Automated bee wings classification : the Deepwings project

Théo Bodrito

December 21, 2018

Abstract

Bees have been dying off at an extraordinary high rate, and no one is quite sure why. To investigate this issue, researchers need a quick and reliable way to identify bee species. This document introduces the Deepwings project, which compares deep learning and features extraction approaches to classify bee species from wings pictures.

1 Introduction

About 35% of the world's crops and 80% of the world's flowers are pollinated by bees! Most bees that are farmed for pollination are European Honeybees, whose physique and behavior makes them very efficient. But native bees can share in the pollination effort, and some may be more efficient.

Recently, honeybees have been dying off at an extraordinarily high rate, and no one is quite sure why. Researchers call it Colony Collapse Disorder, and the problem is serious because of our dependence on honeybees to pollinate food crops. While many are researching the cause of Colony Collapse Disorder, research into native bees may uncover more productive alternatives to the European honeybee. Finding the cause and potential native alternatives involve tracking wild bee populations and habits.

There are many species of bees, more than 500 in Wisconsin alone, but it's not easy to tell which species an individual belongs to.

While bee species identification is essential to research, identifying the species of a bee can be expensive and time-consuming. Since there are few experts who can reliably distinguish many different species, bees must be captured and sent to an expert, which may take several months. Bee research can be excruciatingly slow.

Rather than capturing, killing, and sending the bees off to be catalogued, imagine an iPhone app that lets graduate students and researchers identify bees in the field. One could simply take a photo of a bee and instantly record its species and location. Research could be conducted much faster, and identified bees could be released back to nature.

2 Previous work in bee wings identification

We relied mostly on Christopher Hall's work for features extraction: *An Automated Approach to Bee Identification from Wing Venation* (2011).

3 Dataset

The initial dataset was composed of 1193 images, but we cleaned it so as to have at least 20 images per species. So the dataset we used consists of 1135 images belonging to different genera, and each genus is divided into different species as follows :

genus	species	Number of images
bombus	impatiens	65
	griseocolis	26
	...	...
osmia	ribifloris	61
	lignaria	58
	...	...
agapostemon	texanus	107
...	...	...
5 genera	21 species	1135 images

Figure 1: Dataset structure

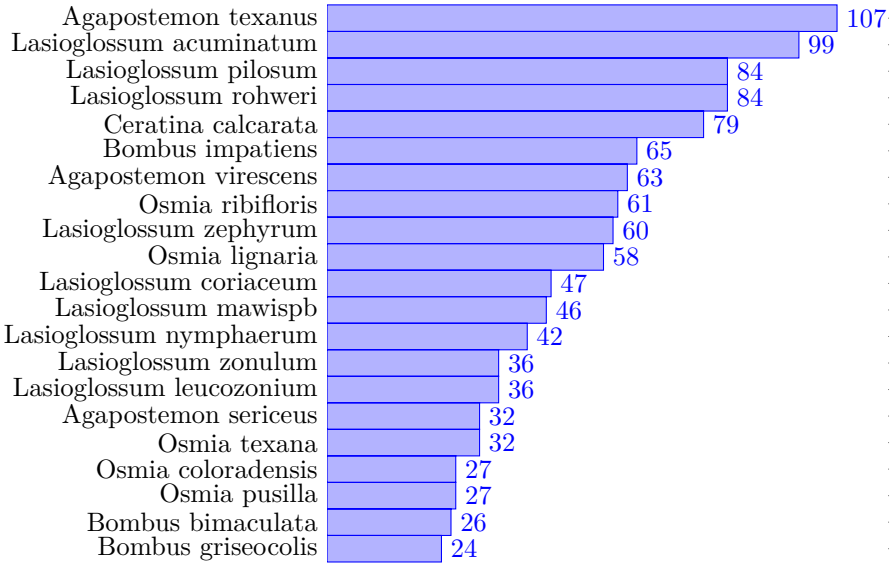


Figure 2: Number of images per species

4 First method : CNN

Convolutional Neural Networks are best suited for image classification. We compared two different architectures: VGG16 and DenseNet121.

4.1 VGG16

This model came out in 2014. It is a very deep and very classical CNN base folowed by 3 Dense layers. It achieved great performance at the time, but one of its biggest flaw is its size: it can be very difficult to train it from scratch.

Layer (type)	Output Shape	Param #
input_1 (InputLayer)	(None, 150, 150, 3)	0
block1_conv1 (Conv2D)	(None, 150, 150, 64)	1792
block1_conv2 (Conv2D)	(None, 150, 150, 64)	36928
block1_pool (MaxPooling2D)	(None, 75, 75, 64)	0
block2_conv1 (Conv2D)	(None, 75, 75, 128)	73856
block2_conv2 (Conv2D)	(None, 75, 75, 128)	147584
block2_pool (MaxPooling2D)	(None, 37, 37, 128)	0
block3_conv1 (Conv2D)	(None, 37, 37, 256)	295168
block3_conv2 (Conv2D)	(None, 37, 37, 256)	590080
block3_conv3 (Conv2D)	(None, 37, 37, 256)	590080
block3_pool (MaxPooling2D)	(None, 18, 18, 256)	0
block4_conv1 (Conv2D)	(None, 18, 18, 512)	1180160
block4_conv2 (Conv2D)	(None, 18, 18, 512)	2359808
block4_conv3 (Conv2D)	(None, 18, 18, 512)	2359808
block4_pool (MaxPooling2D)	(None, 9, 9, 512)	0
block5_conv1 (Conv2D)	(None, 9, 9, 512)	2359808
block5_conv2 (Conv2D)	(None, 9, 9, 512)	2359808
block5_conv3 (Conv2D)	(None, 9, 9, 512)	2359808
block5_pool (MaxPooling2D)	(None, 4, 4, 512)	0

Total params: 14,714,688
Trainable params: 7,079,424
Non-trainable params: 7,635,264

Figure 3: Architecture base VGG16

Our dataset was not big enough, that's why we had to use *transfer learning* : the idea is to use a pretrained convolutional base (trained on imagenet). The first step is to train the top layers (the dense ones) while the weigths of the base remain frozen. Then, we defreeze the top 3 layers of the convolutional base, to fit our dataset more precisely. This last step is called *fine tuning*.

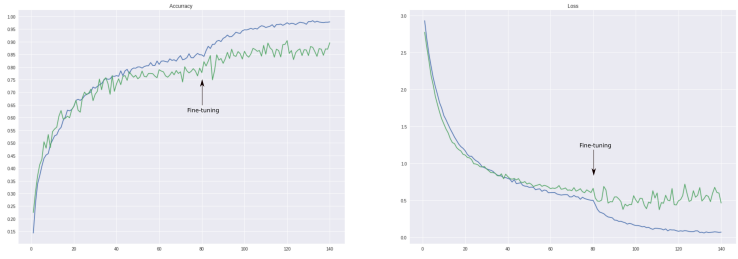


Figure 4: VGG16 training with fine-tuning

4.2 DensetNet121

DensNets came out in 2016. This model is lighter than VGG16 and faster to train than VGG16 thanks to shorter connections between layers: each layer is connected to every other layer in a feed-forward fashion.

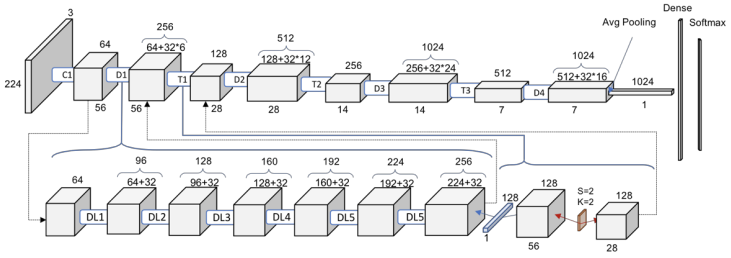


Figure 5: Architecture DenseNet121

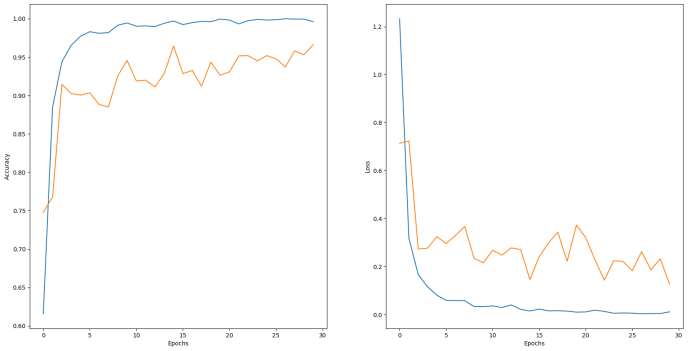


Figure 6: DenseNet121 training (without fine-tuning)

As you can see on the curves, DenseNet121 reaches better accuracy than VGG16 on the same dataset (95% vs 88%). Furthermore, its light weight make it the perfect CNN model for our project.

5 Second method : Features extraction

5.1 Bee wings cells

Bee wings are divided into several cells :

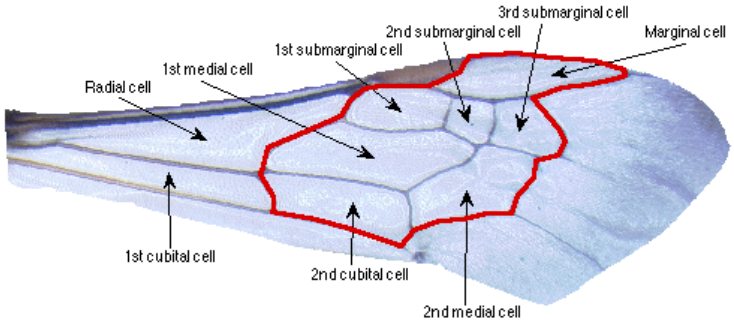


Figure 7: Cells names

The cells we are interested in are circled in red. Those cells were chosen because they are entirely visible on all images from the dataset. Nevertheless we need to be careful because the 3rd submarginal cell is not always separated from the 2nd submarginal cell, it depends on the species. We have to keep that in mind for the next steps.

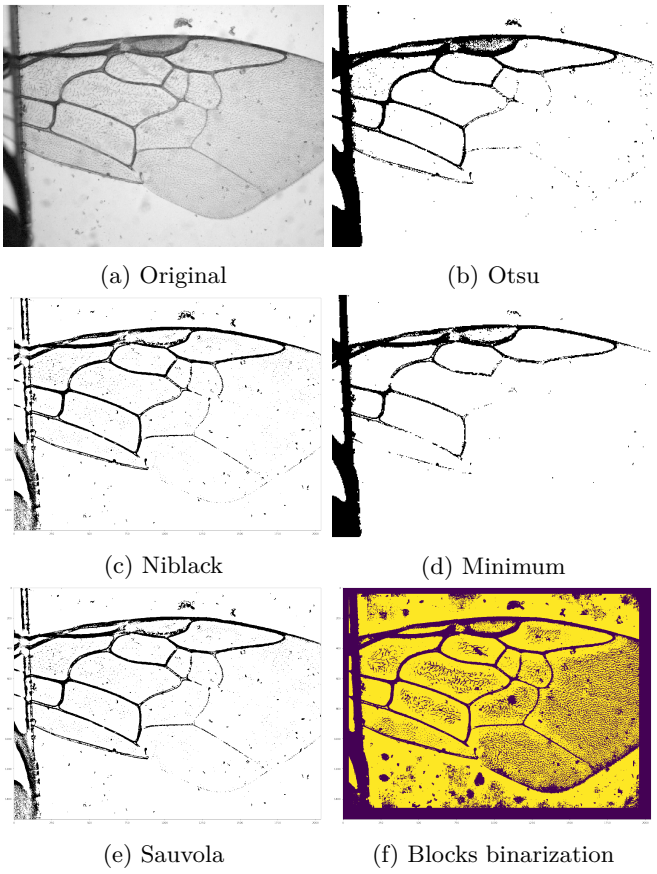
First of all, we need to detect those cells, then we have to identify them and finally, extract features from them.

5.2 Regions detection

5.2.1 Binarizations

The first step is to binarize the image so as to detect the veins and cells clearly. To do that, we compute a grayscale image and then we compare the value of each pixel to a threshold. If the value is above the threshold, it becomes "True" or white in the binarized image, otherwise it turns black or "False". This threshold can be global :

- Global : unique threshold for all pixels.
- Local : different thresholds for different part of the image. It allows for a more refined binary image, and it helps discarding variation of shades in the input image.



(a) Original (b) Otsu

(c) Niblack (d) Minimum

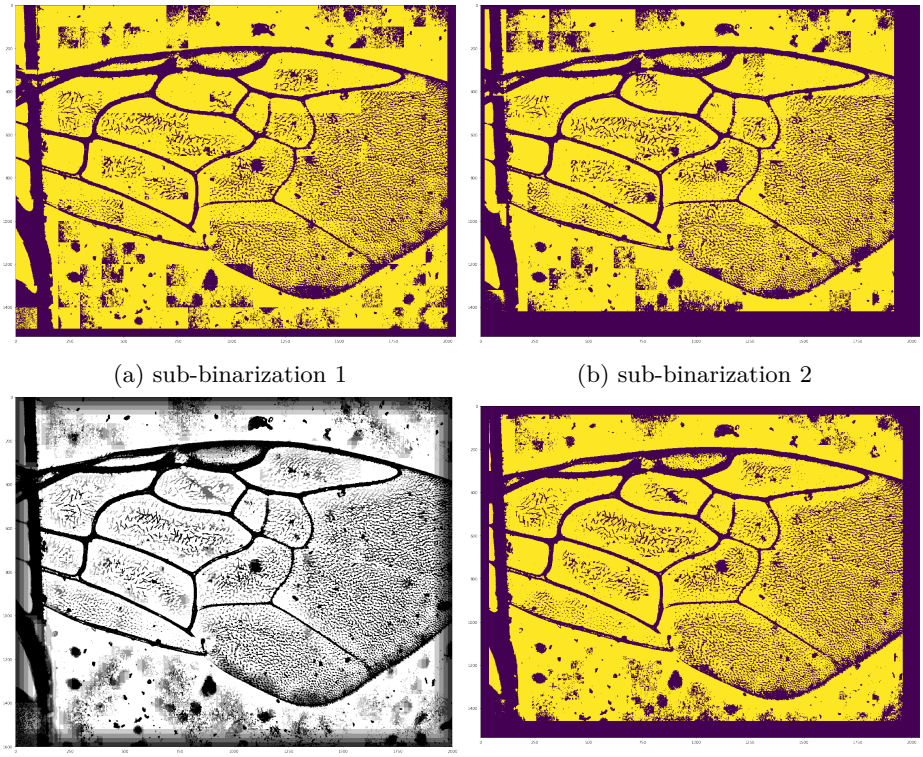
(e) Sauvola (f) Blocks binarization

Figure 8: Different binarizations: global (*b, d*) vs local (*c, e, f*) thresholds

As shown in figure 8 veins are better segmented with local thresholds, and especially with blocks binarization (ref Chris Hall). In other cases, the veins are not complete which will be an issue in the following steps.

5.2.2 Blocks binarization

The idea here is to divide the input image into blocks and binarize each block independently with Otsu binarization to obtain a "sub-binarization". We repeat the process with different overlapping grids, and then sum all the "sub-binarizations" to obtain a grayscale image, to which we apply one last global Otsu binarization. The technique results in the binary image shown in figure 9.



(a) sub-binarization 1

(b) sub-binarization 2

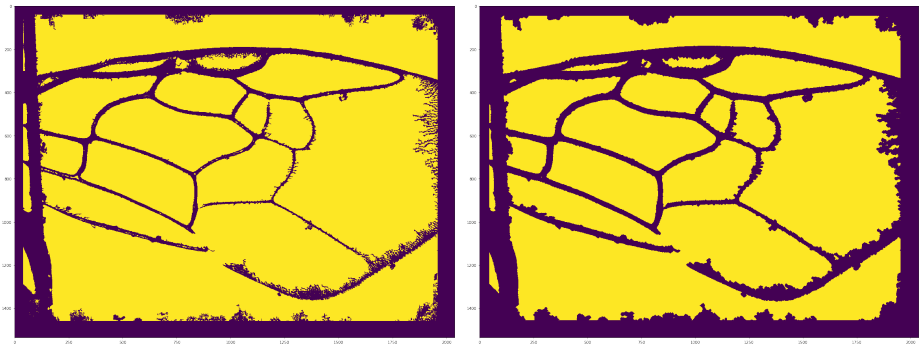
(c) Sum of all sub-binarizations

(d) Final binarized image

Figure 9: Blocks binariation : step by step

The advantage of using overlapping grids is to overcome border effects : taken independently, sub-binarizations 1 and 2 in figure 9 are not sufficient to clear the veins, but when combined, the binary image is much more refined.
At this stage, still having hairs inside the cells is not an issue. They will be removed in the next step.

5.2.3 Post processing



(a) Morphological reconstruction by erosion

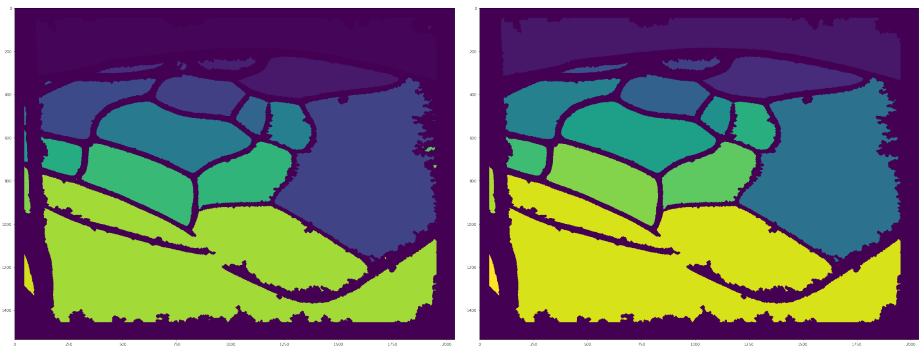
(b) Binary erosion

Figure 10: Post-processing

Now we need to remove the remaining hairs inside the cells. To do that, we use morphological reconstruction by erosion : it removes all black pixels not connected to the border of the frame. This way, veins are preserved and cleared. In order to reinforce veins, we use binary dilation on top of that (see figure 10).

5.2.4 Labels

Now that all veins are detected, we want to separate the different regions in order to detect cells. We attribute labels (ie. integer values) to those regions in order to refer to them.



(a) Labelized image

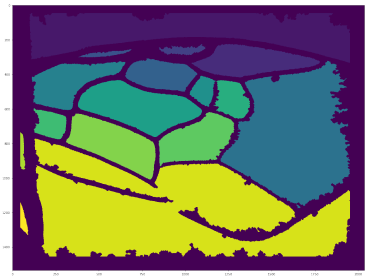
(b) Small regions discarded

Figure 11: Labels

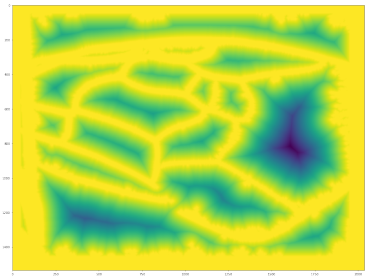
Then we discard small regions as they they might cause trouble in the next stages (see figure 11).

5.2.5 Watershed

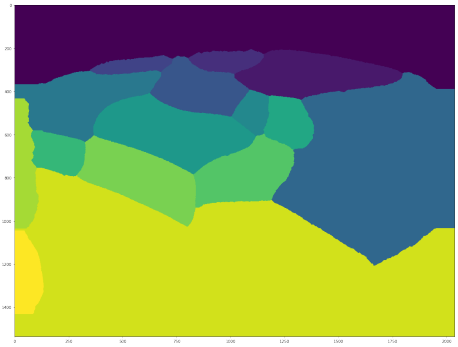
Now we have regions, but their edges are not ivery well defined because of the binary dilation done before. To improve accuracy on the edges, we use watershed segmentation. To do so, we first need to compute a distance map of our binary image : for each pixel of the foreground, we compute the shortest distance to a background pixel.



(a) Labels



(b) Distance map (reversed)



(c) After watershed segmentation

Figure 12: Watershed segmentation

We use both markers (labels) and reversed distance map to compute the watershed segmentation. We have now a much more refined description of our regions. This will be useful to filter them.

5.2.6 Clear borders

As the regions we are interseted in (ie. the cells previously described) are located at the center of the image, we discard all regions in direct contact with the top, left and bottom borders of the frame. We don't remove regions adjacent to the right border because it happens sometimes with the marginal cell, which we wan to keep absolutely.

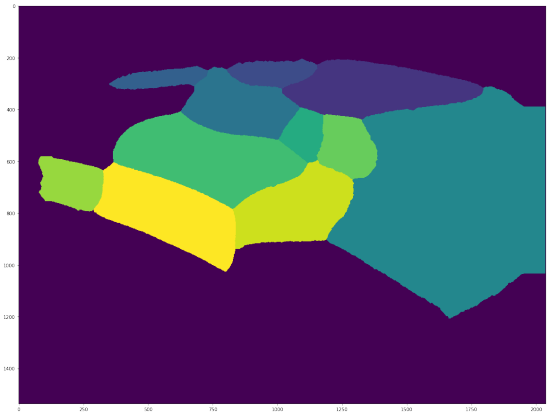


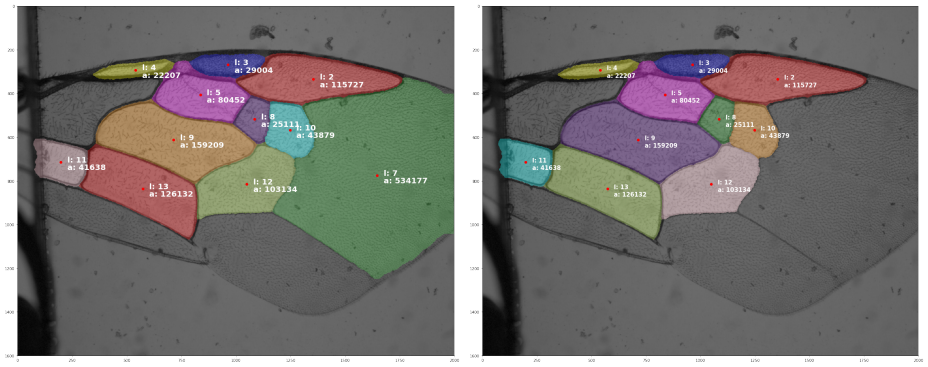
Figure 13: Clear borders

5.3 Regions filtering

Now that we have roughly detected the regiosn of interest, we are going to refine this selections through a set of filters.

5.3.1 Area filter

The first filter discard regions based on their area. It uses a threshold proportionate to the area of the whole image.



(a) Before filtering (b) Area filter

Figure 14: Area filter

5.3.2 Neighbors filter

We determine each region's neighbors, and we discard those having less than 2 neighbors.

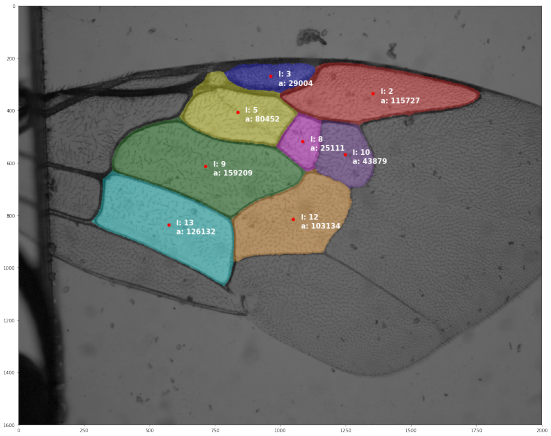


Figure 15: Filter neighbors

5.3.3 Central cell

To continue the precess pf selection, we need a cell of reference that we call "central cell", whiwh corresponds to the first submarginal cell. To find it, we

compute a score for each region : $score = area * neighbors$ For better accuracy, we also dismiss regions that are too eccentric to be the central cell.

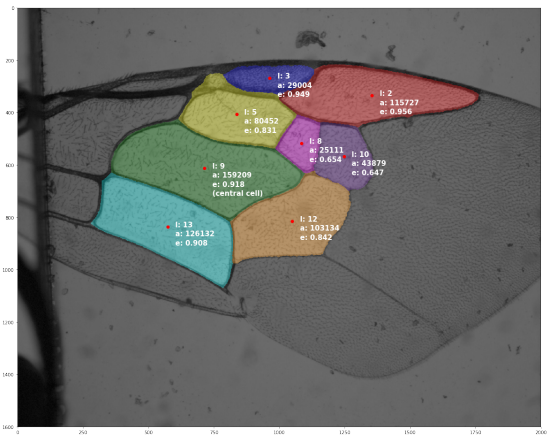
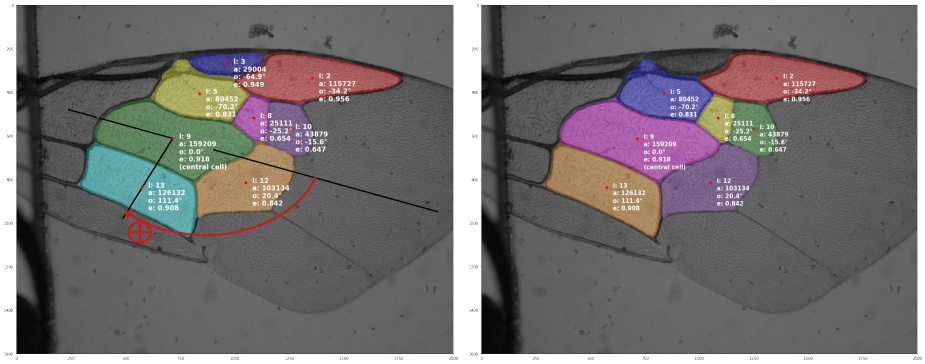


Figure 16: Central cell found

As shown on figure 16, the central cell as been correctly found and labeled.

5.3.4 Angles filters



(a) Computing angles (b) After angle filtering

Figure 17: Angles filter

Now that we have a point of reference, we can determine angles of other region-swith respect to the central cell. We first compute the orientation of the central cell, then we can find the angle between : the centroid of the focused region, the central cell's centroid, and the central cell's orientation.

Then we will discard regions based on their angle and their proximity to the central cell :

- Direct neighbors of the central must have an angle ranging from -80° to 150°
- Degree 2 neighbors must have an angle ranginf from -40° to 5°

- Other regions are discarded

5.4 Cells identification

Now we should have 6 or 7 regions remaining, depending on the species. If not, the image is discarded as invalid. We use the angles and their proximity to the central cell to label them, the result is shown in figure 18.

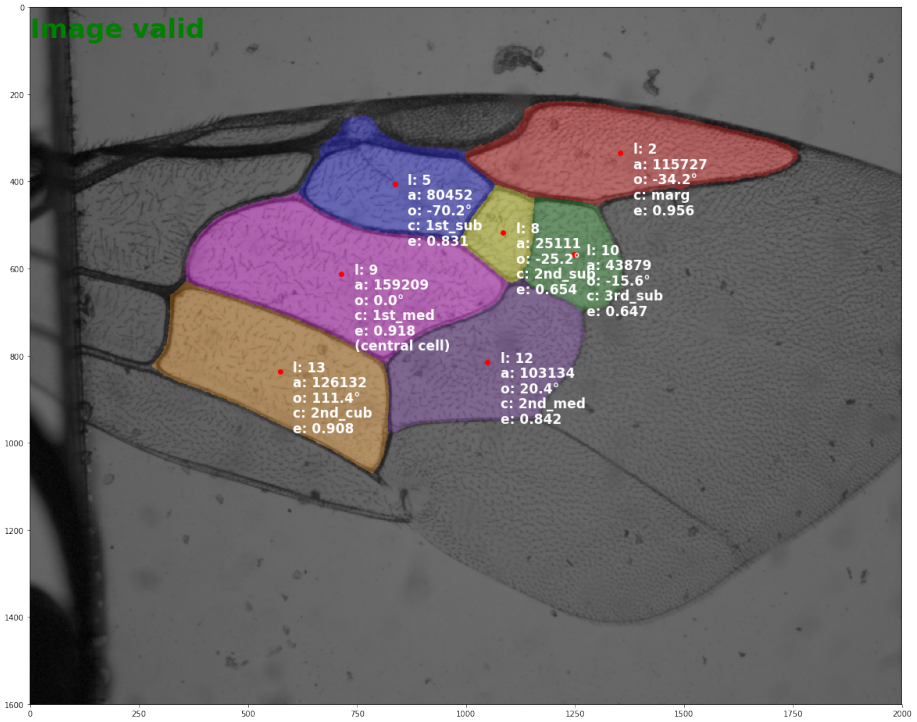


Figure 18: Identified regions

5.5 Features extraction

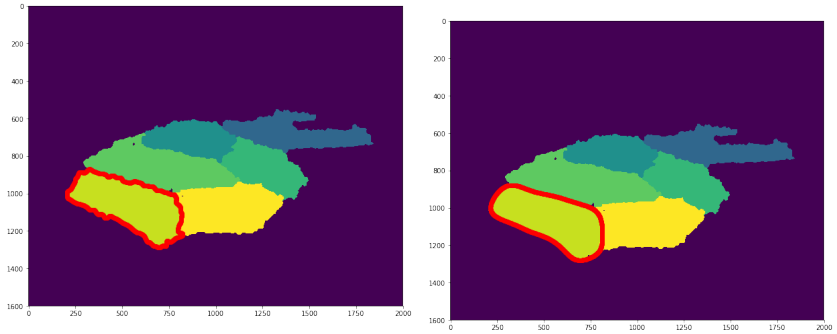
Now that our cells are indentified and labeled, we can extract features from them that will be used for classification.

5.5.1 Fourier descriptors

For each cell, we compute the Fourier descriptors, which contain information about the overall cell's shape. To do so, we first compute the boudnary of each cell using the Moore neighboroog algorithm, and then we use Fourier tranformation to keep only the first coefficients.

Fourier descriptors:

Let $x[m]$ and $y[m]$ be the coordinates of the mth pixel on the boundary of a given 2D shape containing N pixels, a complex number can be formed as $z[m] = x[m] + jy[m]$



(a) Boundary tracing: Moore neighborhood

(b) Inverse Fourier transformation

Figure 19: Extraction of the first 15 Fourier descriptors

$$\forall k \in [0, 1, -1, .. \text{ N-1}, -(\text{N-1})], \quad Z[k] = \frac{1}{N} \sum_{m=0}^{N-1} z[m] e^{-j2\pi mk/N}$$

Inverse transformation:
To reconstruct the signal from the first M Fourier descriptors :

$$\forall m \in [\![0, \text{N-1}]\!], \quad \hat{z}[m] = \sum_{k=-M/2}^{M/2} Z[k] e^{-j2\pi mk/N}$$

5.5.2 Results features extraction

For each cell, we extract the following features:

- area (scaled with the total area of detected cells)
- eccentricity
- angle with respect to the central cell
- first 15 normalized Fourier descriptors

Results of cells detection:

category	Features extraction	Groundtruth
invalid	101 (9%)	0
valid	1034 (91%)	1135 (100%)

We then applied a Random Forest classifier on the valid dataset, which had a 83% accuracy on test set (and 100% on training set).

6 Conclusion

Model	training accuracy	test accuracy
DenseNet121	99%	95%
VGG16	98%	88%
Features extraction + Random Forest (on valid dataset)	100%	83%

Figure 20: Models comparison

We can see the CNN models remain more accurate than features extraction + random forest with out method, at the cost of a heavier model. Furthermore, no features engineering is required with CNN, whichh is a huge advantage in such a project.