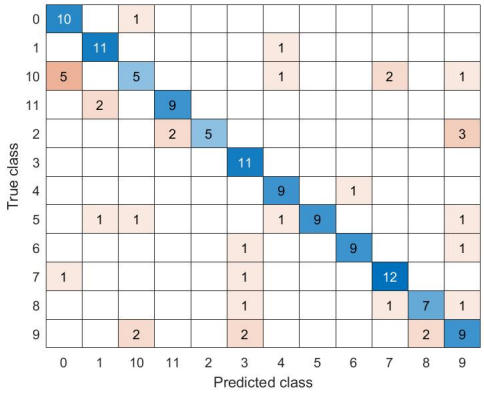


Figure 21: Confusion matrix of the validation set without using the Data Augmentation images.

Once we saw the accuracy we got using only an 80% of the given images, we wanted to try to use more data, the one generated with the Data Augmentation technique. For each image of the training set, we got 4 more images (explained in the previous section). With this, we passed from 539 images from training to 2704. We trained again the Bagged Trees model in the Classification Learner App and we got a better result: 90% of accuracy. With this new trained model, we got an accuracy of 82.3944% with the validation set. Hence, the Data Augmentation method has helped to improve a lot the performance of the classifier.

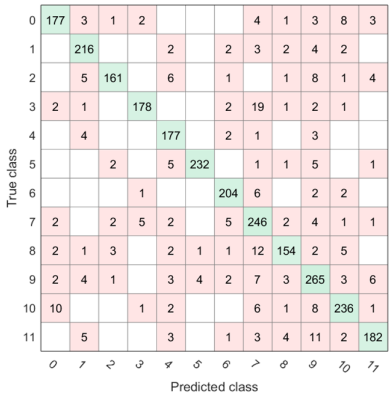


Figure 22: Confusion matrix of the **Classification Learner App** using the Data Augmentation images.

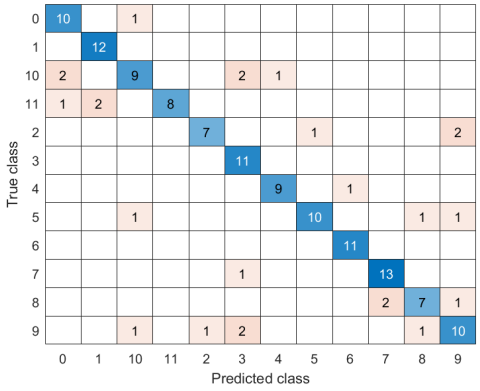


Figure 23: Confusion matrix of the validation set using the Data Augmentation images.

We have also tried to introduce the zero class (which has label 40). However, we have not found a good threshold that minimizes the error for all the classes. The bigger the threshold, the lower the percentages of the 12 classes and the higher the percentage of the zero class, and vice versa. With a threshold of 0.42 we get an accuracy of 53.5316%, 66.1972% only with the 12 classes.

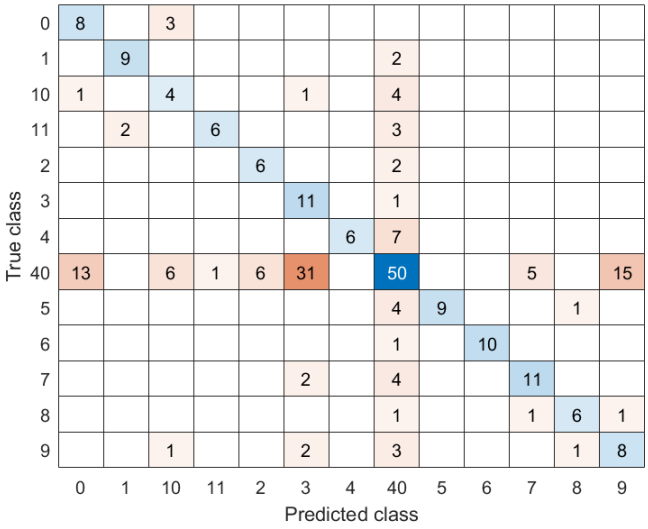


Figure 24: Confusion matrix with the zero class (label 40).

8 List of functions and libraries that have used

All the following functions used have been implemented by ourselves, except from **image2palette** and **the albumentations library**.

- **Bounding Box**: function that finds the relative position of the centroid from the bounding box. It uses the matlab utility **regionprops** to get the centroid.

- **getCompactness**: function that calculates the compactness of a flower. It uses the matlab utility **regionprops** to get the area. To get the perimeter we have made an erosion and then a substraction.
- **getForma**: function that gets the shape Fourier descriptor.
- **getHOG**: function that gets the HOG features.
- **getNumberPetals**: function that gets the number of petals using the skeleton. It uses the matlab utility **bwmorph** for the skeletonization.
- **image2palette** [1]: function extracted from Matlab File Exchange. It parses the image into pixels in L*a*b space, and returns the major components clustered by k-means method. The L*a*b color space (defined by the International Commission on Illumination) expresses color as three values: L* for the lightness from black (0) to white (100), a* from green (-) to red (+), and b* from blue (-) to yellow (+). It was designed so that the same amout of numerical change in the values corresponds to the same amount of visually perceived change. We use it to get the 3 main colors of the image and the percentage of pixels in each cluster.
- **getImage**: function that gets us the image and its segmentation.
- **segment**: function that segments an image using the active countour method. It uses the matlab utility **activecontour**.
- **Albumentations library** [5] for data augmentation issues.
- **Google images download library** [6] to download several pictures of the *zero* class.

References

[1] Han. H. (2018). *image2palette: Simple K-means color clustering* [online] Code available at: https://es.mathworks.com/matlabcentral/fileexchange/69538-image2palette-simple-k-means-color-clustering [Accessed 28 April 2019].

[2] *Image Segmentation with MATLAB* [online]. Available at : https://www.mathworks.com/discovery/image-segmentation.html [Accessed 28 May 2019]

[3] Nilsback, M-E. and Zisserman, A. (2008) *Automated Flower Classification over a Large Number of Classes* [online] Available at:http://www.robots.ox.ac.uk/ vgg/publications/papers/nilsback08.pdf [Accessed 26 May 2019]

[4] M.A. Wirth.(2004) *Shape Analysis & Measurement* [online] Available at: http://www.cyto.purdue.edu/cdroms/micro2/content/education/wirth10.pdf [Accessed 24 May 2019].

[5] Buslaev A.(2018). Based on *Albumentations: fast and flexible image augmentations* [online]. Paper available at:https://arxiv.org/abs/1809.06839. Code available at: https://github.com/albu/albumentations [Accessed 3 June 2019].

[6] Vasa H. (2019). *Python Script to download hundreds of images from 'Google Images'.* [online] Code available at: https://github.com/hardikvasa/google-images-download [Accessed 4 June 2019].

[7] balcilar. (2019). *Color-Image-Segmentation-Using-Region-Growing-and-Region-Merging* [online] Code available at: https://github.com/balcilar/Color-Image-Segmentation-Using-Region-Growing-and-Region-Merging [Accessed 6 June 2019].