Bee Subspecies Classification using Machine Learning Methods over Bee Images

AA - Project 2

Beatriz Costa
Department of Electronics,
Telecommunications,
and Informatics.
University of Aveiro
Email: beatriztcosta@ua.pt

Miguel Carvalho
Department of Electronics,
Telecommunications,
and Informatics.
University of Aveiro
Email: miguel.paz.carvalho@ua.pt

***Abstract*—Pollinators, such as bees, are crucial for our environment's sustainability. Although not all species of bees are nearing extinction or endangered, such a scenario must be avoided. In order to do so, solutions designed to tackle possible threats to their population, invasive species for example, must be developed. The creation of a classifier capable of accurately differentiating bees, and their subspecies, from possible predators, may assist those solutions in preserving the bee population when threatened.**

Keywords: bee, invasive species, Logistic Regression, Machine Learning, Neural Networks, Supervised Learning.

I. Introduction

Responsible for the pollination of nearly 75% of the world's food crops , according to the Intergovernmental Science-Policy Platform on Biodiversity and Ecosystem Services (IPBES), bees, alongside other pollinators, play a major role in the conservation and sustainability of the world's flora and fauna [1]. Unfortunately, according to the International Union for Conservation of Nature (IUCN), from the estimated 800 wild bee species within Europe, seven are classified as critically endangered, further 46 are endangered, 24 are vulnerable, and 101 near threatened. Although this does not mean that the bee population is nearing extinction in its entirety just yet, the mere possibility of such an occurrence would have devastating consequences.

Aside from pesticides and climate change, the presence of invasive species, such as the Asian hornet which preys on bees [2], is another reason for this decline, giving rise to the need of creating effective solutions to resist such threats and preserve the bee populations. We believe that such solutions can benefit from machine learning applications capable of recognizing and differentiating bees from other species in the surrounding environment.

II. State of the Art

The application of machine learning techniques for bees and bee subspecies classification, although not vastly explored, has been done previously, including our previous report as well. Important mentions of significant developments in this area are [3], [4], [5], [6] and [7], which we will now explore.

In the article titled "Image recognition using convolutional neural networks for classification of honey bee subspecies" [3], the authors strategy to identify the multiple subspecies was to employ four different Convolutional Neural Networks (**CNN**) architectures and evaluate their efficiency in discriminating the honey bee subspecies through analysis and classification of wing images. These CNN were trained using a set of 9887 forewing images, belonging to a reference collection containing 7 European subspecies and the Buckfast intraspecific hybrid. It is worth noting that the amount of images per subspecies present in this collection varies between 267 and 2771, being a very unbalanced dataset, possibly making the classifier biased towards certain classes but no mentions were made regarding the balance of this data in the article.

Another important note regarding the mentioned study is that the images used were also resized and padded with white pixels to match the square format expected by the considered neural networks, which, as shown in our previous report, may lead to worse results since the additional padding of white pixels can incorrectly train the network. Regardless, the applied models ResNet 50, MobileNet V2, InceptionV3, and Inception ResNet V2, using implementations from TensorFlow, Keras, and PyTorch, were described as providing a "reasonably good overall performance", reaching accuracy values of 0.99% (by the Inception ResNet V2 and InceptionV3) on classification of a subset of 242 colony samples, performing better than a morphometric approach, which reached an accuracy of 0.86% in the same scenario.

The developments documented in the articles "Assessing the potential for deep learning and computer vision to identify bumble bee species from images" [4] and "A Machine Learning Based Approach to Study Morphological Features of Bees" [5] also mention the usage of deep learning models for the classification of bee species and relevant feature extraction.

In the first [4], from an initial dataset of 120.000 images belonging to 42 species, 89.776 images belonging to 36 species are retained after quality control, which consisted of image cropping and discarding. Since the dataset was considered as

being highly imbalanced by the authors, having much more images from a certain specie than others, the number of images per specie was limited to a maximum of 10.000 to help limit the classification bias associated with imbalanced data sets. It is worth bearing in mind that, although this limit was established, it may not solve the imbalance issue since, if one or more classes have up to 10.000 images, it still represents a considerable percentage from all the 89.776 available images.

Like in the previous article [3], four different deep learning models were tested, the ResNet-101, Wide-ResNet-101, InceptionV3 and MnasNet-A1. Wide-ResNet had the highest test accuracy of 91.7% followed closely by InceptionV3 (91.6%) and ResNet100 (91.3%). MnasNet, however, had relatively low test accuracy of 85.8%. Of the four models tested, InceptionV3 presented a good balance between performance and speed, having the highest precision and nearly matched Wide-ResNet's accuracy and recall while being 39% (2.1 ms) faster. Likewise, InceptionV3 was 5.8 percentage points more accurate than MnasNet while only 0.06 ms slower.

In the following article [5], machine learning methods were applied to extract variants of class activation maps that visually display distinguishing morphological features of bees. The dataset used on this study was composed of 24,695 images from 11 different species of Bumblebee, Honey bee and Carpenter bee from iNaturalist, having 2,245 images of each specie. These images were resized to 224 x 224 pixels and normalised to fit into pre-trained ImageNet models. From this processed dataset, 80% was used for training and 20% for testing purposes of the models.

11 state-of-the-art deep learning models, namely EfficientNet, ResNest101e, CSPDarkNet, TresNet, RexNet, CSPResNet, NoisyStudent, Vision Transformer, LCA-CNN, SpinalNet, and RegNet which were all pretrained with ImageNet for fine-grained image classification tasks. The deep networks were trained using a mini-batch stochastic gradient descent optimiser with a batch size of 32. Learning rate, momentum, and weight decay were kept at 0.01, 0.9, and 0.0001 respectively. A dropout value of 0.2 was also employed to prevent overfitting. The gradients within the best performing model were utilised to obtain a Gradient-weighted Class Activation Mapping (Grad-CAM) which visually indicates the particular region used by the model to classify bee species. Overall, the ResNest and TFefficient Net with noisy student performed better, reaching the highest accuracy for bee classification, 93.66% was obtained with a ResNest101e model, and 94.27% with data augmentation.

Regarding the article [6], "Towards bees detection on images: study of different color models for neural networks", the main focus was shifted towards the comparison of different color models used as an input format for feedforward convolutional architecture applied for bees detection. This is considered an important contribution since it partially dictates how images should be fed to the network regarding their color model, in order to acquire better results. Unfortunately not much detailed information is given regarding the actual dataset and the classifier model used, aside from "a deep convolutional neural network composed of few convolutional layers and few full connected layers", as cited in [6]. Regardless, this model was used to construct a binary classifier which allowed them to distinguish frames that contain bees from background images. This article concluded that the usage of 3-channel color models has significant advantage over 1-channel models and RGB in particular has a small advantage over the HSV color model.

The final article we explored was titled "Tracking individual honeybees among wildflower clusters with computer vision-facilitated pollinator monitoring" [7]. Although it documents an approach more focused on real time video processing, with the main purpose of accurately mapping a sequence of interactions between a particular insect and its foraging environment, it is an interesting solution encompassing multiple strategies instead of adopting a single, existing one, for the detection of honeybees. The Hybrid Detection and Tracking (**HyDaT**) algorithm proposed by the authors is described as follows: "A hybrid detection algorithm begins at the start of the video and moves through the footage until it first detects and identifies an as yet untracked insect. If this insect is not detected within a subset of subsequent frames, the algorithm uses novel methods to predict if it is occluded or has exited the view. Positional data collected from the algorithm is then linked to synthesise coherent insect trajectories. Finally, this information is analysed to obtain movement and behavioural data".

What makes this solution so interesting is the use of both a hybrid algorithm consisting of background subtraction and a deep learning-based detection to locate an insect, making use of the advantages of background subtraction, through which it is possible to detect movements in the foreground without prior training, working efficiently where the background is mainly stationary, and a deep learning-based detection, which is able to detect and identify an insect irrespective of changes in the background, but requiring training with a dataset prior to use.

The algorithm begins using the trained deep learning model to initialise the detection process by locating the insect's first appearance in a video. After that, the switch between both of the previous methods is determined by the number of regions of inter-frame change within a calculated radius of the predicted position of the insect in the next frame, if there is a single region of significant change identified between frames, the background subtraction technique is used to locate the insect, otherwise, the deep learning approach is used.

For the deep learning-based detection, a CNN based on a Darkflow implementation of the model YOLOv2 (You Only Look Once) is used. the images required for training the deep learning-based detection model were extracted from videos recorded in Scaevola groundcover using FrameShots, which were then manually filtered to remove those without honeybees, ending up with 2799 honeybee images. A K-nearest neighbour (**KNN**) based background/foreground segmentation, using OpenCV, to detect foreground changes in the video, was used for the background subtraction-based detection.

As it can be seen, from the articles that were selected

and analysed, the application of deep learning models, some performing better than others, was considered a central piece for detecting the presence of bees and/or their subspecies in images. The were also important considerations regarding the pre-processing and manipulation of the dataset data to have in mind during the development of this project, such as its balancing, color model and image resizing.

III. Dataset

A. Data Retrieval

In order to create and train the bee subspecies classifier we decided to use one of kaggle's available datasets "The BeeImage Dataset: Annotated Honey Bee Images". This dataset provides 5172 images of bees, extracted from still time-lapse videos of bees, and the subspecie they belong to, as well as additional information regarding the location of the hive is was sighted at, date of sight, time of sight, health condition of the bee, caste, and if it was carrying pollen or not.

The subspecies provided by this dataset are as follows:

- Italian Honey Bee;
- Russian Honey Bee;
- Carniolian Honey Bee;
- Mixed Local Stock;
- -1 (later renamed to 'Unknown subspecies');
- VSH Italian Honey Bee;
- Western Honey Bee;

As it can be seen, this dataset is able to provide much more information aside from the actual images, which could also serve as important inputs for the classifier. However, as previously mentioned, this classifier is meant to assist other solutions in preserving the bee population by distinguishing them and their subspecies from other, possibly invasive, species, meaning that such solutions may only have access to cameras, being able to generate image data only as possible inputs and no additional information. As a result, we decided to train the classifier based solely on image data.

B. Data Manipulation

After selecting the dataset to be used, a more thorough analysis regarding the actual data composition and structure was made, giving rise to issues previously undetected.

By observing multiple images present in the dataset, such as the random set in figure 1, it is possible to perceive that most of the images have different sizes, appearing to have been cropped manually given the disparity in image sizes, as shown in figure 2.

In order to tackle this issue, the dataset images need to be resized to a proper size (100x100). The proper image size was decided by taking into account the average width and height of all the images (76x76) and the computational time for image processing. It is worth mentioning that all image manipulation operations were done over flat normalized arrays of image pixel data, containing all 3 RGB channels instead of simply being grayscaled.

After selecting the desired image size, every image was rescaled to the desired height and width, regardless of their initial size, making most dataset images usable for model training, therefore enabling it to generate more accurate results.

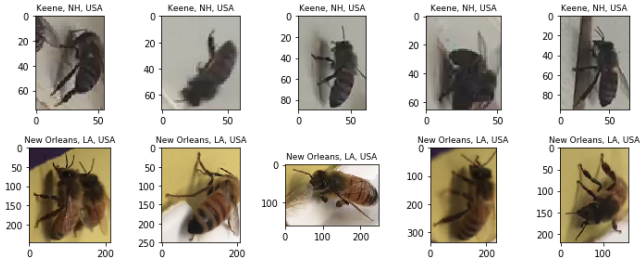

Fig. 1: Example of images provided by the dataset.

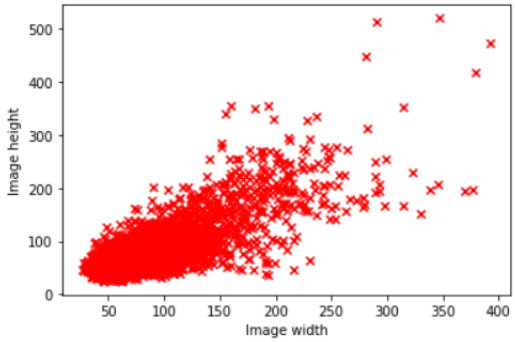

Fig. 2: Height and width of every image in the dataset.

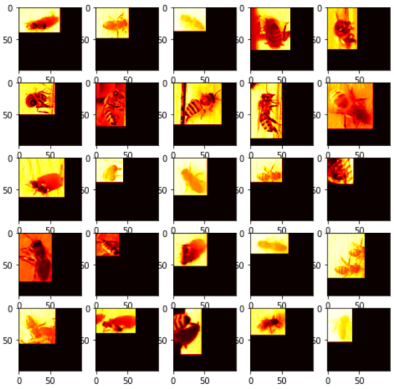

Fig. 3: Example of resized dataset images generated by the "Image extending" strategy.

Although the images are now resized to a common and proper size, there is still another issue related to the amount of images available per class. An unbalanced dataset may cause suboptimal classification results [8], since the model may have few images of certain classes to derive meaningful information and/or may not have sufficient images to generate a test and/or validation set from.

As it can be seen in figure 6, the number of images belonging to the "Italian Honey Bee" subspecie (3008) are much higher than the number of images for the rest of the classes, averaging 360 images. It can also be observed that the "Western Honey Bee" only contains 37 images, which is

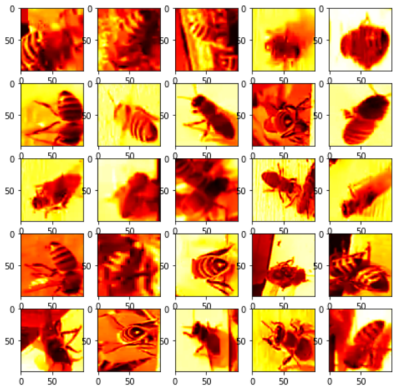

Fig. 4: Example of resized dataset images generated by the "Image rescaling" strategy.

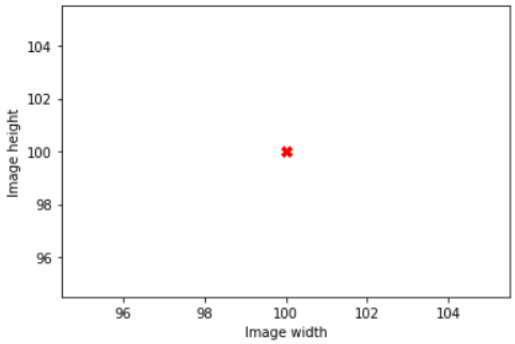

Fig. 5: Height and width of every image in the dataset, resized.

a much lower number than the others.

Instead of simply undersampling or oversampling each class, we decided to have the minimum desired number of images per class as an input parameter to our image balancing algorithm. This parameter acts as a minimum threshold because we intend to supply the classifier with the most amount of images as possible, therefore, for a minimum threshold of 500 images per class, given the graph in figure 6, the balancing algorithm discards all classes with a number of images inferior to 500 and, from the remaining classes, selects a number of images, from each, equal to the image count of the smallest class (501). Figure 7 displays the balancing algorithm's result for the previous scenario.

Although the previous solution is able to correctly balance the amount of images per class, simply discarding whole classes because they were missing a number of images which could be easily generated based on the existing ones, is not an optimal solution. To solve this, the balancing algorithm has been adapted to oversample images depending on the number of images left for the class to reach the minimum amount required.

Now, the classes with a number of images higher than half (half since a single transformation is applied to each original image) of the defined count are kept, the rest are discarded. From the remaining classes which exhibit a number of images higher than the minimum threshold, the class containing the least amount of images, limits the amount of images each class must have. The classes which display a number of images below the defined count but above half of it are oversampled until the limit amount is reached. The oversampling process takes into account the amount of images needed to reach the limit and replicates and rotates (randomly between 90 and 270 degrees) that same number of images from that class. Figure 8 displays the new amount of images per class for the same minimum threshold as before (500).

After these steps, the dataset was considered ready to be used.

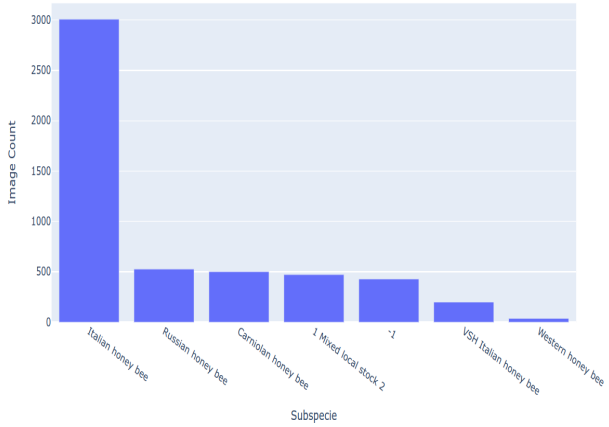

Fig. 6: Number of images per subspecie.

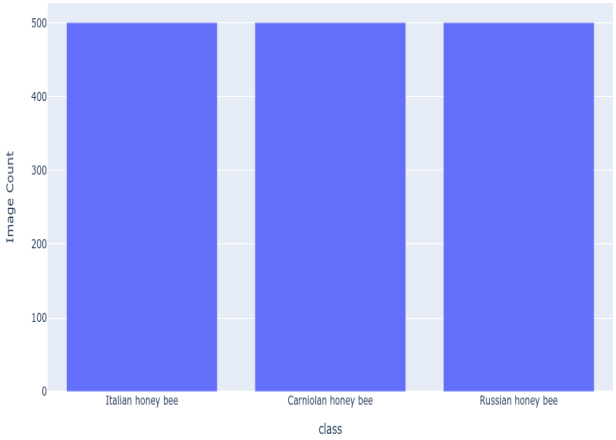

Fig. 7: Number of images per subspecie, balanced data given a minimum image count threshold per class.

IV. Methodologies

After the process of acquiring and managing the chosen dataset we were able to start implementing the ML algorithms. In this project we aimed to use supervised learning in order to solve a common problem in this area of investigation, which is the classification problem. Since the dataset has a relatively small amount of data, there were many approaches to the classifier. In a first phase of the process, described on the

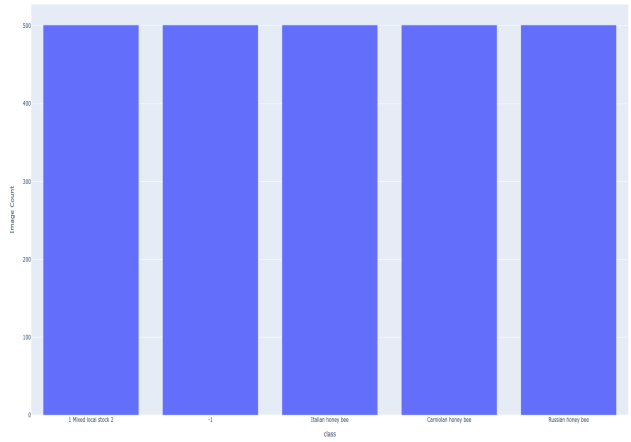

Fig. 8: Number of images per subspecie, balanced data given a minimum image count threshold per class with data augmentation (oversampling).

previous report, the main focus was to apply a Neural Network. In a further approach, we applied Deep Learning using a CNN (Convolutional Neural Network) in order to compare both of the strategies and its results and see what's the best classifier method for this dataset and problematic. The process was divided into seven important stages:

1) Study and definition of the problematic;
2) Loading of the Data;
3) Creation of a CNN model with certain hyper-parameters;
4) Creation and training of the model;
5) Evaluation and testing of the model;
6) Correction and adaptation of the hyper-parameters to decrease errors.
7) Performance comparison between the models and metrics;

A. Definition of the problematic

As said before, this project aims to solve a classification problem with supervised learning recurring to two known algorithms for comparison of results between both.

As specified in the previous report, the dataset has a very unbalanced data distribution for the classes shown above. That way, we only considered for the classification the first five classes itemized on III-A, using the Oversampling method explained to balance the data between said classes. Since it is a very complex nonlinear model with large images of 100x100x3 (RGB) pixels, the Convolutional Neural Network is the chosen fit for solving this problematic. In the previous work, we incurred in an overfitting problem in some of the results, which we aim to correct in the present work recurring to this algorithm (which is less prone to overfitting) and using the oversampling method.

For the purpose of introduction to the model implemented in this project, we present below a brief theoretical description of the Convolutional Neural Network algorithm.

Convolutional Neural Network (CNN) [9] is a deep learning approach that is used for solving complex problems with remarkable performance, like image classification. It is used in various domains because it overcomes the limitations of traditional machine learning approaches, not suffering overfitting due to lesser parameters. This lies in the idea of using concept of weight sharing, turning the model a better approach for particular problems and making the implementation easier than other type of strategies.

B. Load of the Data

An important step that is common between all models in this project and the previous one is the loading of the data from the *.csv* created with the features and labels for the chosen classes to be studied. As the labels appear in the file in strings, we needed to sort the classes into a dictionary and attribute an integer to each one in order to use a Sparse Categorical strategy and creating the dummy variables corresponding to the categorical target variable. This can be done using a categories encoder, as well. We then mapped the entire array of labels to substitute the strings according with the dictionary formed.

Next, it's crucial to choose the holdout method and how we want to split the data, to avoid incurring in a overfitting problem [10] and getting worse predictions as it fails to generalize to the new examples. In our case, due to the application of the oversampling method, we tried two approaches, with the chosen division being the traditional one that is more accurate to smaller datasets like ours (80% - 20%).

C. Creation process of the Convolutional Neural Network model

After the common step with the models described in previous work, of loading the data, we had to create and train the CNN using the functions we learned on class.

We first created the model, adapted from class, from a simple approach. The first model had only one conv layer, a batch normalization and an activation block using relu. It was applied maxpooling and a dense layer using sigmoid activation. This approach was dropped after training and adapting the parameters to get a better performance and minimal loss.

There were three phases in the creation of this model, and all of them comprehend different hyper-parameters and layers, depending on the phase and evolution of our study. In the following list we enumerate the differences that characterize a specific phase on the CNN creation process.

- First Phase: The first phase of the CNN consisted on a five classes prediction with a 100x100x3 RGB image size. In this phase we used 80%-20% proportion to the train set and test set. The parameters used were the first model described before. This had a poorly performance, being discarded. The tests resulting in this application can be found in Appendix A, Table A1.
- Second Phase: The model evolved to this phase by changing the model layers and activation functions. The compilation parameters were also updated to better fit the model and problematic described. Here we used a holdout method called Three-way-split. This strategy of

splitting data consists on dividing the dataset in three sub-sets (training set, cross validation set - also known as 'dev' set - and test set). The chosen division was the traditional one that is more accurate to small datasets like ours (60% - 20% - 20%). The tests resulting in this application can be found in Appendix A, Table A2.

- Third Phase: We ran the final model to different batch sizes and epochs and changed the holdout method to the traditional (80% - 20%), after analysing some problems in the proportion of the accuracy between training and testing sets, evaluating as well the performance metrics and watching if there were any non-trainable parameters, this way obtaining the best performance and classification results. The tests resulting in this strategy can be found in Appendix A, Table A3.

1) First Phase: The model elements and compilation parameters used in this phase of the training are described below to better summarize the first methodology used, discarded after poor results in few tests that were made.

Elements of the model:

- One convolutional layer, with 32 filters of dimension 7;
- A Batch Normalization Layer;
- A ReLU (Rectified Linear Unit) activation function;
- One Maxpool2D layer, with reduction factor 2;
- A flatten layer;
- A dense layer with 5 units and activation function Sigmoid;

The choice of optimizer was based on the paper [11] that actively argues that SGD is conceptually stable for convex and continuous optimization. These papers argument that although Adam converges faster, SGD has a better generalization and final performance. This was dropped in the consequent phases after studying forward and finding recent papers that suggest that the hyper-parameter selection can be the reason that adaptive optimization algorithms like Adam fail to generalize. The experiments took in paper [12] show better results using Adam than using SGD, when choosing the right hyper-parameters. We can see after that this is confirmed in our study as well.

Since we used the dense layer with activation function Sigmoid, it was recommended that we used the Mean Square Error as the loss function. This was later dropped when correcting this parameters to a better fit for our problem.

Compilation Parameters:

- Optimizer - SGD (Gradient descent (with momentum) optimizer);
- Loss - Mean Square Error;
- Metrics - Accuracy;

2) Second Phase & Third Phase: The following description of the final model elements and compilation parameters summarizes the process and strategy used in this phase that resulted in the best performance and classification results. We then verified that the changes made an enormous difference in the performance of the model.

In this phase we chose to decrease the dimension of the kernel size on the convolutional layer to a more common convolution filter (3x3).

At this point we verified as well that we made an error on choosing the activation function Sigmoid in the dense layer for this problematic since it's a better fit for two-classes (binary) classification problem. This happens because the probabilities produced by a Sigmoid are independent, since it looks at each raw output value separately. Verifying this fact, we then chose the Softmax approach, since the outputs of a softmax are all interrelated and so it's a better fit for our problematic of multiclass classification.

In this phase we tested and later dropped the Batch Normalization layer after verifying that it hurt our performance and that we had better results without using it.

Elements of the model:

- One convolutional layer, with 16 filters of dimension 3, with same padding (so the output size is the same as input size);
- A ReLU (Rectified Linear Unit) activation function;
- One Maxpool2D layer, with reduction factor 2;
- A flatten layer;
- A dense layer with 5 units and activation function Softmax;

The choice of loss function to apply in the model must be specific to the problem. Not only that but also the configuration of the output layer must be appropriate for the chosen loss function. In terms of our study, since we have a multiclass classification problem, the possible solutions for the loss function varied between the Categorical cross entropy and the Sparse cross entropy. The difference between the two loss functions is the need of using one-hot encoding when using the Categorical cross entropy, while in Sparse categorical cross entropy you can encode as normal integers (which is our dataset is treated, as specified before in the subsection IV-B). Our classes are also mutually exclusive so this last loss function is a better fit for our model, since is also computationally less expensive.

Compilation Parameters:

- Optimizer - Adam (Adam optimization is a stochastic gradient descent method that is based on adaptive estimation of first-order and second-order moments);
- Loss - Sparse Categorical Cross Entropy (best for multiclass classification problems);
- Metrics - Accuracy;

D. Performance comparison between the models and metrics

Every machine learning implementation process has its performance metrics parts in its pipeline. It's crucial to evaluate how a algorithm is behaving and use the metrics to be able to correct parameters and obtain better results. As the problems of machine learning can be broken in two types (Regression and Classification), its performance methods are also fitted for each problem.

Other important measure of the models model performance is the cost loss function. In a CNN it's important to choose the perfect loss function to evaluate the performance. The

loss function varies depending on the type of problem we are facing.

In our case, as we are dealing with a classification problem, we needed to sort what metrics are important to evaluate our models. After further study into the metrics [13], we decided to present the following ones:

- Accuracy (number of correct predictions made by the model);
- Confusion Matrix (not a metric but essential to get other performance metrics);
- Precision and Recall (first being important to see the false positives and the second to see the false negatives);
- F1-score (it is a combined metrics that represents both precision and recall);
- Summary details of the applied layers (to see if there were any non-trainable parameters)

As we can see from the comparison between the various tables presented on the Appendixes, the main focus was using a systematic approach, having the hyper-parameters adapted according to the analysis of the performance metrics stated above and cost loss function graph to try to reach the most desirable results.

V. Results

A. Convolutional Neural Network Model

The best result and its performance metrics obtained from testing the Convolutional Neural Network model (presented on the Appendix A) are presented in the subsequent subsections. It was reached from the application of the image scaling method (as in III-B) on the tests performed, using the Oversampling approach, and using the Three-way split holdout method as explained on the subsection IV-B. This model, conjugating the CNN characteristics and the Oversampling method, helped eliminate the overfitting problem that we obtained using a simple Neural Network, as in the previous report, and obtain better performance results. This results were better than a previous approach of a CNN model from another author using the same dataset [14]. This is due to the data pre-processing and manipulation applied and that was described in Section III. We ran the model training two times, since the model will continue to train with the parameters it has already learned instead of reinitializing them and to see what score we would achieve.

1) Accuracy Set Scores:

- Training Set accuracy: 1.0
- Testing Set accuracy: 0.97206

2) Loss Scores:

- Training Set loss: 0.01161
- Testing Set loss: 0.10371

3) Confusion Matrix and Classification Report: The classification report is very useful to obtain the majority of performance metrics that we want to analyse and that are listed on the subsection IV-D, such as the precision, recall and f1-score.

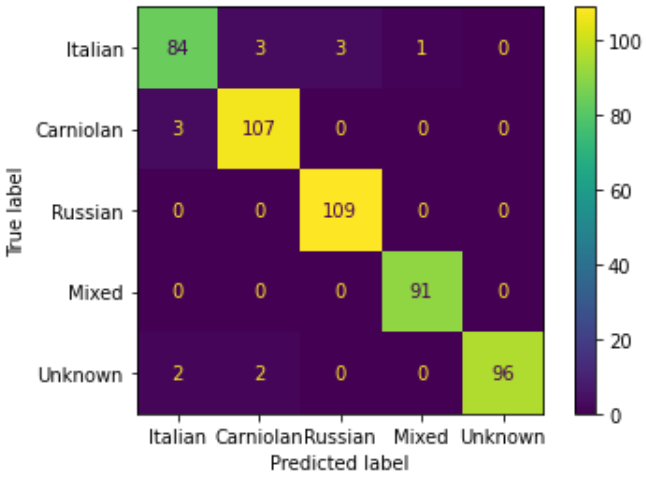

Fig. 9: Confusion matrix display for the convolutional neural network model.

TABLE I: Classification report for the convolutional neural network model.

Class	Accuracy	Precision	Recall	F1-Score
Italian honey bee	97.6%	0.92	0.94	0.93
Carniolan honey bee	98.4%	0.97	0.96	0.96
Russian honey bee	99.4%	1.0	0.97	0.99
1 Mixed Local Stock 2	99.8%	1.0	0.99	0.99
Unknow subspecies	99.2%	0.96	1.0	0.98

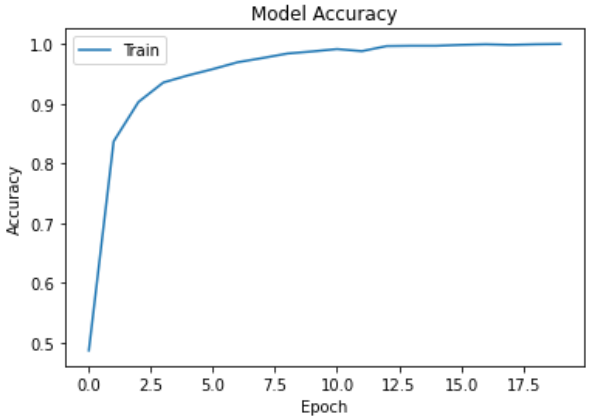

Fig. 10: Accuracy graph for the convolutional neural network model.

B. Neural Network Model with Oversampling

On the previous report we focused on the application of a Neural Network model, using different techniques for the data pre-processing of the original dataset. In this report we applied a new strategy for treating the unbalanced data, which is the Oversampling method, better described in section III-B. This focus aimed to correct the overfitting problem that happened from the previous data pre-processing. On the Appendix B, Table B1 and B2 we can compare both of results for the same hyper-parameters, the difference being the application of the method stated before. The best result and its performance metrics obtained from testing the Neural Network model

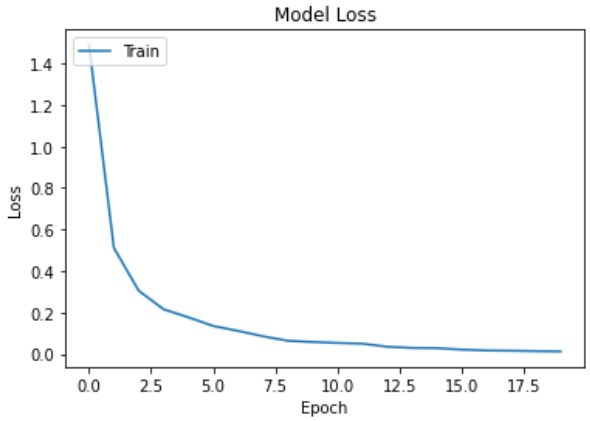

Fig. 11: Loss graph for the convolutional neural network model.

(presented on the Appendix B) are described in the subsequent subsections. It was reached from the application of the image rescaling method (as in III-B) in a image size of 100x100 and three classes prediction on the tests performed and using the 80%-20% proportion holdout method and applying the Oversampling strategy to eliminate overfitting as explained on the subsection IV-B. The following hyper-parameters created the best result:

- Choice of activation function: *Sigmoid*
- Choice of optimization algorithm: *Gradient Descent*
- Train-test split ratio: 80%-20%
- Input Layer Size: 10000
- Number of hidden layers: 1
- Hidden Layer Size: 15
- Number of iterations: 10000
- Learning rate (Alpha): 0.03
- Regularization amount (Lambda): 0.001

1) Accuracy Set Scores:

- Training Set accuracy: 70,958%
- Testing Set accuracy: 64,671%

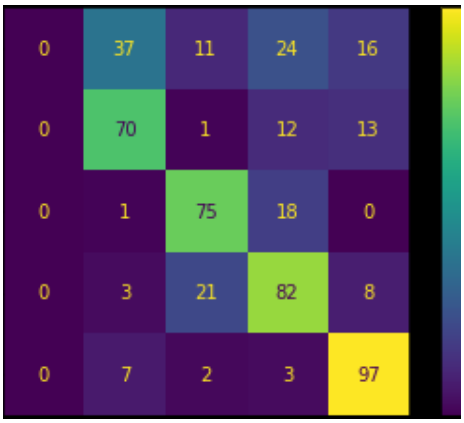

Fig. 12: Confusion matrix display for the neural network model using oversampling.

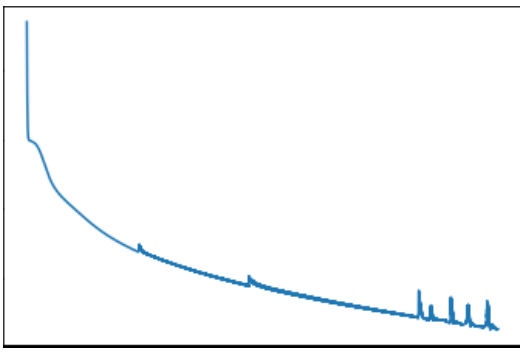

Fig. 13: Cost function graph using the gradient descent for the neural network model using oversampling.

C. Comparison of the results

In terms of overall results and proportion between the training set and testing set accuracy, we can affirm that the Convolutional Neural Network presents the best result, presented on the Appendix B table B2. It can also predict a wider variety of bee subspecies and the training times are faster for this approach (while the best result with the NN had a training time of a day, the CNN trains in minutes).

The overfitting problem that was obtained in the previous report is also considered solved by using the Oversampling method to both the NN and CNN models.

Although, it is important to note that the results from the CNN model and from the Neural Network are not fit from the same data, since one is implemented using gray-scale images and the other one is RGB images.

All the training and testing was made using other computing tools at our disposal such as “Google Colab” (a cloud service where machine learning models can be trained on a Tesla K80 GPU for free), which allowed the training process of the models to be done much faster. This was another improvement compared to the previous work.

Despite this improvement, some obstacles were faced using this service. A lot of tests that were made for both of the models were very time-consuming and required high computational resources. For that reason, “Google Colab” stopped many runtimes for achieving the limit of usage in GPU for the free version of the service and some tests did not make to the final screening of tests display.

VI. Conclusion

In conclusion, we believe that although not all subspecies ended up being actually classified, compared to the previous work, there was a wider integration of subspecies, and that given the present testing parameters, the developed classifier is a step forward towards the protection of bee population, enabling the distinction of bees from other possible species, while, at the same time, having a notion of the existing subspecies and some of their unique features. This work differs and stands out from works of other authors on this topic because a majority of the existent solutions aim to protect the bee population by focusing on monitoring the hive itself

rather than protecting it from outside threats, as is depicted in the present document.

The future approaches described in the previous report were applied in the present work and were considered to be an improvement and evolution of the algorithm developed. The current version using the oversampling method allows the classifier to not only generate more realistic results, given more images and classes to distinguish from, but also improve the classifier's accuracy.

In a future approach, we would try to integrate a higher number of classes to fully integrate the dataset and obtain more images for those classes to create a more balanced and complete dataset by, for example, improving the oversampling process, generating multiple rotations per image instead of a single one and applying other transformations.

In short, we consider that the model behaviour is better than other proposed models from another authors for the same dataset [14], leading us to conclude that our data pre-processing approach to treat this unbalanced and small dataset was a good strategy and is what makes a difference from the other proposed works.

REFERENCES

[1] P. G. Kevan and B. F. Viana, "The global decline of pollination services," *Biodiversity*, vol. 4, no. 4, pp. 3–8, 2003. [Online]. Available: https://doi.org/10.1080/14888386.2003.9712703

[2] K. Monceau, O. Bonnard, and D. Thiéry, "Vespa velutina: a new invasive predator of honeybees in europe," *Journal of pest science*, vol. 87, no. 1, pp. 1–16, 2014.

[3] D. De Nart, C. Costa, G. Di Prisco, and E. Carpana, "Image recognition using convolutional neural networks for classification of honey bee subspecies," *Apidologie*, vol. 53, no. 1, pp. 1–15, 2022.

[4] B. J. Spiesman, C. Gratton, R. G. Hatfield, W. H. Hsu, S. Jepsen, B. McCornack, K. Patel, and G. Wang, "Assessing the potential for deep learning and computer vision to identify bumble bee species from images," *Scientific reports*, vol. 11, no. 1, pp. 1–10, 2021.

[5] J. Yoo, M. Hossain, and K. Ahmed, "A machine learning based approach to study morphological features of bees. proceedings 2021, 1, 0," 2021.

[6] J. Dembski and J. Szymański, "Towards bees detection on images: study of different color models for neural networks," 2019.

[7] M. N. Ratnayake, A. G. Dyer, and A. Dorin, "Tracking individual honeybees among wildflower clusters with computer vision-facilitated pollinator monitoring," *Plos one*, vol. 16, no. 2, p. e0239504, 2021.

[8] N. V. Chawla, N. Japkowicz, and A. Kotcz, "Special issue on learning from imbalanced data sets," *ACM SIGKDD explorations newsletter*, vol. 6, no. 1, pp. 1–6, 2004.

[9] S. Indolia, A. K. Goswami, S. Mishra, and P. Asopa, "Conceptual understanding of convolutional neural network-a deep learning approach," *Procedia Computer Science*, vol. 132, pp. 679–688, 2018, international Conference on Computational Intelligence and Data Science. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1877050918308019

[10] D. M. Hawkins, "The problem of overfitting," *Journal of Chemical Information and Computer Sciences*, vol. 44, no. 1, pp. 1–12, 2004.

[11] M. Hardt, B. Recht, and Y. Singer, "Train faster, generalize better: Stability of stochastic gradient descent," in *Proceedings of The 33rd International Conference on Machine Learning*, ser. Proceedings of Machine Learning Research, M. F. Balcan and K. Q. Weinberger, Eds., vol. 48. New York, New York, USA: PMLR, 20–22 Jun 2016, pp. 1225–1234. [Online]. Available: https://proceedings.mlr.press/v48/hardt16.html

[12] D. Choi, C. J. Shallue, Z. Nado, J. Lee, C. J. Maddison, and G. E. Dahl, "On empirical comparisons of optimizers for deep learning," 2020. [Online]. Available: https://openreview.net/forum?id=HygrAR4tPS

[13] M. Hossin and M. N. Sulaiman, "A review on evaluation metrics for data classification evaluations," *International journal of data mining...*, vol. 5, no. 2, pp. 1–12, 2015.

[14] G. Preda, "Honey bee subspecies classification," 2021. [Online]. Available: https://www.kaggle.com/code/gpreda/honey-bee-subspecies-classification/notebook

APPENDIX A
CONVOLUTIONAL NEURAL NETWORK TESTS

TABLE A1: First phase of tests of the CNN.

# Test	Batch Size	Epochs	Holdout	Training Loss	Training Accuracy	Testing Loss	Testing Accuracy
#1	32	10	80/20	2.99700	0.20009	3.01197	0.19960
#2	32	15	80/20	2.99700	0.20309	3.01197	0.20758
#3	64	10	80/20	2.99716	0.13572	3.01215	0.14570
#4	64	15	80/20	2.99700	0.23552	3.01197	0.21556

TABLE A2: Second phase of tests of the CNN.

# Test	Batch Size	Epochs	Holdout	Training Loss	Training Accuracy	Testing Loss	Testing Accuracy
#1	32	10	60/20/20	0.00771	1.0	0.16111	0.94212
#2	32	15	60/20/20	0.01745	0.99800	0.14302	0.95010
#3	32	20	60/20/20	0.00604	1.0	0.19673	0.94411
#4	32	25	60/20/20	0.00999	0.99933	0.22625	0.93014
#5	32	30	60/20/20	0.00420	1.0	0.19187	0.94611
#6	64	10	60/20/20	0.08095	0.98403	0.18807	0.92814
#7	64	15	60/20/20	0.03668	0.99401	0.13780	0.94411
#8	64	20	60/20/20	0.08629	0.97139	0.26338	0.90020
#9	64	25	60/20/20	0.02167	0.99867	0.15258	0.95010
#10	64	30	60/20/20	0.00275	1.0	0.16923	0.94212

TABLE A3: Third and final phase of tests of the CNN.

# Test	Batch Size	Epochs	Holdout	Training Loss	Training Accuracy	Testing Loss	Testing Accuracy
#1	32	10	80/20	0.03899	0.99451	0.11167	0.96008
#2	32	15	80/20	0.02212	0.99950	0.13465	0.95210
#3	32	20	80/20	0.01049	1.0	0.10519	0.96407
#4	32	25	80/20	0.01468	1.0	0.16146	0.94211
#5	32	30	80/20	0.00160	1.0	0.10014	0.96806
#6	64	10	80/20	0.05308	0.98902	0.11978	0.96407
#7	64	15	80/20	0.02936	0.99800	0.12657	0.96007
#8	64	20	80/20	0.01161	1.0	0.10371	0.97206
#9	64	25	80/20	0.00462	1.0	0.12996	0.95808
#10	64	30	80/20	0.00195	1.0	0.11213	0.96606

APPENDIX B
NEURAL NETWORK TESTS

TABLE B1: Phase of tests using the holdout method and 100x100 images without oversampling.

Test	I. Size	Classes	Train Set	Test Set	Input Layer	Hidden Layer	# Iters	Lambda	Alpha	Train Set Acc	Test Set Acc
#1	100x100	3	0.8	0.2	10000	9	5000	0.001	0.03	84.609%	72.093%
#2	100x100	3	0.8	0.2	10000	9	5000	0.003	0.01	80.699%	73.090%
#3	100x100	3	0.8	0.2	10000	15	5000	0.001	0.03	80.449%	73.422%
#4	100x100	3	0.8	0.2	10000	15	10000	0.001	0.03	86.439%	76.744%

TABLE B2: Phase of tests using the holdout method and 100x100 images with oversampling.

Test	I. Size	Classes	Train Set	Test Set	Input Layer	Hidden Layer	# Iters	Lambda	Alpha	Train Set Acc	Test Set Acc
#1	100x100	4	0.8	0.2	10000	9	5000	0.001	0.03	60,629%	59,281%
#2	100x100	4	0.8	0.2	10000	9	5000	0.003	0.01	49,451%	47,904%
#3	100x100	4	0.8	0.2	10000	15	5000	0.001	0.03	65,569%	64,072%
#4	100x100	4	0.8	0.2	10000	15	10000	0.001	0.03	70,958%	64,671%