The University of Sydney

ENGG2112

Final Report

GROUP 6
Adam Noorali, Alex Ianitto, Laura Hunt, Rahul Jayakanthan
520658590, 530482567, 530527374, 530527684

Contents

1 Executive Summary

This report details the conception, reasoning and design of a machine learning (ML) based solution to aid with early detection of Varroa Destructor mite infestation in bee colonies. Through the application of a Convolutional Neural Network (CNN) trained on publicly available datasets of labelled bee images, the constructed model is able to classify whether bees are infected with Varroa mites to around an 87% Area Under the Curve (AUC) by default. In addition, the model was made robust to environmental changes through the investigation of various preprocessing techniques. This allowed the investigation of different preprocessing recipes that allowed the model to reach an AUC of 92%. The model enables fast, scalable and accurate mite detection purely through hive imaging, supporting beekeepers in applying targeted treatment, helping efforts to preserve global pee populations and thus food security and biodiversity.

2 Project Overview

2.1 Introduction & Background

Bees remain as one of the most crucial benefactors to food security around the world, being responsible for the pollination of 80% of the world's flowering plants. In addition, 35-40% of the food that human's consume is reliant on the pollination from honeybees[1]. As a result, the global decline in honeybee populations poses a major threat to food security. Not only that, but as a keystone species, their health directly supports biodiversity in the food web through sustaining wild plant life and, in turn, providing habitats and food for other fauna.

Multiple factors have contributed to bee decline in recent decades — including pesticides, parasites, habitat loss, climate change and poor beekeeping practices — often leading to Colony Collapse Disorder (CCD). In the United States, beekeepers have consistently reported losses of 30–40% of their colonies annually since the mid-2000s[2]. In Europe, nearly 1 in 10 wild bee species is threatened with extinction, and population trends are decreasing for over 37% of bee species[1].

2.2 Problem Definition & Proposed Solution

The Varroa destructor mite is regarded as the leading cause of CCD among all others. Existing detection methods for the invasive mite are currently purely manually implemented, including sugar shakes, alcohol washes, and drone brood uncapping. These techniques are hive-invasive, labour-intensive, and prone to human error[3]. In addition, they provide only periodic snapshots of hive health and do not scale well to larger operations, as they take heavy amounts of time and resources to conduct.

Thus, the problem is then identifying non-invasive solutions which involve little resources and human labour, while providing a more accurate identification of the mites, on demand, as frequently as necessary.

The ML project designed in this report details a non-invasive, automated solution using CNNs to classify bee images as infested or healthy, in response to the problem definition. Cameras aimed at hive entrances could enable real-time, image-based detection, which is far more scalable to commercial apiaries. Early, accurate detection would allow targeted treatment of the mites per hive, before they have a chance to weaken bees and cause CCD. This would reduce both unnecessary chemical use during blanket treatment applications, and the risk of untreated infestations. At the same time the solution also lessens stress on bees and promotes more sustainable and future-proof hive management.

2.3 Project Impacts and Beneficiaries

If deployed, this project has the potential to deliver widespread benefits to a variety of stakeholders. Commercial apiaries are likely to be among the primary beneficiaries, as the use of non-invasive methods for early diagnosis of Varroa mite infestations can significantly contribute to the preservation of hive populations. These populations are essential for the production of honey, wax, and other bee-related products. By potentially reducing Colony Collapse Disorder (CCD), the project may also support an increase in global bee populations, which is crucial for biodiversity and food security.

Broader impacts include promoting non-invasive approaches in agriculture and encouraging more sustainable beekeeping practices. However, potential negative consequences must also be considered. These include the generation of electronic waste, increased energy consumption, initial deployment and maintenance costs, and potential accessibility challenges for smaller or rural apiaries.

3 Project Objectives

The overarching goal of this project was to develop a robust, non-invasive, and scalable ML solution for the early and accurate identification of Varroa destructor mite infestations in honeybee colonies. To achieve this, the project focused on the following primary, secondary and tertiary objectives, with their solutions also detailed:

- **Primary - Maximising Mite Detection Accuracy:** To train a CNN model capable of accurately classifying whether a bee in a given image is infested with a Varroa mite (`has_varroa_mite` $\in$ `True, False`). The accuracy of the CNN is the crux of the project and determines whether the project was successful and can be considered field-deployable.

  → This was achieved through gaining a holistic understanding of CNN architecture, and then conducting iterative model improvements and tuning of parameters and hyper-parameters to gain the incrementally better results. This also involved finding the best preprocessing recipe that yielded the highest accuracy.

- **Secondary - Enhancing Robustness to Environment Variability:** To optimise the model's performance under the uncertainties that arise from real-world bee imagery. The aim was to ensure reliable detection even in noisy or unclear environments, such that the group can be confident in the product's field-deployability.

  → This was achieved via systematically applying controlled preprocessing techniques to the dataset and evaluating the model's robustness using metrics such as F1 score and accuracy.

- **Tertiary - Ensuring Model Extensibility:** To design a model with inherent potential for adaptation and use in related or even unrelated fields. This objective was kept in the back of the group's minds, as it was found there are a number of parasitic insects for which the model can be extended, such as ticks within the livestock industry or in veterinary practice.

  → This was achieved through ensuring the designed model was well modularised, allowing for fast and intuitive adjustments. Having standardised preprocessing allows for the adjustment of the model to new datasets with a similar binary classification task.

4 Data

The dataset that was sourced for the group's goals contained 827 images of bees, sourced from RoboFlow [4], and labelled with (`has_varroa_mite` $\in$ `{True, False}`) as the class. Each image contains a single bee, usually centred with all body parts visible, and with any mites clearly visible.

The most obvious limitation of the dataset was the lack of images. The dataset used for the model presented in this report was the largest and highest quality the team was able to find, though a number of smaller and worse quality image sets were considered, the selected dataset was easily the largest and best option. To combat the lack of samples, image augmentation as discussed in section 5.3 was performed.

Overall, the data set is highly relevant to the project's primary objective. As the data-set is focused around identification of Varroa mites within honey bee populations, it is an ideal base to develop a detection model on. Below show example images from the dataset, with a Varroa mite clearly outlined.

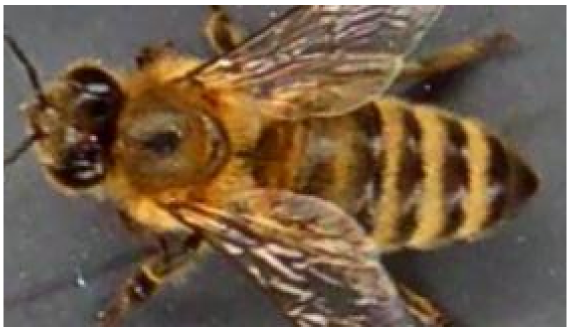


Figure 1: An example image of a bee with `has_varroa_mite = False`

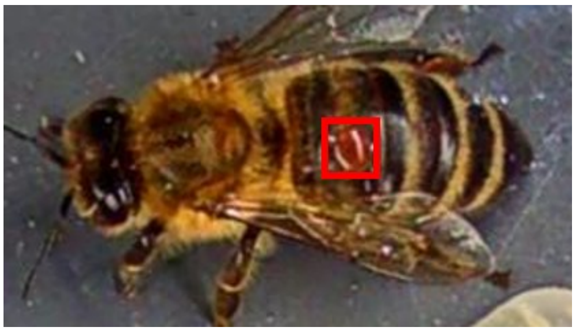


Figure 2: An example image of a bee with `has_varroa_mite = True`

5 Methodology

To develop an accurate and scalable system for detecting Varroa mite infestations, the group approached the project in three main stages: model selection and design, data pre-processing and augmentation, and model training and evaluation.

The group started by evaluating suitable machine learning approaches and determined that a Convolutional Neural Network (CNN) was best suited to our classification task. After reviewing several architectural options, a custom architecture that balanced performance with efficiency and aligned with the project scope was selected.

Given the limited dataset, the first priority was finding data augmentation techniques such that the database was effectively expanded and more manipulable. Then, various pre-processing transformations were applied to the dataset to simulate real-world image conditions. This allowed the assessment of the model's robustness under common deployment challenges such as lighting and temperature variation or lighting changes.

Finally, we iteratively trained and refined our model using cross-validation, holdout testing, and early stopping. These steps ensured reliable performance metrics and minimised overfitting. The following sections outline the methodology in detail, including design justifications, code snippets, and discussion of trade-offs encountered.

5.1 Model Selection and Design

5.1.1 CNN Justification

Given the spatially localised nature of the Varroa mites on the bee images in the dataset, and the goal of pattern-based classification, a Convolutional Neural Network (CNN) was selected as the ML architecture of choice. CNN's are well suited to image classification tasks due to their ability to learn hierarchical features from pixelated data, which aligns perfectly with the dataset and problem at hand.

In terms of trade-off of differing CNN approaches, the group landed upon Image Classification rather than Object Detection for a few reasons. Firstly, it was deemed that the crux of the problem was determining the *presence* of a mite, not *whereabouts* the mite sits. This extra information would not improve the treatment of an infected hive and thus isn't useful. Further, an Image Classifier would have faster convergence, be less computationally expensive, and be more suitable for the binary prediction task: `has_varroa_mite` ∈ {`True`, `False`}.

5.1.2 CNN Architecture

For the requirements of the project, a custom CNN architecture was defined using TensorFlow (Figure 3). The primary goals of the model for which the architecture was designed are as follows:

- Reduce over-fitting due to the small dataset
- Computationally efficient, to maximise the teams ability to train and test iterations
- Apt at identification of small, non-contrasting objects within an image

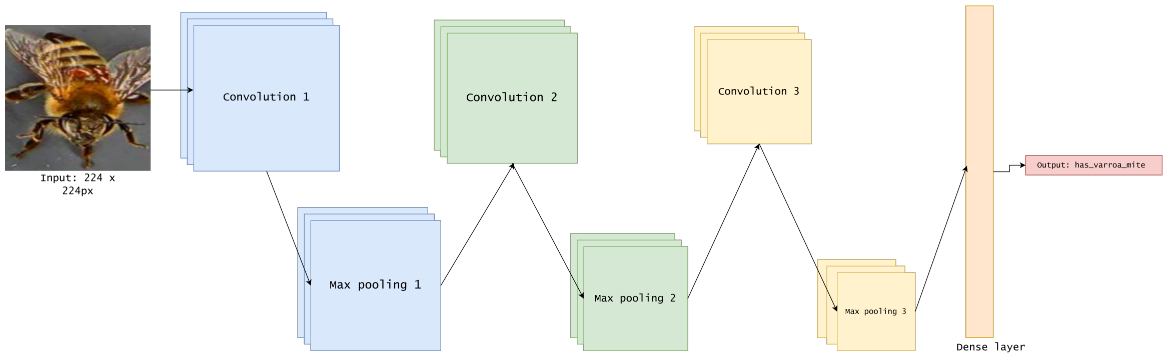


Figure 3: Simplified schematic of custom Bee-Gone CNN architecture

Convolutional Layers

The chosen architecture employees three convolutional layers, striking a balance between over-fitting the data, and providing enough filtering to allow identification of fine details, as outlined by the above goals of the architecture. Each convolutional layer has different filtering behaviour, which is passed as input to the next. This allows each layer to identify different features of the model, including patterns, textures and eventually fine details such as entire parts of an object. The first layer (32 filters, 3 x 3 kernel size) has the key function of detecting lines and edges within the image, allowing the model to learn low-level features. The next (64 filters, 3 x 3 kernel size) addresses mid-level features, primarily object parts. Taking this information, the final layer (128 filters, 3 x 3 kernel size) is able to detect the high-level features of the model, being full object patterns. Three layers were selected to provide enough filtering to identify complex features and relations, without risking model over-fitting, especially given the limited dataset.

Max Pooling Layers

Between each convolution, there is a Max-Pooling layer. The specific purpose of each varies, but they all serve to reduce spatial detail and dimensions from the original 224 x 224px, increasing efficiency and controlling over-fitting, aligning with the previously outlined architecture goals. Additionally as the goal of the model is Image Classification, not Object Detection, there is no need for accurate location identification, so the level of spatial reduction is appropriate. The specific goals of each pooling layer are as follows:

1. Reduces the image resolution while retaining strong features. It achieves this by down-sampling the feature map by retaining the maximum value in each 2 x 2px window (Figure 4).
2. Same goal and application as 1.
3. Applies a final spatial compression. Now the dimensions become smaller, and the feature maps deeper.

Three pooling layers allows the architecture to achieve these features without losing too much spatial detail and affecting the models ability to identify small objects.

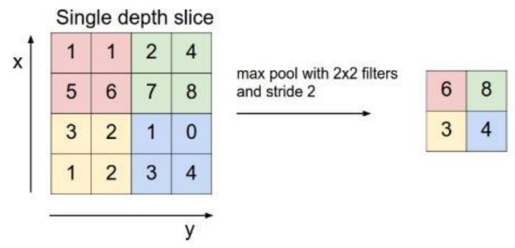


Figure 4: Schematic representation of a max pooling layer identifying strongest features and downsizing

Dense Layer

After applying the aforementioned layers, the 3 dimensional output of the previous pooling layer is converted into a 1 dimensional vector. This is then fed into the dense, or fully connected layer. This layer acts like a traditional neural network layer, by mapping features to intermediate representations for a final binary classification. It achieves this by learning complex combinations of the high-level features and making a decision based on them.

Output Layer

This final layer consists of a single neuron that performs the binary classification of mite presence using the Sigmoid activation function, which is well-suited for binary classification tasks. It takes the outputs from the previous dense layer and applies the Sigmoid function to map the result to a probability between 0 and 1. This probability represents the likelihood that the input belongs to class 1 - `has_varroa_mite = True`.

5.1.3 Regularisation

Regularisation was employed for use in the model to prevent over-fitting. The method used for the team's custom architecture is EarlyStopping, which monitors the validation loss (val-loss) during the training process. It prevents over-fitting by stopping training when the model stops improving for a given number of epochs. This is known as the patience value and was variable and iterated upon during model refinement (5.3). Additionally, it restores the best weights and does not keep weights from a worse epoch when over-fitting does begin.

5.1.4 Optimiser

The chosen optimiser for this architecture was Adam (Adaptive Moment Estimation). The optimiser here has the primary goal of updating weights based on the current error, using gradients to adjust them to reduce loss when continuing training. Adam was selected due to its use for image based CNN, and its adaptivity to different layers and parameters. This is ideal for out-of-the-box use with a custom CNN architecture. Another considered Optimiser was Stochastic Gradient Descent (SGD), which is better for large-scale data compared to Adam, however Adam boasts a faster convergence, at the risk of generalising worse in some cases. For the purposes of this model (small dataset, custom architecture), Adam was considered more suitable.

5.1.5 Parameters and Hyper-parameters

Parameters are determined internally by the model, and for this architecture, mainly include the weights/biases of each of the aforementioned layers. These parameters are updated each epoch during training. The best weights are then restored after each fold.

The hyper-parameters utilised control the accuracy of the model, and were iterated upon by the team throughout the validation process of the model. Some critical hyper-parameters include our validation-data split ratio, number of k-folds, patience, layer parameters (filer numbers, kernel size), epochs, and batch size. These hyper-parameters are discussed in depth in 5.3.

5.2 Data Augmentation and Preprocessing

5.2.1 Data Augmentation

To address the glaring issue of the limited size of the labelled dataset (827 images), a simple yet effective augmentation pipeline was implemented in the model. Each original image was horizontally flipped, and the result rotated in 90 degree increments, which eventuated in 4 additional variation of the original image. This allowed the group to increase the dataset five-fold, increasing resources. This increase is justifiable as it simulates the variability in images that is expected from a static camera capturing bees from different angles, if this model were to be physically deployed. This improves the model's generalisation without introducing unrealistic transformations for the sake of increasing the amount of images in the dataset.

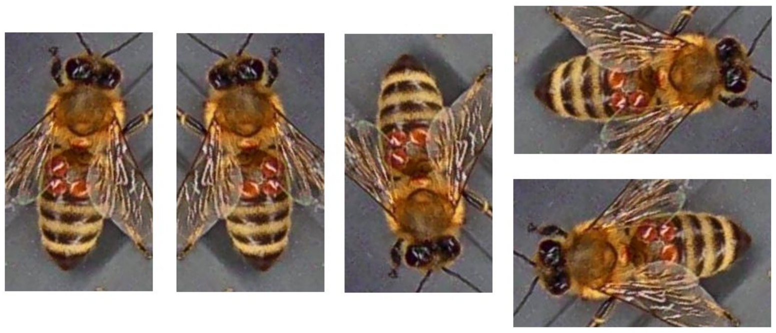


Figure 5: The original image (leftmost) followed by the 4 variations through flip and rotations.

5.2.2 Data Preprocessing

In preparation for testing the robustness of the model, a range of quality-altering preprocessing techniques were applied to simulate real world deployment conditions, such as variable lighting and motion blur. The different strategies were informed by basic colour and photographic theory, serving the primary purposes of improving classification accuracy. The image transformations included various levels of:

1. Temperature control - Redshift and Blueshift
2. Increased brightness
3. Increased contrast
4. Gaussian blurring

These various perturbations allowed controlled experiments on the model's sensitivity to image conditions, and allowed the group to ascertain any beneficial modifications, if any existed. In addition, all images were resized to 224 x 224 pixels and normalised to a 0–1 pixel value range to ensure compatibility with the model input.

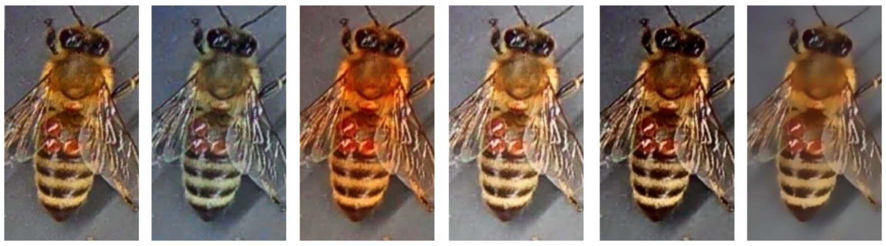


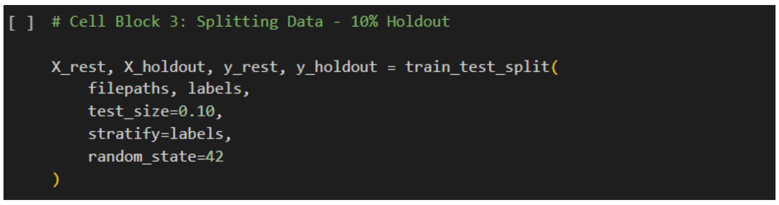
Figure 6: The original image (leftmost) followed by blueshifted, redshifted, brightened, contrasted and blurred preprocessed variations.

5.3 Final Model Overview

The model was iteratively developed from a basic prototype to a more complex product as insight was gained. Early experiments confirmed the feasibility of Image Classification, while later iterations added regularisation and strategies to improve generalisation. The final model has its attributes described below:

5.3.1 Holdout Set

Before performing the cross-validation, 10% of the augmented dataset of 4145 images was reserved as a true holdout set and excluded entirely from model selection and hyperparameter tuning. This set acted as a final, unbiased benchmark for evaluating generalisation, after training the final model. The remaining 90% of the data was used exclusively within the cross-validation framework for training, validation, and performance averaging.



```
[ ]  # Cell Block 3: Splitting Data - 10% Holdout

     X_rest, X_holdout, y_rest, y_holdout = train_test_split(
         filepaths, labels,
         test_size=0.10,
         stratify=labels,
         random_state=42
     )
```

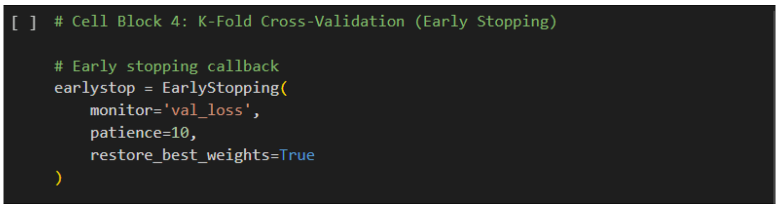
Figure 7: Code snippet that reserves 10% of the data as holdout.

5.3.2 K-Fold Cross Validation - Stratification

To rigorously assess model performance and minimise bias due to random sampling, the training data was evaluated using k-fold cross-validation with $k = 10$. Stratified splitting was employed to ensure that each fold preserved the proportion of `True` and `False` classes, maintaining balanced representation across subsets. For each fold, 10% of the data was reserved for testing, while the remaining 90% was further split into training and validation sets using an 80/20 ratio. This resulted in 72% of the total dataset used for training and 18% for validation in each fold. A k value of 10 is a standard value, which strikes a balance between a larger k resulting in more computation time, and a smaller k risking instability.

5.3.3 Early Stopping Condition - Val Loss

To prevent over-fitting during training, an early stopping mechanism was implemented. The val-loss was monitored across epochs, and the fold was halted if the val-loss did not improve for 10 consecutive epochs - the patience value. The model's weights were restored to those from the epoch with the lowest val-loss, ensuring optimal generalisation performance. This approach also allowed the identification of the best epoch for each fold, which was then averaged to guide the final model's evaluation. The trade-off of selecting a patience value is the balance between a short patience risking model under-fitting, and a longer patience increasing training time and leading to over-fitting. For the purpose of this model, a patience of 10 was found to be a reasonable compromise, balancing the limitations of hardware for training and reducing over-fitting.



```
[ ] # Cell Block 4: K-Fold Cross-Validation (Early Stopping)

    # Early stopping callback
    earlystop = EarlyStopping(
        monitor='val_loss',
        patience=10,
        restore_best_weights=True
    )
```

Figure 8: Code snippet showing the early stopping condition features such as 10 patience, weight restoration, and val-loss monitoring.

5.3.4 Model Evaluation - Mean Epochs

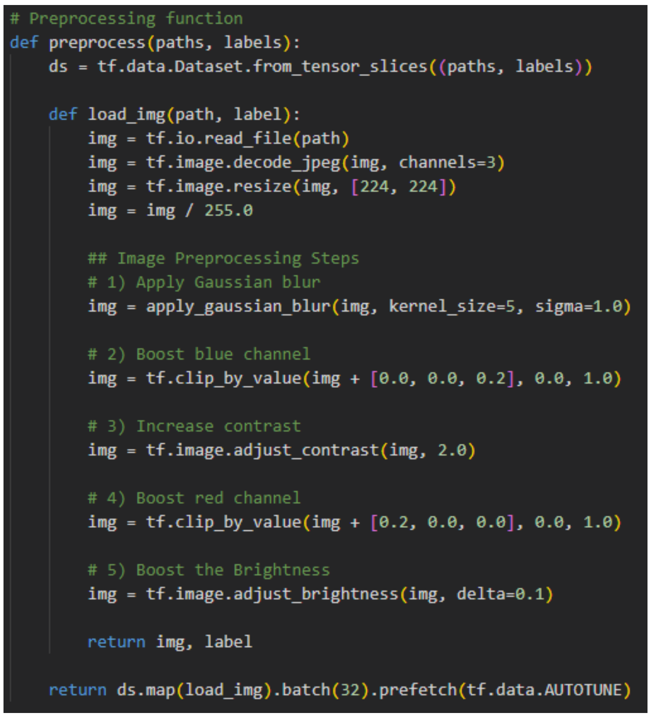
After cross-validation, the optimal number of training epochs was determined by averaging the best-performing epochs across all 10 folds. This value, `ep_mean`, was used for a final training on the entire 90% of non-holdout data, without any validation split or early stopping. The final trained model was then evaluated once on the holdout set, providing a realistic estimate of deployment performance.

5.3.5 Performance Reporting

Model performance was assessed using accuracy, precision, recall, and F1-score, computed on the cross-validation folds and the final holdout set. Standard deviation across folds was also reported to indicate model stability. Finally, a ROC curve of the model was also graphed, indicating the AUC of the final model. These metrics allowed the group to evaluate the model's performance on the dataset, whether preprocessed or not.

5.3.6 Preprocessing Integration

Pre-processing was integrating into the model via an augmentation pipeline applied consistently to all folds (training, validation, and testing), this includes the aforementioned normalisation and image resizing. A dedicated method for toggling various pre-processing strategies (Figure 9) allows for toggle-able pre-processing strategies depending on what combination is to be investigated to test the model.



```
# Preprocessing function
def preprocess(paths, labels):
    ds = tf.data.Dataset.from_tensor_slices((paths, labels))

    def load_img(path, label):
        img = tf.io.read_file(path)
        img = tf.image.decode_jpeg(img, channels=3)
        img = tf.image.resize(img, [224, 224])
        img = img / 255.0

        ## Image Preprocessing Steps
        # 1) Apply Gaussian blur
        img = apply_gaussian_blur(img, kernel_size=5, sigma=1.0)

        # 2) Boost blue channel
        img = tf.clip_by_value(img + [0.0, 0.0, 0.2], 0.0, 1.0)

        # 3) Increase contrast
        img = tf.image.adjust_contrast(img, 2.0)

        # 4) Boost red channel
        img = tf.clip_by_value(img + [0.2, 0.0, 0.0], 0.0, 1.0)

        # 5) Boost the Brightness
        img = tf.image.adjust_brightness(img, delta=0.1)

        return img, label

    return ds.map(load_img).batch(32).prefetch(tf.data.AUTOTUNE)
```

Figure 9: Code snippet demonstrating the application of preprocessing to dataset

6 Findings

6.1 Model Improvement

Firstly, a clear distinction can be seen between the group's original CNN model and the final product, after applying the model refinements and training strategies outlined in Sections 5.3.1 through 5.3.4. The improvement is evident when comparing the ROC curves generated and visible in Figures 10 and 11 below:

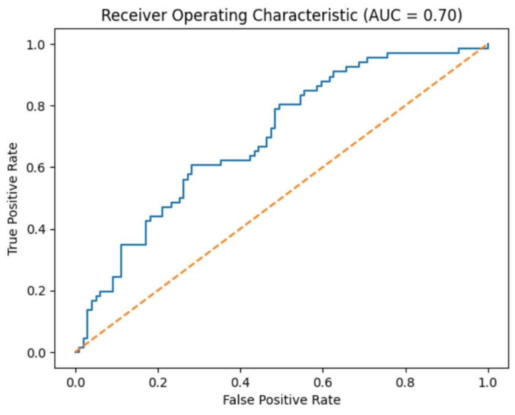


Figure 10: Preliminary model ROC, with an AUC of 0.70

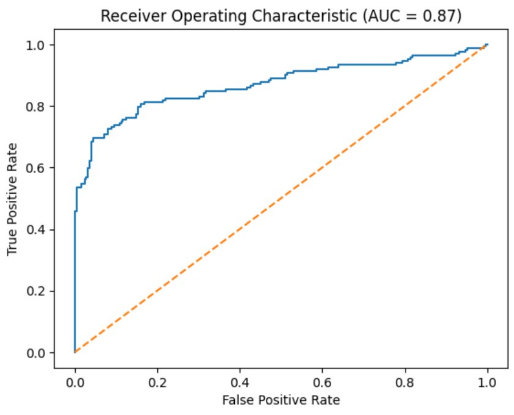


Figure 11: Final model ROC, with an AUC of 0.97

The most immediate improvement is the increase in AUC from 0.70 to 0.87. This 0.17 gain reflects a significant boost in the model's ability to distinguish the two classes. A higher AUC indicates better performance across all classification thresholds, meaning the model is more robust in balancing false positives and false negatives.

Further, the final model's ROC curve is smoother compared to the initial model's more stepwise curve. This indicates that the final model outputs a finer-grained and better-calibrated distribution of prediction probabilities. This predictive resolution is likely due to the increased dataset size from data augmentation as well as the aforementioned techniques.

6.2 Parameter/Hyper-Parameter Evaluation

Table 1: Impact of Early Stopping on 10-Fold Accuracy

Patience	Accuracy	Variance
N/A	0.6411	0.0586
3	0.6029	0.0000
5	0.6024	0.0005
10	0.6430	0.0529

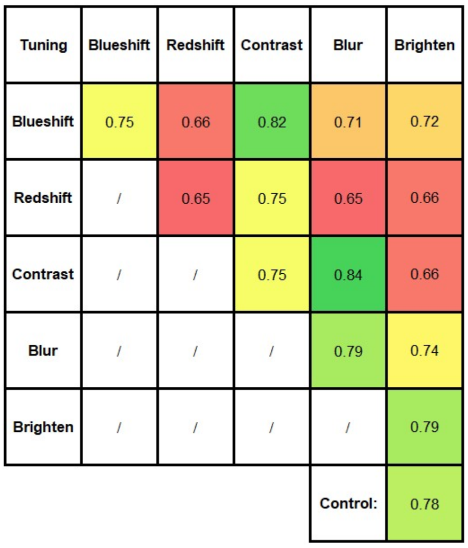
Section 5.3.3 discussed early stopping, with patience being a key hyper-parameter that is directly proportional to the training time of the model. With time constraints being a limiting factor of this project, the goal for optimising patience was to achieve greater than or equal to accuracy with similar variance to the base case.

From table 1, it is observed that the lack of an early stopping condition resulted in a base accuracy of 0.6411. By definition, an early stopping condition saves computation power at the trade-off of lost learning, thus, a patience level of 3 showed an accuracy of 0.6029 with no variance. The lower accuracy served as evidence that the model was underfitting, and the lack of variance was an indicator that the convergence criteria were being met too soon. This notion is reinforced when experimenting with an early stopping of patience 5, which saw two accuracy convergences. A first at 0.6029 for the first 5 folds and a second at 0.6019 for the latter half; Resulting in a 0.6024 ± 0.0005. Albeit with a lower accuracy, this model showed indications of positive and negative learning, highlighting that some folds struggled more than others. Given that the overall accuracy was still lower than the no early stopping model by 4%, learning was still being lost, and as such, the patience needed to be increased to combat the underfitting. It was found that with a patience level of 10, the model saw an improvement in accuracy by 0.29%, while showing a lower variance of 0.0529. This was indicative of a patience parameter that replicated the same learning as the base case while saving resources on overfitting folds and delegating more epochs to underfitting folds.

Further patience levels were not explored, as increasing the patience level beyond 10 (which matched our requirements) would require more computational power while risking overfitting. This would ultimately delay our plans to further explore pre-processing recipes.

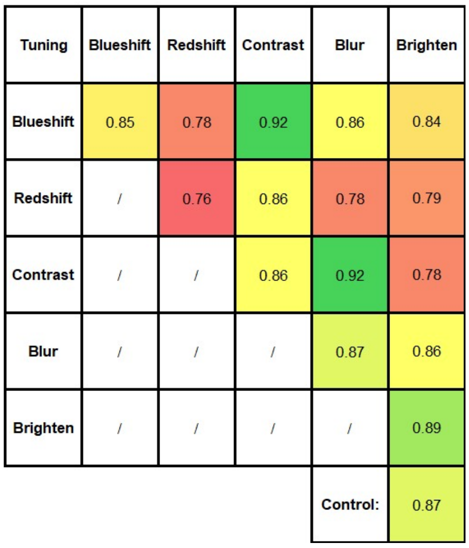
6.3 Robustness Evaluation

As mentioned in Section 5.1.2, the final model was further evaluated on a range of image preprocessing conditions to assess its robustness across different environments. Beyond individual filters, the model was also tested on all pairwise combinations of preprocessing techniques. Performance was assessed using both F1 score and AUC, as shown in the heatmaps in Figs. 12 and 13:



Tuning	Blueshift	Redshift	Contrast	Blur	Brighten
Blueshift	0.75	0.66	0.82	0.71	0.72
Redshift	/	0.65	0.75	0.65	0.66
Contrast	/	/	0.75	0.84	0.66
Blur	/	/	/	0.79	0.74
Brighten	/	/	/	/	0.79
				Control:	0.78

Figure 12: Heatmap of the final model's F1 scores on various preproccessing combinations



Tuning	Blueshift	Redshift	Contrast	Blur	Brighten
Blueshift	0.85	0.78	0.92	0.86	0.84
Redshift	/	0.76	0.86	0.78	0.79
Contrast	/	/	0.86	0.92	0.78
Blur	/	/	/	0.87	0.86
Brighten	/	/	/	/	0.89
				Control:	0.87

Figure 13: Heatmap of the final model' AUCs on various preproccessing combinations

A number of key insights emerge from these figures. Firstly, the wide range of scores demonstrates the significant influence that preprocessing has on the model's effectiveness. While some pipelines lead to performance below that of the control (i.e., no preprocessing), others result in substantial improvements. For instance, the combination of Contrast + Blur and Contrast + Blueshift achieved AUCs of 0.92 — notably higher than the control's 0.87 — while other combinations such as Redshift + Blur performed worse, with AUCs as low as 0.66.

This range highlights that no single preprocessing recipe will suit all deployment scenarios. The model's performance is highly dependent on the environmental conditions represented by the client's image data. For example, Redshift consistently degraded performance — which aligns with the fact that Varroa mites are red-brown in colour. However, in a "bluer" environment, Redshift may actually be beneficial.

These findings suggest that in deployment, the preprocessing pipeline should be tailored to the specific characteristics of the client's image environment. While theoretically a poor preprocessing choice could degrade performance close to random guessing (AUC $\approx$ 0.5), the right preprocessing strategy could achieve near-perfect classification (AUC $\approx$ 1.0). Therefore, to maximise model robustness across diverse use cases, a preprocessing calibration step — informed by sample data from the client — should precede deployment.

7 Discussion

7.1 Performance and Impact of Trade-offs on Results

Upon inspection of Figures 11, 13, and 12, we observe that the model has a baseline AUC/F1 score combination of 0.78 and 0.87 respectively, and a best performance of 0.92 and 0.84. Overall, the baseline model is relatively successful at achieving the goal of high accuracy of identification as outlined in Section 3, though performance is greatly enhanced by pre-processing, with improvements discussed below.

The trade-offs mentioned throughout Section 5 are mainly concerned with striking a balance between the limited computational power and training time availability, the risk of over-fitting due to the small dataset, and still achieving a high accuracy. Overall, through the hyper-parameter choices and tuning outlined above, a balance was struck which allows the team to meet the project's design goals with the resources available. Future development of the model, with access to increased resources and extended training time, may revisited the compromises made to further improve the model.

7.2 Improvements

As outlined in Section 5.3, significant tuning of hyper-parameters allowed for large optimisations to the baseline model, thus a larger limiting factor of accuracy of the final model is likely attributed to the small dataset, as referenced in Section 4. Identification and implementation of the model with a larger and higher quality dataset would be the first recommendation for improvement, and would allow for better identification of further hyper-parameter tuning requirements and improvements.

While the final model was able to achieve a best case result of 0.92 and 0.84 through the employment of varied pre-processing strategies, there are still improvements to be made to increase the models accuracy and thus applicability. As pre-processing of data greatly affects the accuracy and precision of the model, further experimentation with pre-processing strategies and combinations should be investigated to explore the possibility of even better combinations. This may include the use of more than two strategies at a time, or the investigation of completely different strategies. Moreover, further exploration into the strengths of each strategy could yield potential gains. Lastly, at the cost of computing power, experimenting with higher patience levels may yield further minor gains, while most lie within the strategies applied.

7.3 Deployment and Ethical Considerations

There are a number of deployment considerations to be made for this model to be viable in commercial apiaries or similar applications. The proposed deployment strategy is for consumers to place high speed cameras at beehive exits, with the camera taking bursts of photos in a desired increment over time, providing an insight into the average behaviour of the hive. Thus, this project assumes the existence of intermediary software able to isolate a single bee from an image containing multiple, such that the input into the model is formatted in the same way as previously demonstrated in Section 5.2. In order to achieve an accurate result, another software must be capable of identifying what pre-processing is required of a specific environment, and apply it to images, or a sample must be sent to a managing team in order to determine the best way to process images. While this report presents successful results for implementation of the identification of mites on individual bees, further work must be conducted for successful deployment of the model in a commercially viable sense.

The described deployment strategy implies some ethical considerations which must be met before the distribution of the model. A primary concern is with the distribution and privacy of images used for identification. Privacy of staff in a commercial context, or members of the public, such as in a communal garden, must be considered. Risks of geo-tagging or GPS identification of private apiaries from images captured for the model must also be addressed. There must be regulation of model use, to ensure that over-reliance on the model does not occur, and that human intervention and hive check-ups still occur as required.

8 Conclusions

This report describes the development of a Machine Learning solution to Colony Collapse Disorder in honeybee populations caused by the Varroa Destructor. A Convolutional Neural Network solution was implemented using a custom architecture, resulting in a baseline Area Under Curve of 0.87 after extensive tuning of hyper-parameters and additional features. Pre-processing combinations evaluated the robustness of the model, and demonstrated the range of possible results based on the combination of pre-processing strategies used.

Overall, this project demonstrates the potential for commercial deployment of the model, outlining an efficient, accurate, and non-invasive solution to the identification of the Varroa Destructor. With further development, the model has potential to contribute to supporting honeybee populations, offering significant environmental, economic, and societal benefits.

References

[1] United Nations Environment Programme. “Why bees are essential to people and the planet.” (2023), [Online]. Available: https://www.unep.org/news-and-stories/story/why-bees-are-essential-people-and-planet (visited on 04/18/2025).

[2] ABC News. “Beekeepers alarmed about australia’s food security and prices.” Accessed: 2025-04-19. (2024), [Online]. Available: https://www.abc.net.au/news/2024-11-17/beekeepers-alarmed-about-australias-food-security-and-prices/104589836 (visited on 04/19/2025).

[3] Agriculture Victoria. “Testing to detect varroa mite.” Accessed: 2025-04-19. (2024), [Online]. Available: https://agriculture.vic.gov.au/biosecurity/pest-insects-and-mites/priority-pest-insects-and-mites/varroa-mite-of-honey-bees/testing-to-detect-varroa-mite (visited on 04/19/2025).

[4] TensorFlow. “Image classification.” Accessed: 2025-04-19. (2024), [Online]. Available: https://www.tensorflow.org/tutorials/images/classification (visited on 04/19/2025).

[5] Keylabs. “Image classification techniques for different tasks: E.g. object detection, scene recognition.” Accessed: 2025-04-19. (2024), [Online]. Available: https://keylabs.ai/blog/image-classification-techniques-for-different-tasks-e-g-object-detection-scene-recognition/ (visited on 04/19/2025).

[6] Label Your Data. “Object detection vs. image classification: What’s the difference?” Accessed: 2025-04-19. (2024), [Online]. Available: https://labelyourdata.com/articles/object-detection-vs-image-classification (visited on 04/19/2025).

[7] Viso Suite. “What is object detection? a complete guide.” Accessed: 2025-04-19. (2024), [Online]. Available: https://viso.ai/deep-learning/object-detection/ (visited on 04/19/2025).

[8] Neptune AI. “Image segmentation: A quick introduction.” Accessed: 2025-04-19. (2024), [Online]. Available: https://neptune.ai/blog/image-segmentation (visited on 04/19/2025).

[9] SuperAnnotate. “A guide to semantic segmentation: Techniques and applications.” Accessed: 2025-04-19. (2024), [Online]. Available: https://www.superannotate.com/blog/guide-to-semantic-segmentation (visited on 04/19/2025).

[10] Analytics Vidhya. “Understanding transfer learning for deep learning.” Accessed: 2025-04-19. (2021), [Online]. Available: https://www.analyticsvidhya.com/blog/2021/10/understanding-transfer-learning-for-deep-learning/ (visited on 04/19/2025).

[11] Geeks for Geeks. “Transfer learning with convolutional neural networks.” Accessed: 2025-04-19. (2024), [Online]. Available: https://www.geeksforgeeks.org/ml-transfer-learning-with-convolutional-neural-networks/ (visited on 04/19/2025).

[12] yolov5 object detection on honey bees, *Varroua mite detection in honey bees dataset*, Open Source Dataset, visited on 2025-04-19, Oct. 2022. [Online]. Available: https://universe.roboflow.com/yolov5-object-detection-on-honey-bees-u8kya/varroua-mite-detection-in-honey-bees.

[13] TensorFlow. “Convolutional neural network (cnn).” Accessed: 2025-04-19. (2024), [Online]. Available: https://www.tensorflow.org/tutorials/images/cnn (visited on 04/19/2025).

[14] PyTorch. “Training a classifier on cifar10.” Accessed: 2025-04-19. (2024), [Online]. Available: https://pytorch.org/tutorials/beginner/blitz/cifar10_tutorial.html (visited on 04/19/2025).

[15] OpenCV. “Pytorch vs. tensorflow: A detailed comparison.” Accessed: 2025-04-19. (2024), [Online]. Available: https://opencv.org/blog/pytorch-vs-tensorflow/ (visited on 04/19/2025).

[16] Simplilearn. “Keras vs tensorflow vs pytorch: What’s the difference?” Accessed: 2025-04-19. (2024), [Online]. Available: https://www.simplilearn.com/keras-vs-tensorflow-vs-pytorch-article (visited on 04/19/2025).