Bee Subspecies Classification using Machine Learning Methods over Bee Images

AA - Project 1

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. 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

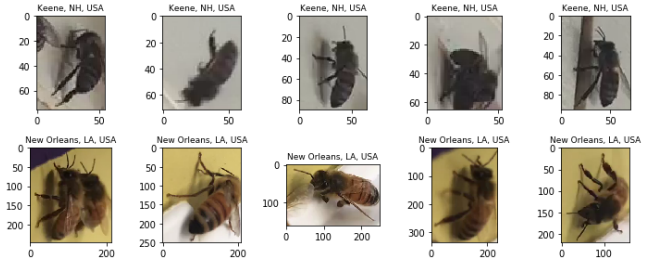

Fig. 1: Example of images provided by the dataset.

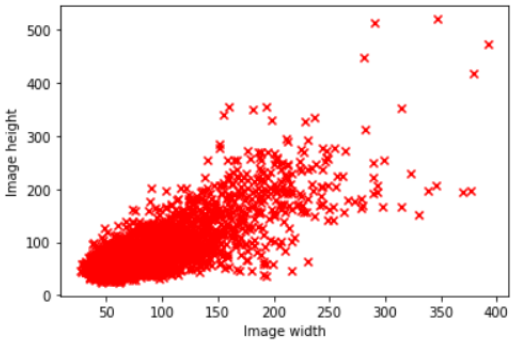

Fig. 2: Height and width of every image in the dataset.

operations were done over flat normalized arrays of image pixel data, previously grayscaled.

After selecting the desired image size, 2 resizing strategies were applied:

1) Image extending (figure 3): Images which height and width are below the desired image size are filled with black colored pixels until the target size is reached, while the rest are discarded, since cropping operations over these images (which mainly contain the actual bee(s)) would likely lead to the loss of important information;
2) Image rescaling (figure 4): Every image is rescaled to a desired height and width, regardless of their initial size.

Both strategies exhibit advantages and disadvantages, as detailed in the following table:

Strategy	Advantages(+)/Disadvantages(-)
Image extending	+ Faster image processing - Lower number of usable images - Additional black pixels may mislead the classifier
Image rescaling	+ Most dataset images are usable - Significant slower image processing

While the image extending strategy was used as a first approach, and as observed later on (section IV), it has shown better results on a superficial analysis. The second strategy was adopted as the main and preferable solution given the fact that the classifier is supplied with more images, enabling it to generate more accurate results versus the "Image extending" solution. Despite having more features in each image and for that reason scaling the tests time, there were enough tests of this strategy to confirm this accuracy.

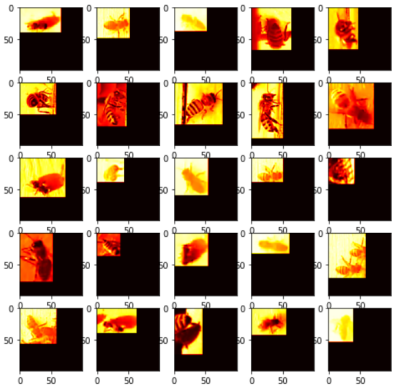

Fig. 3: Example of resized dataset images generated by the "Image extending" strategy.

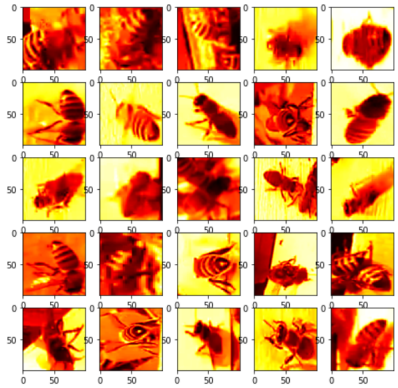

Fig. 4: Example of resized dataset images generated by the "Image rescaling" strategy.

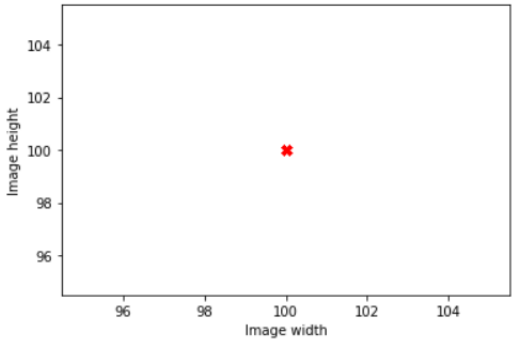

Fig. 5: Height and width of every image in the dataset, resized.

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 [3], 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 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).

Since this update to the balancing algorithm was implemented in a later phase of the development of this project, the additional images and classes the algorithm is now able to select were not used during the training of the models.

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

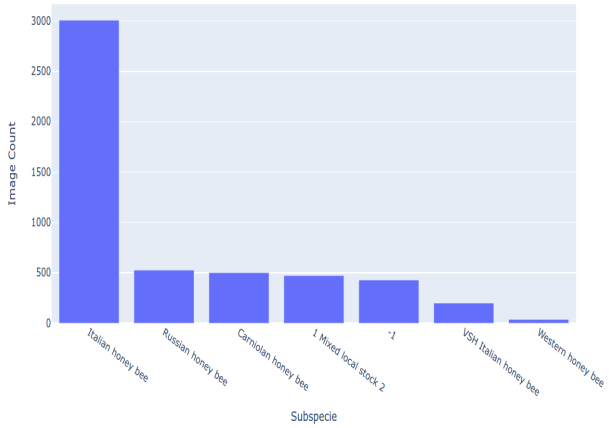

Fig. 6: Number of images per subspecie.

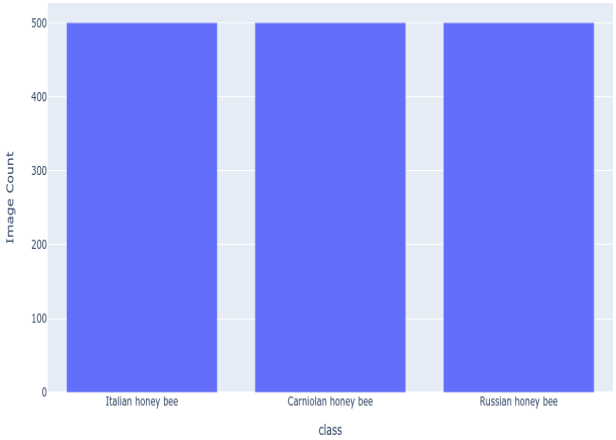

Fig. 7: Number of images per subspecie, balanced data given a minimum image count threshold per class.

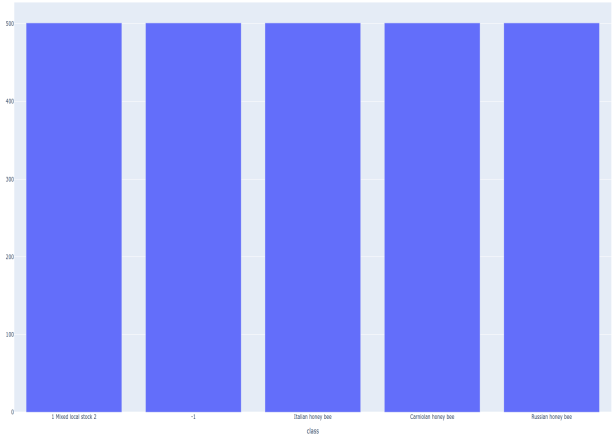

Fig. 8: Number of images per subspecie, balanced data given a minimum image count threshold per class with data augmentation (oversampling).

III. 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. Despite this fact, the main focus was always aiming to developing a Neural Network. The process was divided into seven important stages:

1) Study and definition of the problematic;
2) Loading of the Data;
3) Application of the Logistic Regression algorithm;
4) Creation of a Neural Network with certain hyper-parameters;
5) Testing of the Neural Network;
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. The provided dataset has seven classes so we used a One-versus-all strategy for the multiclass classification of the following subspecies, sorted in descending order by number of images:

- Italian Honey Bee;
- Russian Honey Bee;
- Carniolian Honey Bee;
- Mixed Local Stock;
- Unknown subspecies;
- VSH Italian Honey Bee;
- Western Honey Bee.

As specified before, the dataset has a very unbalanced data distribution for the classes shown above. That way, we only considered for the classification the first four classes itemized, and for the majority of the tests we only considered the first three classes. The logic behind this choice is explained later on. Since it is a very complex nonlinear model, the Neural Network is the chosen fit for solving this problematic. As a matter of comparison, we used Logistic Regression also.

For the purpose of introduction to the models implemented in this project, we present below a brief theoretical description of the Logistic Regression and Neural Network algorithms.

Logistic regression is a predictive analysis and in that way is used to describe data as well as the relationship between one dependent binary variable and one or more other type of continuous independent variables [4].

On the other hand, a neural network consists on the recognition of the relationships in a data set using a process that has the purpose of mimicking the human brain neurons and its connections [5]. A neural network refers to a organic or artificial system of neurons in nature.

B. Load of the Data

An important step that is common between all models 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 to be a string, to be able to do the One-vs-all strategy, we needed to sort the classes into a dictionary and attribute an integer to each one. 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 [6] and getting worse predictions as it fails to generalize to the new examples. In our case, due to the small dataset and number of images in each class, we divided 80% of the data into the training set and the remaining 20% were used to test the trained model (test set). This, alongside with the 70%-30% relation are the most common holdout methods to use.

There was also a time in the project where we considered using a data split method called Three-way split to the Neural Network model, as we did on the Logistic Regression Model - better described on the following subsection "Application of the LR model" (III-C). 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 , Table B2.

However, this method was dropped after testing and acknowledging that our dataset is too small for it to be advantageous having a cross validation set to the algorithm at stake, as we needed a bigger proportion between the training set and test set.

C. Application of the LR model

"Logistic regression is one of the most used statistical procedures in research."(Hilbe, 2011, p. ix) [7]

For an easy implementation of the LR model, we resorted to the known library *scikit-learn*. With this library we are able to choose the maximum iterations for the logistic regression and fit the train set. We then test the results with the function ***score()*** using the previously defined test set.

For this algorithm there was some liberty for testing the different cross validation methods. The following list contains the tested methods and its pros and cons:

- K-Fold cross-validation: As a pro of the usage of this validation, the whole dataset is used as both a training set and validation set, which is valuable for our type of study since the extent of our data. However, there's a con that's being unsuitable for imbalanced data sets and for that reason created difficulty in our first phase of the tests (since there were some classes in our data with large quantities of images and others with almost none);
- Stratified K-Fold cross-validation: This validation solves the previous stated con on the K-Fold CV because it works perfectly well for imbalanced data. As its cons, which are not relevant in our case, it is not suitable for Time Series data.
- Leave One Out cross-validation: This validation was implemented because it is considered convenient when treating small data sets. However, it comes with the con that it can be computationally expensive and time-consuming if a model is particularly complex and for that

reason we couldn't produce tests in a productive time using this CV.

The tests and results from the usage of this tactics are detailed on the section IV - "Results".

D. Creation process of the Neural Network model

After the common step with the LR model described before, of loading the data, we had to train the neural network using the functions we learned on class. We needed to compute the cost function and the gradient of the NN, recurring to the use of the known sigmoid function

$$h_\theta(x) = \frac{1}{1 + e^{-\theta^T x}}$$

to get the hidden and output layer and implementing the back propagation algorithm [8] to compute the gradients.

Next, we computed the gradient of sigmoid function and randomly initialized the weights of a layer with the input layer size (incoming connections) and the hidden layer size (outgoing connections) for the initial value of $Theta1$. For the value of the $Theta2$ we initialized the weights with the hidden layer size (incoming connections) and number of labels (outgoing connections).

After defining the model hyper-parameters (number of layers in the NN, number of hidden units (neurons), number of labels, learning rate (α), regularization parameter (λ) and number of iterations) we computed the gradient descent that returns theta and the list of the cost of theta during each iteration to then predict the label of an input given the already trained neural network. This is done by computing the output of the hidden layer (with sigmoid activation functions) using inputs to the hidden layer neurons and outputs of the hidden layer neurons and by computing the output of the output layer (with the same sigmoid activation functions) using the inputs and outputs of the output layer neurons.

We then predicted the training set accuracy and the testing set accuracy and that tells us if we are incurring on the overfitting problem explained on the subsection III-B - "Load of the Data" or in a underfitting problem - which is a high bias problem, characterized by the high value of both the training error and testing error.

There were four 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 neural network creation process.

- First Phase: The first phase of the NN consisted on a four classes prediction with a 50x50 image size and so a 2500 input layer size. In this phase we alternated between 80%-20% and 70%-30% proportion to the train set and test set.
- Second Phase: The model evolved to this phase by choosing to do a Three-Way Split method to split the data and resizing the images size. There was also a change of the number of classes to use more balanced data.
- Third Phase: In this phase we noted that as we had a very unbalanced dataset it would be best to only train and test to predict 3 classes which had a 500 image threshold (minimum number of images to be considered). As the dataset is too small for the three way split, we also returned to the holdout method of splitting data. Here we also trained bigger number of iterations to get the cost function to converge to a flat rate so that we would consider the learning finished. Here we also used a bigger number of neurons in the hidden layer to watch the behaviour of the model.
- Fourth Phase: The process of changing from phase three to phase four was the one that made us change the resizing strategy to the image rescaling method as stated in the subsection II-B - "Data Manipulation".

E. 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. It varies from the performance metrics as it is often used to train a model with the gradient descent optimization (like in our neural network).

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 [9], 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);

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.

IV. Results

A. Logistic Regression Model

The best result and its performance metrics obtained from testing the Logistic Regression 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 II-B) on the tests performed and using the Three-way split holdout method as explained on the subsection III-B.

1) Accuracy Set Scores:

- Training Set accuracy: 100%
- Testing Set accuracy: 79.734%

2) 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 III-E, such as the precision, recall and f1-score.

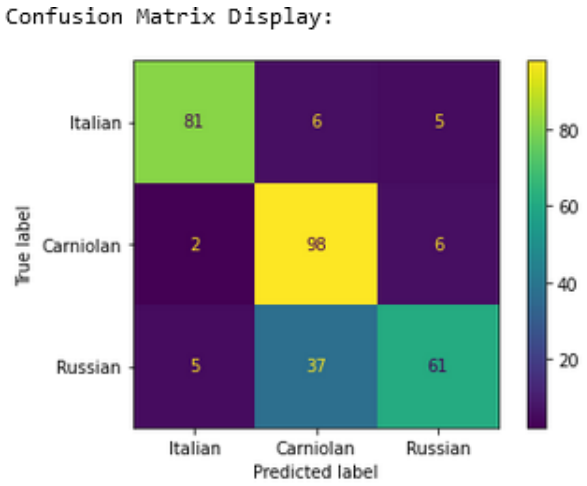

Fig. 9: Confusion matrix display for the logistic regression model.

Classification Report:

	precision	recall	f1-score	support
Italian honey bee	0.92	0.88	0.90	92
Carniolan honey bee	0.70	0.92	0.79	106
Russian honey bee	0.85	0.59	0.70	103
accuracy			0.80	301
macro avg	0.82	0.80	0.80	301
weighted avg	0.82	0.80	0.79	301

Fig. 10: Classification report for the logistic regression model.

B. K-Fold and Stratified K-Fold cross-validation

The results from the usage of this methods for the cross-validation are derived from the tests presented on the Appendix A, tables A2 and A4. What differs from both of the tables is the image method used, since it was performed tests for both of the methods explained on the subsection II-B, which are the Image Expansion and Image Scaling methods.

C. Neural Network Model

The best result and its performance metrics obtained from testing the Neural Network model (presented on the Appendix B) are presented in the subsequent subsections. It was reached from the application of the image rescaling method (as in II-B) in a image size of 100x100 and three classes prediction on the tests performed and using the 80%-20% proportion holdout method as explained on the subsection III-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: 20
- Number of iterations: 20000
- Learning rate (Alpha): 0.01
- Regularization amount (Lambda): 0.003

1) Accuracy Set Scores:

- Training Set accuracy: 97.421%
- Testing Set accuracy: 84.053%

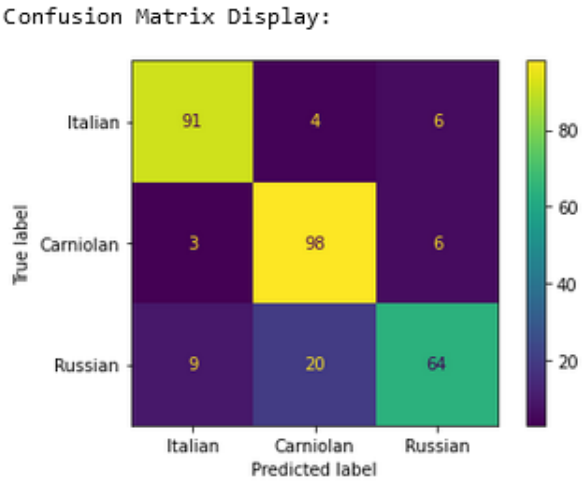

Fig. 11: Confusion matrix display for the neural network model.

Classification Report:

	precision	recall	f1-score	support
Italian honey bee	0.88	0.90	0.89	101
Carniolan honey bee	0.80	0.92	0.86	107
Russian honey bee	0.84	0.69	0.76	93
accuracy			0.84	301
macro avg	0.84	0.84	0.84	301
weighted avg	0.84	0.84	0.84	301

Fig. 12: Classification report display for the neural network model.

D. Comparison of the results

As observed on the antecedent subsections, the testing set accuracy is higher on the Neural Network model for this type of problematic. However, on terms of the training set accuracy, we can see that the Logistic Regression model presents better results from the fact that it is not manually implemented, resulting in a better efficiency for the training of the model. In terms of testing set accuracy, the Neural Network overall surpasses the Logistic Regression as it was expected.

Although, it is important to note that the results from the Logistic Regression model and from the Neural Network are not fit from the same data, since one is implemented recurring

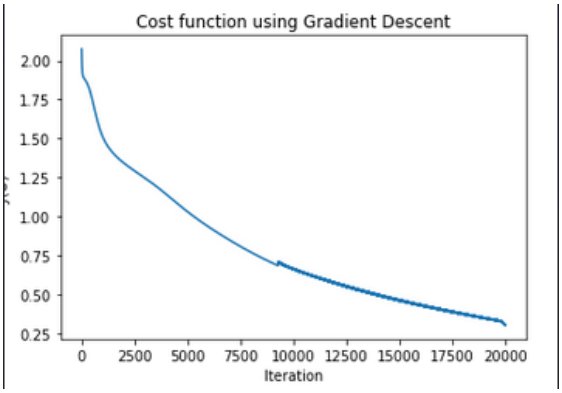

Fig. 13: Cost function graph using the gradient descent for the neural network model.

to a library and it generates new random sets and the other one is a manual implementation adapted to our case and uses the sets for the tests within the same phase.

In terms of proportion between the training set and testing set accuracy, we can affirm that the Neural Network presents the best result in the phase four of the tests, presented on the Appendix B table B4.

In a future approach, we would try to test the data for a higher number of iterations using the hyper-parameters used on the considered best result for the NN, since it would be better to see the cost function converge to a complete flat rate.

V. Conclusion

In conclusion, we believe that although not all subspecies ended up being actually classified, 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.

For better results, in a future approach, the output of the improved version of the balancing algorithm should be used over the current version, allowing the classifier to not only generate more realistic results, given more images and classes to distinguish from, but could potentially improve the classifier's accuracy. Another improvement would be making earlier use of other computing tools at our disposal, aside from our own machines that, although tested for this project, such as "Google Colab" (a cloud service where machine learning models can be trained on a Tesla K80 GPU for free) allowing the training process of the models to be done much faster, was not used to its full potential due to time constraints.

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] 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.

[4] S. Sperandei, "Understanding logistic regression analysis," *Biochem. Med. (Zagreb)*, vol. 24, no. 1, p. 13, 2014.

[5] J. Zupan, "Introduction to artificial neural network (ann) methods: what they are and how to use them," *Acta Chimica Slovenica*, vol. 41, pp. 4–5, 1994.

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

[7] J. M. Hilbe, "Logistic regression." *International encyclopedia of statistical science*, vol. 1, p. ix, 2011.

[8] R. Rojas, *The Backpropagation Algorithm*. Berlin, Heidelberg: Springer Berlin Heidelberg, 1996, pp. 149–182. [Online]. Available: https://doi.org/10.1007/978-3-642-61068-4_7

[9] 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.

APPENDIX A
LOGISTIC REGRESSION TESTS

TABLE A1: Tests for the Logistic Regression Model using the image expansion method.

Test	Images Size	# Classes	Train Set	Test Set	Dev Set	Num Iters	Training Set Accuracy	Testing Set Accuracy
#1	100x100	3	0.8	0.2	/	20000	100.000%	76.744%
#2	100x100	3	0.7	0.3	/	20000	100.000%	76.940%
#3	100x100	3	0.6	0.2	0.2	20000	100.000%	77.409%

TABLE A2: Tests for the cross validation strategies (K-Fold vs Stratified) from the tests in table A1.

Test	Average Cross Validation Accuracy	Minimum Cross Validation Accuracy	Maximum Cross Validation Accuracy	Standard Deviation
#1	77.037%/77.371%	75.416%/76.25%	77.916%/79.166%	0.00965/0.01226
#2	78.039%/77.376%	74.761%/73.809%	81.042%/80.952%	0.02313/0.02788
#3	73.919%/74.699%	72.375%/70.718%	75.555%/76.666%	0.01314/0.02467

TABLE A3: Tests for the Logistic Regression Model using the image scaling method.

Test	Images Size	# Classes	Train Set	Test Set	Dev Set	Num Iters	Training Set Accuracy	Testing Set Accuracy
#1	100x100	3	0.8	0.2	/	20000	99.750%	77.076%
#2	100x100	3	0.7	0.3	/	20000	99.905%	77.827%
#3	100x100	3	0.6	0.2	0.2	20000	100.000%	79.734%

TABLE A4: Tests for the cross validation strategies (K-Fold vs Stratified) from the tests in table A3.

Test	Average Cross Validation Accuracy	Minimum Cross Validation Accuracy	Maximum Cross Validation Accuracy	Standard Deviation
#1	76.791%/77.289%	73.443%/73.858%	81.25%/81.25%	0.02938/ 0.02780
#2	78.039%/77.376%	74.761%/73.809%	81.042%/80.952%	0.02313/0.02788
#3	76.133%/76.023%	71.667%/ 71.111%	80.663%/79.005%	0.03860/0.03155

APPENDIX B
NEURAL NETWORK TESTS

TABLE B1: First phase of tests using 50x50 images.

Test	Images Size	# of Classes	Train Set	Test Set	Input Layer	Hidden Layer	# Iters	Lambda	Alpha	Testing Set Accuracy
#1	50x50	4	0.7	0.3	2500	25	2500	0.1	1	41.176%
#2	50x50	4	0.8	0.2	2500	25	2500	0.1	1	45.098%
#3	50x50	4	0.7	0.3	2500	25	3000	0.1	1	30.719%
#4	50x50	4	0.8	0.2	2500	25	3000	0.1	1	40.196%
#5	50x50	4	0.8	0.2	2500	1	2500	0.1	1	39.216%
#6	50x50	4	0.8	0.2	2500	1	3000	0.1	1	39.216%
#7	50x50	4	0.7	0.3	2500	1	3000	0.1	1	33.333%
#8	50x50	2	0.8	0.2	2500	1	3500	0.1	1	43.478%

TABLE B2: Second phase of tests using different images size and using a three way split method.

Test	Images Size	# Classes	Train Set	Dev Set	Test Set	Input Layer	Hidden Layer	# Iters	Lambda	Alpha	Test Set Acc
#1	100x100	3	0.6	0.2	0.2	10000	1	2500	0.1	0.001	32.890%
#2	100x100	3	0.6	0.2	0.2	10000	1	2500	0.1	0.1	54.817%
#3	100x100	3	0.6	0.2	0.2	10000	1	2500	0.1	0.01	54.817%
#4	100x100	3	0.6	0.2	0.2	10000	1	3500	0.1	0.03	59.468%
#5	256x256	3	0.6	0.2	0.2	65536	1	3500	0.001	0.03	61.462%
#6	256x256	4	0.6	0.2	0.2	65536	1	3500	0.001	0.08	50.794%
#7	256x256	3	0.6	0.2	0.2	65536	1	4500	0.001	0.03	65.116%
#8	256x256	2	0.6	0.2	0.2	65536	1	4500	0.001	0.03	45.498%
#9	256x256	3	0.6	0.2	0.2	65536	1	6500	0.001	0.03	62.791%

TABLE B3: Third phase of tests using the holdout method and 100x100 images.

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	4500	0.001	0.03	87.022%	78.405%
#2	100x100	3	0.8	0.2	10000	9	10000	0.001	0.02	94.010%	77.409%
#3	100x100	3	0.8	0.2	10000	10	10000	0.001	0.02	95.840%	74.751%
#4	100x100	3	0.8	0.2	10000	15	10000	0.001	0.01	90.932%	80.731%
#5	100x100	3	0.8	0.2	10000	15	10000	0.003	0.01	90.765%	82.060%
#6	100x100	3	0.8	0.2	10000	15	10000	0.001	0.02	96.256%	76.412%
#7	100x100	3	0.8	0.2	10000	15	10000	0.004	0.01	89.767%	76.412%
#8	100x100	3	0.8	0.2	10000	15	10000	0.006	0.01	90.599%	80.066%
#9	100x100	3	0.8	0.2	10000	15	20000	0.001	0.01	96.506%	82.060%
#10	100x100	3	0.8	0.2	10000	25	10000	0.001	0.01	91.514%	76.080%

TABLE B4: Fourth phase of tests using the holdout method and 100x100 images with image rescaling method.

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%
#5	100x100	3	0.8	0.2	10000	15	10000	0.003	0.01	88.270%	78.738%
#6	100x100	3	0.8	0.2	10000	20	10000	0.01	0.01	86.439%	76.744%
#7	100x100	3	0.8	0.2	10000	20	15000	0.003	0.01	95.591%	78.405%
#8	100x100	3	0.8	0.2	10000	20	20000	0.003	0.01	97.421%	84.053%