An Evaluation of Classical and Deep Learning Approaches for Pest Detection and Classification

Arindam Mukherjee
Computer Science and Engineering
University of New South Wales
Sydney, Australia

Raul Munif
Computer Science and Engineering
University of New South Wales
Sydney, Australia

Suraj Uppal
Department of Engineering
University of New South Wales
Sydney, Australia

Djimon Jayasundera
Computer Science and Engineering
University of New South Wales
Sydney, Australia

Mario Krivosic
Computer Science and Engineering
University of New South Wales
Sydney, Australia

I. Introduction

It is without a doubt that healthy and productive agriculture is quintessential to humans and for global economic growth, however, the increasing damage to agriculture as a result of insect pests are making it difficult to maintain. Through this study, we made use of a range of unique supervised Machine Learning (a sliding-window HOG+SVM detector with Hard Negative Mining, an ORB+BoVW detector, and a SIFT+Bag-of-Visual-Words (BoVW) classifier with both SVM and k-Nearest Neighbour (kNN) back-ends) and Deep Learning (YOLOv12 variants and Faster R-CNN with a ResNet-50 backbone) methods in order to test the effectiveness of both detecting and classifying insects across the small-sized AgroPest-12 dataset. This dataset consists of 11502 training, 1095 validation and 546 testing images all with 12 different classes of insects along with the necessary ground truth class labels and bounding boxes. As such, by training our models on the training dataset and tweaking hyperparameters against the validation set, we aimed to compare and improve the effectiveness of these industry-grade methods in detecting and classifying these insects such that we could better protect our agriculture.

II. Literature Review

Pest recognition is a critical agricultural issue for farmers due to crop destruction leading to diminishing economic returns (Malathi & Gopinath, 2021). Furthermore, manual identification is labour-intensive and error-prone, largely due to the diversity of species, complexity of life cycles, and growth in invasive pest numbers (Wu et al., 2019). As such, work has progressed in automating these difficult tasks with both machine-learning handcrafted methods and, more recently, deep-learning strategies (Wu et al., 2019).

Early approaches relied heavily on handcrafted features combined with classical feature classifiers (e.g., support vector machines [SVM], k-nearest neighbours [KNN]). For example, authors benchmarking the IP102 dataset for insect pest recognition tested a large number of such methods, including colour histograms, Gabor filters, GIST, SIFT, and SURF (Wu et al., 2019). They found that even the most modern handcrafted methods, such as SURF (Bay et al., 2008), achieved only 19.5% accuracy, suggesting that local descriptors alone are insufficient to capture the fine-grained inter-species differences present in large, diverse pest datasets. These limitations were diminished in more sophisticated bio-inspired pipelines. Deng et al. (2018) explored one such pipeline, in which a saliency model (SUN) based on natural image statistics was extended with an HMAX hierarchy, combined with dense SIFT descriptors, non-negative sparse coding (NNSC), and local configuration pattern (LCP) features, and classified through SVM. This method achieved 85% accuracy on its target dataset and outperformed other baseline handcrafted methods. To achieve this, however, a very complex and dataset-specific pipeline had to be crafted, and it was still shown to be sensitive to templates. Work on linear spatial pyramid matching with sparse coding further demonstrated that handcrafted methods can be improved via more structured pooling and coding schemes (Bao et al., 2015). Nevertheless, all of these approaches depend on engineered descriptors, which are prone to overfitting due to their specialised nature (Wu et al., 2019).

The advent of deep learning has altered the pest classification landscape. In the same IP102 paper, authors fine-tuned multiple pre-trained CNN architectures with varying hyperparameters, and also combined CNN feature extractors with SVM and KNN by removing the final classification layer (Wu et al., 2019). Across these experiments, deep-learning methods consistently outperformed handcrafted methods. ResNet-based models provided the strongest feature quality, and SVM outperformed KNN when used as the final classifier—consistent with the handcrafted baselines, which similarly showed SVM as more effective than KNN (Wu et al., 2019). Traditional handcrafted methods suffered from low recall and G-mean, highlighting their inability to handle class imbalance and subtle inter-class variability, whereas CNN methods achieved substantially higher values on both metrics and proved more robust to imbalance. The IP102 results therefore established deep CNNs as the de facto baseline for fine-grained pest classification.

Subsequent work focused on larger, higher-quality datasets and more complex architectures to increase performance. The AP162 dataset introduced by Wang et al. (2025) was designed to address limitations of previous datasets, such as limited sample sizes, inconsistent quality, and mislabelling. To mitigate these issues, AP162 employed a rigorous seven-step annotation pipeline. The dataset also used a dual organisation into agricultural-based and vision-based subsets, enabling both domain-informed and purely visual analyses of model performance. Six architectures were benchmarked—ResNet, DenseNet, MobileNetV3, ViT, Swin Transformer, and ConvMixer—all initialised with ImageNet weights. Contrary to Wu et al.'s (2019) findings, Swin Transformer models were the best-performing, which was attributed to their hierarchical transformer design with shifted windows that capture both local fine details and broader global context. More lightweight architectures such as MobileNetV3 performed substantially worse, suggesting that pest classification requires fine-grained recognition that benefits from high representational capacity (Wang et al., 2025).

Building on these results, Wang et al. (2025) proposed a weighted fusion strategy that ensembled three Swin Transformer-Base models: one trained on agricultural subsets, one on vision-based subsets, and one on the full AP162 dataset. At inference, combining predictions from these models produced 89.2% accuracy, outperforming all single models and showcasing the benefits of ensembling deep architectures. Interestingly, classification on the vision-based split was more challenging than on the agricultural split, highlighting that minute visual differences between pests remain the largest challenge even for advanced architectures (Wang et al., 2025).

Similar to ensemble approaches, Malathi and Gopinath (2021) explored how transfer learning can achieve strong results with minimal training. In a study on paddy crop pests, they fine-tuned a ResNet-50 model on a targeted dataset and, by leveraging transfer learning, achieved approximately 95% accuracy. Although this result exceeds AP162 and IP102 benchmarks, the dataset was narrower in scope, being restricted to paddy pests. Even so, the study illustrates the value of ImageNet-pretrained backbones as strong initialisations for pest recognition models.

Shifting from pest classification to detection, AgriPest highlighted the difficulties of detecting small pests in cluttered field regions (Wang et al., 2021). When insects occupied less than 13% of the image, two-stage region-based classifiers (e.g., Faster R-CNN, FPN, Cascade R-CNN) consistently outperformed one-stage detectors (e.g., RetinaNet, SSD). Despite detection being distinct from classification, this study reinforces the need for models that can resolve fine-grained details in pest imagery (Wang et al., 2021).

Across the literature, pre-processing and data augmentation play a critical role in success. Typically, images are resized from diverse sensors to a fixed resolution while preserving aspect ratio via padding, followed by pixel normalisation. Data augmentation is widely used to increase robustness and reduce overfitting, employing geometric transformations, photometric changes, and colour-space conversions. In some pipelines, segmentation methods isolate pest regions from background clutter, either via classical techniques or deep segmentation models. Standard evaluation metrics include precision, recall, and F1-score for classification, with additional metrics such as G-mean and micro-averaged AUC used to account for class imbalance.

Overall, the literature shows a clear progression from handcrafted feature-based machine learning to deep learning on large, professionally annotated datasets, with modern transformer-based architectures and ensemble strategies achieving the strongest performance when paired with transfer learning for initialisation. Nonetheless, several challenges remain: accurate classification of visually similar species, robustness to complex backgrounds and varied imaging conditions, and effective handling of multi-class scenes with overlapping pests. These gaps motivate continued exploration of powerful deep architectures, sophisticated data augmentation and fusion strategies, and high-quality domain-specific datasets for reliable pest classification in real-world agricultural settings.

III. METHODS

We implemented a total of 5 methods. These were both consisting of renowned Deep Learning and Machine Learning methods.

A. SVM using HOG features and HNM

The first Machine Learning method we implemented was a sliding-window object detector based on Histogram of Oriented Gradients (HOG) features and a Linear Support Vector Machine (SVM). Additionally, we designed a second version of this model which underwent additional training using Hard Negative Mining (HNM). The system processes images using a multi-scale pyramid approach and refines detections using Non-Maximum Suppression (NMS).

1) Feature Extraction: We utilised HOG features to capture morphological structure of the images. In order to leverage the information contained within the colour of specific insect species, HOG features were compute independently on each RGB channel and concatenated together. We utilised a patch size of 64×64 pixel, with 9 orientation bins, 6×6 pixels per cell, and $2 \times\ 2$ cells per block. This resulted in a feature vector of length 8748 per window.

2) Classifier Training: We employed a linear Support Vector Machine (SVM) trained via Stochastic Gradient Descent (SGD) with a hinge loss function. The reason we used an SGD approach was to facilitate out-of-core learning, allowing the model to be trained on the dataset in chunks to preserve memory. Class weights were balanced to address the disparity between different insect samples.

3) Detection Pipeline: Inference is performed using a multi-scale image pyramid with a scaling factor of 1.5. At each scale, HOG features are computed for the entire image, and a sliding window traverses the feature map. This optimisation avoids redundant feature calculations. Candidate detections are filtered using NMS with an Intersection over Union (IOU) threshold of 0.3 to merge overlapping bounding boxes.

4) Hard Negative Mining (HNM): To reduce false positives caused by background clutter such as foliage, we implemented a secondary training stage. The initial model was evaluated on the training set to identify "hard negatives", which were background patches incorrectly classified as insects with high confidence. The model was then re-trained on these specific samples to refine the decision boundary.

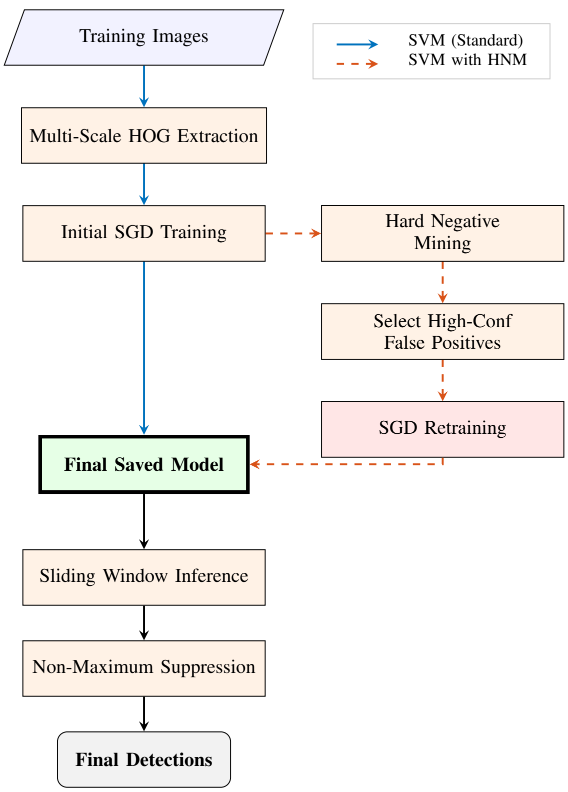

Fig. 1: Flowchart SVM using HOG features and HNM.

B. *ORB with BovW and SVM classifier*

The pipeline combines local keypoint descriptors, a Bag-of-Visual-Words (BoVW) representation, and a superpixel-based detection stage. During training, ORB descriptors are clustered to form a visual vocabulary, and BoVW histograms are computed for both ground-truth pest crops and synthetically generated background regions. At test time, superpixel-based proposals are classified using the trained SVM and refined with Non-Maximum Suppression (NMS).

1) Feature Extraction and BoVW Encoding: For descriptor extraction, each image is first read as a single-channel grayscale image. We apply a lightweight pre-processing pipeline to enhance local contrast and edge structure: Contrast Limited Adaptive Histogram Equalization (CLAHE) with an 8×8 tile grid, followed by Gaussian blurring and an unsharp mask. The unsharp mask is implemented by subtracting the blurred image from the CLAHE output and adding the result back with a moderate gain, which emphasizes high-frequency details to help with feature extraction.

keypoints and descriptors are then extracted using a single ORB detector configured to detect up to 1000 keypoints per image. When operating on ground-truth crops or superpixel proposals, if the region is very small, it is upscaled to a minimum side length (e.g. 32 pixels) using bi-cubic interpolation to ensure that ORB can still detect sufficient keypoints.

To obtain a fixed-length representation suitable for classification, we adopt a Bag-of-Visual-Words scheme. All ORB descriptors from the training set are pooled and randomly subsampled per image (up to a 100 per image) before being clustered using MiniBatchKMeans with $K = 400$ clusters. For each crop or region proposal, its descriptors are assigned to the nearest visual words, and a 400-dimensional histogram of word counts is computed. This histogram is then ℓ_2-normalised by dividing by its Euclidean norm, yielding a scale-invariant BoVW feature vector. These normalised BoVW vectors serve as the input features to the SVM classifier.

2) Classifier Training with Background Class: Ground-truth pest crops form the positive training examples. To teach the classifier to explicitly reject foliage, soil and other non-insect regions, we generate additional negative samples by randomly sampling bounding boxes in the training images and discarding any whose Intersection over Union (IoU) with a ground-truth box exceeds a small threshold (e.g. IoU ≥ 0.1). The remaining boxes are treated as background and assigned a dedicated class label.

Both pest and background crops are encoded using the same ORB–BoVW pipeline described above, producing an augmented training set with $C + 1$ classes (the original C pest categories plus extra background). We train a multi-class SVM with an RBF kernel on these features. Class-balanced weights are employed so that the loss contribution of each class is inversely proportional to its frequency, mitigating the strong imbalance between the abundant background features extracted and the rarer pest features.

3) Superpixel-based Detection Pipeline: At inference time, we first apply simple colour pre-processing including, white balancing, bi-lateral filtering and gamma correction. Then we conduct SLIC superpixel segmentation to the RGB image For each superpixel, we derive its tight axis-aligned bounding box and discard regions whose area is either too small or too large. The remaining boxes are treated as candidate object proposals.

Each proposal is then processed by the same ORB-BoVW encoder, the corresponding grayscale region is pre-processed, ORB features are extracted and visual words are predicted, forming a ℓ_2-normalised BoVW histogram. This feature vector is passed through the trained SVM to obtain class scores. Proposals predicted as background are discarded, and the remaining detections are filtered using a confidence threshold on the SVM score. Finally, class-agnostic Non-Maximum Suppression (NMS) with an IoU threshold is applied to merge overlapping detections and produce the final set of pest bounding boxes.

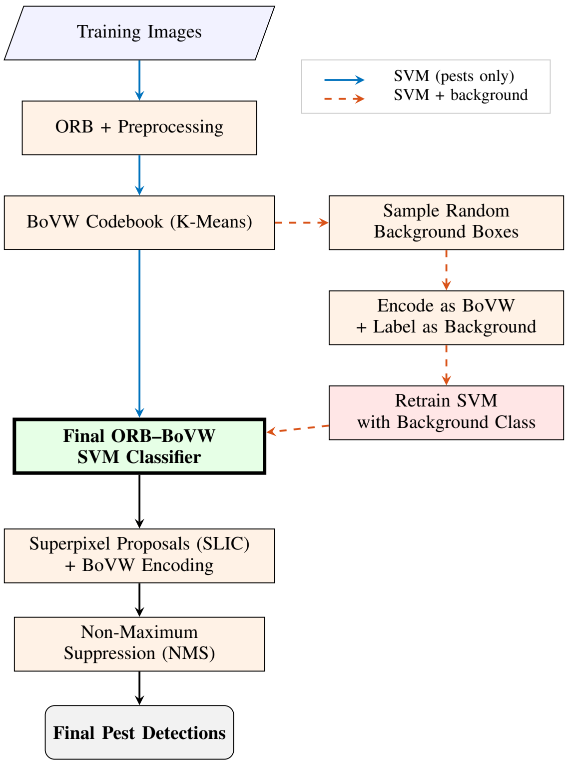

Fig. 2: Flowchart ORB using BovW features and superpixel segmentation

C. *SIFT with Bag-of-Visual-Words and SVM / kNN*

Additionally, we also developed another method using SIFT descriptors combined with a Bag-of-Visual-Words. The following method was primarily devised to try to maximise classification such that it could compete with its DL counterparts. Our method works by utilising the provided bounding-box annotations, where the input was converted to grayscale to be consistent as a result of removing colour disparities across classes. We then utilised SIFT to extract the 128-dimension descriptors and keypoints before they were pooled and subsampled. We then clustered these using MiniBatchKMeans with $K = 500$ clusters. Due to these cluster centres, we are able to create 500 visual words where each insect was assigned to the nearest visual words. This helped create a 500-dimensional histogram of word counts that when normalised, helped form a scale-invariant BoVW feature vector. We then used this feature vector to train an SVM and kNN classifier. For the SVM classifier, we used a one-versus-rest configuration with a hinge loss where the regularisation parameters was computed against the validation dataset to find the best value for C. For kNN, we tested several values for k where we ended up choosing a moderate neighbourhood size that balanced variance and bias. As such, with this method we aimed to keep it consistent with the deep learning approaches in order to accurately evaluate its effectiveness.

D. *Faster R-CNN with Resnet-50 Backbone*

We selected the Faster R-CNN method as one of the deep learning approaches. Since the task needed us to implement a detector and classifier, the Faster R-CNN method was a great tool to achieve this due to its double stage nature. The model builds on from the previous two iterations (R-CNN and Fast R-CNN) by introducing the Region Proposal Network (RPN). The RPN replaced the slow selective search algorithm which would split the input image to provide around 2000 region proposals to get a better indication of where the object is [9]. The RPN is a small network that slides over the feature map to predict the bounding boxes and objectness scores at each position [9]. This as a result leads to near cost-free region proposals, which when coupled with passing in this as an input to the 50-layer deep Convolution Neural Network once, allows us to generate shared convolutional features maps incredibly efficiently and accurately. Due to this efficiency and extensive research highlighting its accuracy in pest detection, this makes this a suitable option in theory to help detect and classify insect pests effectively.

We decided to utilise the pretrained weights trained on the COCO dataset rather than training it from scratch. This decision was made since we recognised the effectiveness of transfer learning in helping our model learn the coutours, edges and patterns from the COCO dataset, and applying it to insect detection. Moreover, training it from scratch would require significant training times, compute power and a significantly larger dataset which we didn't have access to. As a result, by using this pretrained approach that was also recommended by some of the literature, we aimed to devise an efficient and effective method which would tackle our problem.

We ensured to follow an iterative approach, where each iteration of the model addressed the flaws of the previous and built on it. These approaches include:

1) Optimiser Analysis: Made use of optimisers to adjust the models weights and bias to minimise the loss function. Initially, we went with the ADAM optimiser as through research it was a great general-purpose optimiser. However, we realised that it wasn't generalising as well as the Stochastic Gradient Descent (SGD) optimiser.

2) Data Augmentation: To further improve the models detection and classification capabilities, we decided to randomly apply data augmentation methods where we transform images to have Gaussian noise, changes in brightness and hue, rotations and horizontal flips. Since our dataset is already pretty small, adding these augmentation methods during training will considerably improve its performance when dealing with dynamic agricultural environments.

3) Extended training for improved convergence: Since the Faster R-CNN method has a complicated two stage architecture, we decided to train it over a maximum of 100 epochs with the complete dataset and implemented early stoppage in order to ensure we were efficiently using our resources to meet convergence.

4) Efficient hyperparameter tuning: Instead of adopting the conventional grid or random search to find the best parameters which minimise the loss on the validation dataset, we decided to use industry standard methods such as Optuna. As a result of setting up a 100 trials with multiple ranges for parameters such as learning rate, weight decay, batch size, etc., this in turn helped us find the best possible parameters to maximise the models' performance. This is incredibly useful given that the model takes around 7 - 10 hours to train on every iteration.

5) Anchor Box Modification: Since the Faster R-CNN model pretrained weights were initially trained on the COCO dataset, this meant that the sizes of these objects were fairly large and so were the anchor boxes. Anchor boxes are the bounding boxes that are placed over the whole feature map to inform the RPN on where to search for objects. From our data exploration we recognised that the AgroPest-12 dataset consists of images with very small pests (Min Area: 11.250 and Max Area: 409600.000) which the default anchor boxes in pytorch didn't fully cover. As such, we decided to modify this to better suit the bounding box distribution of our dataset to improve model performance.

6) Dataset Imbalance and Distortions: Since the AgroPest-12 dataset was already balanced, we decided to evaluate how robust the model was in a modified dataset that was undersampled and oversampled. Moreover, we decided to apply distortions such as increasing/decreasing the brightness, adding gaussian noise and blurs in order to simulate real world distortions in a dynamic agricultural setting.

7) Explainable AI using Attention Maps: To better understand our models' behaviour, we decided to display attention maps from the final convolution layer of the backbone in order to better understand exactly which parts of the image was our model focusing on and, to analyse points of success and failure to further guide our refinement decisions.

E. YOLOv12 Model Training and Oversampling Strategy

We selected the YOLOv12 family of models (YOLOv12n, YOLOv12s, and YOLOv12m) because of their efficiency, good results in detecting small objects, and the fact that they are capable of detecting agricultural pests in real time. YOLOv12 is an improvement to previous Ultralytics models containing enhancements in the interaction of backbone and neck and task specific detection heads. As with other Ultralytics implementations, YOLOv12 is based around early stopping based on a fitness metric, most of which is computed based on mAP:0.5:0.95, so that the model stops training when the large-object multi-IoU detectors have reached a good level of performance. To take advantage of transfer learning and avoid having to use extraordinarily large training set, all models were trained with pretrained COCO weights.

The strategy of training was an iterative, model scaling approach. Initially YOLOv12n was trained with a batch size of 16, a mosaic of 1.0 and no close mosaic to reach a performance baseline and early stopping was set to 50 epochs and a patience of 5. We then trained the YOLOv12s model with batch size 32, mosaic 0.75, close mosaic = 25 and early stopping with maximum epochs = 100 and patience = 10. Having compared YOLOv12s, we noticed that some classes of insects performed badly compared to others – especially as reflected in low recalls and mAP50 values.

To address this, we designed our metric-based strategy of oversampling using per-class performance of the YOLOv12s model on the test dataset. To ensure that our results were consistent across the different classes, we calculated a score given as

$$\textbf{Score} = 0.6 \times \textbf{mAP50} + 0.4 \times \textbf{Recall}.$$

The inverse of this combined score was taken as the oversampling weight. Duplicates of the sample with each of these weights were created and used to obtain oversampled images with controlled Albumentations transformations like Random Brightness Contrast (p = 0.4), horizontal flips (p = 0.5) and small-angle rotations (15° at p = 0.5) to ensure that the bounding box remained intact, but to maximize variation. At the same time, to avoid an undue distortion of small pests, we minimized mosaic augmentation, as suggested by test results using YOLOv12n.

Finally, the same training configuration as YOLOv12s (batch size, mosaic, close-mosaic and early stopping) was used to train the YOLOv12m model except that the new oversampled dataset based on the YOLOv12s test-set performance metrics was used to analyze the data components. This made the model the immediate beneficiary of the knowledge being learned in the previous reviews, which meant added strengths and a more balanced detection of all pest categories.

IV. Experimental ResultS

The experimental framework we setup for all our methods were standardised to ensure that we could produce results that we could directly compare the overall performance with. Given that we were provided test, train and validation datasets, we didn't decide to split the train dataset which allowed us to train our models on a larger set of images. As such, we used the train set to learn the model weights and parameters, validation set to tweak the hyperparameters and test set to evaluate the models' performance.

A. SVM using HOG features and HNM

This method was not suitable for object detection, due to the high rate of False Positives, even after applying Hard Negative Mining. The mAP score was approximately zero.

However, we used two alternative metrics to provide insights into the impact of HNM. Firstly was a Top-1 classification score, where we took the highest confidence detection and used it as a classification for the insect in the image. Secondly, was an Oracle classification score, where we fed the model an already cropped version of the image containing only the insect, and chose the highest confidence detection as its classification. This allowed us to evaluate its classification capability in isolation, separating it from any localisation error.

To test the generalisation capability of the classifier beyond the standard test set, we also measured its robustness by using five pipelines. These aimed to simulate common environmental effects found in agricultural settings. We compared both the baseline SVM and the HNM-refined SVM on the following variations:

- **Gaussian Noise:** Simulates sensor noise ($\mu = 0, \sigma^2 = 0.1$) typical in high-ISO or low-light photography.
- **Gaussian Blur:** Mimics focus errors or motion blur using a 5×5 smoothing kernel.
- **Low Brightness:** Simulates underexposure (e.g., shadows or dusk) by converting images to HSV space and reducing the Value channel by 40%.
- **High Contrast:** Simulates harsh, direct sunlight using Contrast Limited Adaptive Histogram Equalization (CLAHE) with a clip limit of 3.0.
- **Occlusion:** Simulates foliage obstruction by overlaying a random black patch covering 10% of the total image area.

TABLE I: Performance comparison of SVM models.

Pipeline	Time (s)	Oracle Acc	AUC
SVM (Std) - Ordinary	1064	0.289	0.728
SVM (HNM) - Ordinary	1064	0.228	0.719
SVM (HNM) - G. Noise	1095	0.073	0.571
SVM (HNM) - G. Blur	1069	0.208	0.714
SVM (HNM) - Low Bright	1047	0.206	0.712
SVM (HNM) - High Cont.	1051	0.216	0.702
SVM (HNM) - Occlusion	1051	0.171	0.680

The results demonstrated that the HNM-refined SVM model was highly robust to most of the modifications, including low brightness, high contrast, and blur. However, adding Gaussian Noise significantly degraded accuracy.

B. SIFT with Bag-of-Visual-Words and SVM/kNN

For the SIFT and BoVW pipeline we evaluated the linear SVM and a kNN classifier. Both models were trained on BoVW features extracted from the train dataset and had all hyperparameters tuned with the validation set. As such we got the following results:

Metric	SIFT + SVM	SIFT + kNN
Accuracy	0.421	0.208
Precision (macro)	0.393	0.315
Recall (macro)	0.414	0.201
F1-score (macro)	0.389	0.182
AUC (macro)	0.792	0.624

TABLE II: Global classification performance of the SIFT+BoVW models on the test set.

We can see that the SIFT + SVM configuration outperforms the kNN in all metrics. Specifically, with the 0.792 AUC against the 0.624 from the kNN, this suggests that the SVM is learning different features from the BoVW feature representation better. Since kNN is distance based, it may struggle when dealing with sparse histograms. Let us now analyse Table III with its per-class F1-scores:

Class	SIFT + SVM F1	SIFT + kNN F1
Ants	0.475	0.305
Bees	0.340	0.000
Beetles	0.095	0.111
Caterpillars	0.503	0.117
Earthworms	0.376	0.243
Earwigs	0.256	0.288
Grasshoppers	0.337	0.185
Moths	0.559	0.210
Slugs	0.267	0.143
Snails	0.569	0.169
Wasps	0.485	0.125
Weevils	0.689	0.556

TABLE III: Per-class F1-scores for the SIFT+BoVW models on the test set.

We can see that the SIFT + SVM configuration yields the best scores for Snails, Weevils, Caterpillars and Moths. This is because they are usually larger in size and have better textures that create more stable SIFT descriptors and better BoVW histograms. The rest of the classes are still too difficult to classify due to having less reliable keypoints which in turn produce flawed histograms. We can see that kNN suffered from extremely low f1 score with it being 0.0 in some cases. This suggests that it maybe sensitive to overlap and imbalances in the BoVW histogram. As a result, this makes SIFT + SVM the most robust choice to classify insect pests.

C. ORB with BovW and SVM

The ORB–BoVW–SVM pipeline performed poorly as an object detector. At an IoU threshold of 0.5, detection mAP was effectively zero, indicating that the model rarely produced correctly localised boxes. Relaxing the IoU criterion to 0.2–0.4 yielded only minor gains, suggesting that the dominant failure mode was not just slightly misaligned boxes but a high volume of false positives on background regions. To curb this, we added an explicit background class via randomly sampled negative crops; however, this did not improve detection and instead reduced overall accuracy while causing the detector to over-predict a single pest class (worms). This collapse likely reflects limited separability of ORB–BoVW features between insect bodies and foliage textures, together with a distribution mismatch between the random background negatives seen during training and the structured leaf-edge proposals generated by superpixel segmentation at test time.

In contrast to localisation, the ORB–BoVW classifier showed substantially stronger performance when evaluated purely on class prediction. We first assesed the effect of using image level data for the following results:

Metric	Image-level	Crop-level
Train Time (min)	21.5	19.2
Evaluation Time (s)	130	120
mAP@0.5	0.041	0.010
Accuracy	0.282	0.309
Macro Precision	0.262	0.270
Macro Recall	0.272	0.285
Macro F1 Score	0.264	0.266
Average AUC	0.774	0.791

TABLE IV: ORB–BoVW SVM performance comparison between image-level and crop-level training.

This comparison highlights the extent to which background clutter impacts ORB feature encoding: crop-level inputs, which minimise foliage and soil textures, yield higher accuracy and AUC than image-level inputs. As such, the primary limitation of the ORB–BoVW SVM appears to lie in localisation/background rejection rather than species recognition once a clean insect region is provided. This is further supported by the crop-level confusion matrix, which shows that only a subset of classes are consistently well recognised:

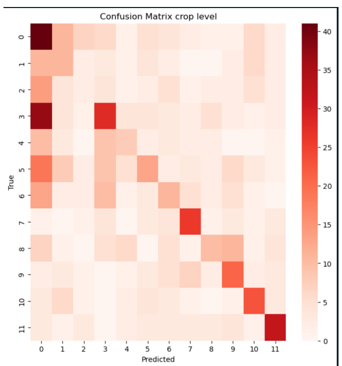

Fig. 3: Confusion matrix for the crop-level ORB–BoVW classifier.

We then assessed how pre-processing influenced the ORB–BoVW representation and downstream classification. Specifically, we compared performance on raw crop-based images against images enhanced with CLAHE and mild unsharp masking for ORB extraction, together with colour normalisation and denoising prior to SLIC proposals:

Metric	Value
Train Time (min)	22.7
Evaluation Time (s)	111
mAP@0.5	0.010
Accuracy	0.328
Macro Precision	0.324
Macro Recall	0.321
Macro F1 Score	0.316
Average AUC	0.791

TABLE V: ORB–BoVW SVM (crop-level features with pre-processing).

Pre-processing produced modest improvements in classification performance, while detection remained effectively unchanged, with mAP@0.5 staying near zero (Table V). Given these limited gains, we next evaluated whether explicitly modelling background with an added negative class could better suppress false positives. The resulting performance is:

Metric	Value
Train Time (min)	35.1
Evaluation Time (s)	320
mAP@0.5	0.009
Accuracy	0.293
Macro Precision	0.313
Macro Recall	0.330
Macro F1 Score	0.295
Average AUC	0.780

TABLE VI: ORB–BoVW SVM with explicit background modelling.

Background modelling made the pipeline substantially slower because it increased the number of regions to encode and the size of the SVM training set. However, this extra computation did not improve detection and slightly weakened classification, meaning the added runtime mainly reflects overhead from negative samples rather than a performance gain.

Finally, we analysed how the superpixel-based detector segmented images under the baseline and background-aware configurations:

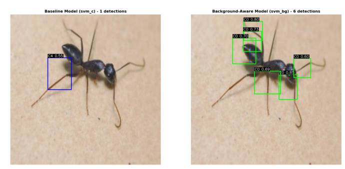

Fig. 4: Detection boxes for the baseline (left) and background-aware (right) ORB–BoVW models.

The background-aware model produces many more detections than the conservative baseline, but most are overlapping or partial boxes around the insect (and occasional background regions), so the extra predictions do not improve localisation quality and mainly add false positives, lowering accuracy. This illustrates that background modelling increased detector activity without meaningfully boosting mAP or overall classification performance. A final comparison of class based detection was done to see where each of the models struggled most:

Class	ORB Proc. Crop F1	ORB Base Img F1	ORB +BG Crop F1
Ants	0.331	0.311	0.354
Bees	0.224	0.302	0.232
Beetles	0.061	0.069	0.062
Caterpillars	0.327	0.333	0.261
Earthworms	0.222	0.179	0.176
Earwigs	0.218	0.180	0.203
Grasshoppers	0.210	0.289	0.192
Moths	0.520	0.460	0.515
Slugs	0.247	0.098	0.253
Snails	0.389	0.200	0.389
Wasps	0.474	0.396	0.474
Weevils	0.566	0.627	0.566

TABLE VII: Per-class F1 comparison for ORB variants.

D. ***Faster R-CNN with ResNet-50 backbone***

We first began by understanding the results from the data exploration stage and applying the necessary preprocessing and data augumentation to combat any limitations. Some of these included the images not being too bright and the sizes of some bugs needed to be larger for proper detection. As such we implemented the necessary augmentation strategies such as ColorJitter, random horizontal flips, random rotations, blur and noise to make our model more robust to address these variabilities in the images. We then used Optuna to programatically find the best possible hyperparameters by defining reasonable ranges for parameters such as learning rate, weight decay, batch size, etc, then running 100 trials where we aimed to capture the set of parameters that minimise the validation loss. After running this for an hour we got the following results:

```
lr: 0.0095, momentum: 0.8215,
weight_decay: 4.6772e-05, step_size: 4,
gamma: 0.2163, batch_size: 2}
```

We then separated evaluation based on two stages, object detection and object classification. Since the model returns a range of predictions based on a set of scores, we aimed to only detect and classify the predictions that it was confident about to prevent artificially skewing our results. As a result, the higher the confidence threshold, the more false positives there would be and the lower the threshold the more false negatives [11]. As a result we aimed to choose a balanced threshold of 0.5 but in our use case false negatives are more damaging than some false positives. As such, in the object detection stage, we aimed to use the mAP with an iou threshold of 0.5 and then compute the mAP@[0.5:0.95] which calculates it with an iou threshold from 0.5 to 0.95. For image classification we calculated the overall accuracy, precision, recall and F1 score. We then also provided per class metrics on those fields and calculated the area under the curve (AUC) with the one versus rest approach. Moreover, we then choose to display some images with a ground truth versus prediction bounding box and label diagram to get a better visual understanding of the performance. Finally, we decided to display attention maps from the final convolution layer of the backbone in order to better understand exactly which parts of the image was our model focusing on to guide our refinement decisions.

Now we will explore the results of the refinements we addressed in the methods section:

1) Optimiser improvement and Data Augmentation:

Metric	Value
Train Time (min)	230.15
Evaluation Time (s)	25.14
mAP@0.5	0.2082
mAP@[0.5:0.95]	0.1023
Accuracy	0.2141
Precision	0.5754
Recall	0.2543
F1 Score	0.3527
Average AUC	0.6146

TABLE VIII: ADAM optimiser w/o Data Augmentation

Metric	Value
Train Time (min)	290.08
Evaluation Time (s)	24.88
mAP@0.5	0.7251
mAP@[0.5:0.95]	0.4110
Accuracy	0.56998
Precision	0.68428
Recall	0.77335
F1 Score	0.72610
Average AUC	0.8653

TABLE IX: SGD optimiser w/ Data Augmentation

Through the tables above we can see that upgrading to the SGD optimiser with data augmentation took an hour longer to train due to the additional augmented images and since ADAM usually has a much faster convergence speed than vanilla SGD [12]. However we can see a huge improvement in the detection and classification performance where all metrics have increased considerably as a result of reducing overfitting. This is primarily due to the augmented images exposing the model to different scenarios to train upon and SGD improving the generalisation. Evaluation time as well remained the same so this was a favorable decision.

2) Anchor box modification:

Metric	Value
Train Time (min)	520.28
Evaluation Time (s)	25.16
mAP@0.5	0.6446
mAP@[0.5:0.95]	0.3169
Accuracy	0.48626
Precision	0.60447
Recall	0.71318
F1 Score	0.65434
Average AUC	0.8398

TABLE X: Updated model with modified anchors

We can see that the addition of custom anchor boxes significantly increased our training times since we got rid of the pretrained weights and had to compute more precise predictions based on the custom anchors we specified. Moreover, all the other metrics slightly decreased. A possibility of why it didn't perform as well is because the model might have needed more epochs to adjust to the new anchors or the removal of pretrained weights from the COCO dataset reduced its effectiveness. Either way, time was a huge bottleneck in continuing to further train this.

3) Imbalanced Dataset and Distortions:

Metric	Value
Train Time (min)	290.08
Evaluation Time (s)	52.92
mAP@0.5	0.5948
mAP@[0.5:0.95]	0.3384
Accuracy	0.26945
Precision	0.49806
Recall	0.36988
F1 Score	0.42451
Average AUC	0.8103

TABLE XI: Best model against an imbalanced dataset

We can see that after running the model from Table IX on an imbalanced dataset, the evaluation time significantly increased and all performance metrics also seem to drastically decrease.

This is expected since when we did the data exploration, we found the AgroPest-12 was balanced across all classes which gave it an equal opportunity to learn the features of each insect class. By introducing this imbalance, we see that the model favoured the majority classes (Bees, Moths, Slugs and Snails) with stronger metrics, however minority classes struggled.

Metric	Value
Train Time (min)	290.08
Evaluation Time (s)	28.41
mAP@0.5	0.6312
mAP@[0.5:0.95]	0.3436
Accuracy	0.50958
Precision	0.62778
Recall	0.73021
F1 Score	0.67513
Average AUC	0.8297

TABLE XII: Best model against distorted images

The distorted dataset performs significantly better than the imbalanced dataset against our best model. This is because we had already introduced a lot of data augmentation strategies to prepare our model to generalise on the distorted dataset.

4) Explainable AI with Attention Maps: We decided to capture the attention maps from the last layer of the ResNet backbone. This is because they contain the highest level features which would give us a better idea of the structure of the things its looking for. As a result we can see that in Figure 5, the model incorrectly classified the Weevil as a bee. The model focuses on the back region of the weevil where the attention contour mimics the structure of a bee. As such this may give an indication that the model wasn't picking up on the distinct features of a bee such as texture but the holistic structure. Figure 6 however was successful in identifying the Grasshopper. We can see the heatmap picked up on the legs and the elongated structure of the Grasshopper. This informs us that the model does a good job identifying the unique anatomy of the Grasshopper rather than generalising it.

Fig. 5: Attention map for Bees with score 0.61

Fig. 6: Attention map for Grasshopper with score 0.98

E. Results from YOLOv12 and Metric based Oversampling

1) Throughout the main metrics of evaluation, we can see that YOLOv12-M using metric-based targeted oversampling shows the best overall results. It achieved the best Recall (0.700), F1-score (0.761), AUC (0.875), mAP50 (0.810), and mAP50-95 (0.496). YOLOv12-N has the largest Precision (0.859) but has lower recall and localization scores, indicating a more conservative prediction tendency. Meanwhile, YOLOv12-S achieved average results without surpassing YOLOv12-M on any core metric. These achievements show that YOLOv12-N performs worse in terms of false positives, whereas YOLOv12-M benefits from increased model capacity and metric-based oversampling, leading to stronger and more stable performance across all classes.

2) In the three models considered- YOLOv12-N, YOLOv12-S, YOLOv12-M- a shared trend can be seen as to how well performing classes are always maintained throughout the three scales whereas low performing ones gradually increase in performance with the higher modeling capacity. Classes like Moths, Wasps, Weevils, Bees, and Snails are high accuracy, high recall, and excellent mAP classes in all models with only minor adjustments made by **N - S - M**, implying that these classes were already highly learned even by the smallest model.

More telling is the behavior of the other classes, in which the metamorphosis of the YOLOv12-N to the YOLOv12-S and to the YOLOv12-M is systematically strengthened. Earthworms, which started in a rather weak position and was characterized by low recall (**0.525**) and modest mAP50 (**0.575**) position in YOLOv12-N, showed even worse results with lower recall (**0.45**) in YOLOv12-S, and then a significant improvement in YOLOv12-M (**0.575** recall, **0.630** mAP50), thus indicate clear successful recovery through the impact of the larger model.

Like in the case of **Caterpillars**, the increase in size **N to S to M** demonstrates that the recall declined progressively **0.591 (N)** to **0.483 (S)** and **0.505 (M)**, meaning that although S and M models have a progressive growth in recall, they still have a downward move compared to the previous state, and the over-all progress is also a challenging issue with this category. Similar behavior occurs with Slugs, Earwigs, and Beetles, with slight differences between **N** and **S** being inflated by the higher recall, F1 and mAP50 with the **M model**, and once again the extra capacity is shown to offset the weaknesses brought about by the smaller size models.

Class / Model	Precision	Recall	F1	mAP50
Caterpillars				
YOLO-M	0.6619	**0.5054**	**0.5732**	**0.5815**
YOLO-N	**0.7143**	0.5914	**0.6471**	**0.6706**
YOLO-S	0.6818	0.4839	0.5660	0.5624
Ants				
YOLO-M	0.7683	**0.7241**	0.7456	0.7663
YOLO-N	**0.7976**	**0.7701**	**0.7836**	0.7917
YOLO-S	0.7750	0.7126	0.7425	**0.8135**
Grasshoppers				
YOLO-M	0.6531	**0.5818**	**0.6154**	**0.6470**
YOLO-N	0.6818	0.5455	0.6061	0.6388
YOLO-S	**0.7250**	0.5273	0.6105	0.6137
Earthworms				
YOLO-M	0.4694	**0.5750**	**0.5169**	**0.6297**
YOLO-N	**0.5833**	0.5250	**0.5526**	0.5751
YOLO-S	0.3830	0.4500	0.4138	0.5017

TABLE XIII: Comparison of Precision, Recall, F1, and mAP50 across YOLO-M, YOLO-N, and YOLO-S for selected classes.

On the whole, the comparison of per-class indicates that **YOLOv12-N** is surprisingly strong for its size, **YOLOv12-S** brings instability to a range of the hard to detect classes, and **YOLOv12-M** resolves those shortcomings—the highest consistency and best detection rates on the classes on which the smaller models performed poorly.

3) **Overall Assessment.** The **YOLOv12-M model**, trained with balanced oversampling and informed augmentation adjustments, emerged as the most reliable and consistent model.

TABLE XIV: Global Metrics Comparison on test dataset for YOLOv12 Models

Model	Precision	Recall	F1	Accuracy	AUC	mAP50
YOLOv12-N	**0.859**	0.694	0.760	**0.742**	0.871	0.795
YOLOv12-S	0.856	0.682	0.750	0.723	0.864	0.795
YOLOv12-M	0.854	**0.700**	**0.761**	0.737	**0.875**	**0.810**

The overall pipeline—establishing a baseline with YOLOv12-N, diagnosing weaknesses with YOLOv12-S, and applying metric-driven oversampling to train YOLOv12-M—formed a systematic and iterative improvement process. This final model demonstrated high and stable performance across both global and per-class evaluations.

V. Discussion

From the classical methods, the SIFT+BoVW+SVM pipeline emerged as the strongest appearance-based classifier, but its performance still lagged behind the deep detectors. Even with a large visual vocabulary, the SIFT features struggled on smooth-bodied or small pests such as Beetles, Slugs and Bees, reflected in consistently low per-class F1-scores. In contrast, the deep models (YOLOv12 variants and Faster R-CNN) were able to exploit discriminative global structures and fine-grained textures to achieve much higher accuracy, recall and mAP. Although some methods like Faster R-CNN took significant time to train, evaluation times were faster than their ML counterparts. The failure of kNN on several categories further highlights that simple distance-based decision rules are inadequate in the high-dimensional, imbalanced BoVW feature space. In the deep learning side, we saw that the Faster R-CNN model produced slightly worse object detection and classification metrics even though theory dictates it to be a more intensive yet accurate model. This was primarily because Faster R-CNN requires more training data before it can reach its true potential. This, coupled with the fact that YOLO has consistent architectural updates whereas Faster R-CNN is older, explains this disparity. Taken together, these results reinforce the trend identified in the literature: hand-crafted descriptors and shallow models can provide interpretable baselines, but modern deep architectures are far better suited to the fine-grained, cluttered nature of real-world pest imagery.

In terms of per-class performance, all classifiers struggled to identify the earthworm and caterpillar classes relative to others. This can be attributed to their high pose variance and elongated body shapes, which produce weak and fragmented visual cues. Deep-learning frameworks were better able to cope with these challenges and still produced serviceable results, whereas the hand-crafted methods degraded substantially when assessing these classes.

VI. Conclusion

Ultimately, we found that hand-crafted methods were limited even for classification of clean inputs and performed very poorly at object detection due to their inability to separate pests from foliage backgrounds. Classical detectors produced near-zero mAP and many false positives, indicating that local keypoint descriptors were not sufficiently discriminative for localisation in cluttered imagery. Among the classical baselines, SIFT+SVM emerged as the most reliable appearance-based classifier. However, its performance remained limited on small or low-texture pests, reinforcing the sensitivity of handcrafted pipelines to keypoint scarcity and background clutter.

Deep-learning frameworks, in contrast, were able to detect and classify pests at substantially higher rates. In particular, YOLO variants consistently outperformed all other models, showing robust localisation and improved recognition of classes that were challenging for shallower models. Faster R-CNN also delivered strong results after optimisation and augmentation, reflecting the greater data and compute demands of two-stage detectors.

These findings confirm that modern deep detectors are currently the most robust and practical solution for fine-grained pest monitoring in agricultural contexts. Future improvements should focus on expanding diverse training data, strengthening small-object detection, and addressing real-world class imbalance to ensure reliable deployment in operational agricultural environments.

References

[1] V. Malathi and M. P. Gopinath, "Classification of pest detection in paddy crop based on transfer learning approach [Retracted]," Acta Agriculturae Scandinavica, Section B–Soil & Plant Science, vol. 71, no. 7, pp. 552–559, Feb. 2021, doi:10.1080/09064710.2021.1874045.

[2] X. Wu, C. Zhan, Y.-K. Lai, M.-M. Cheng, and J. Yang, "IP102: A large-scale benchmark dataset for insect pest recognition," in Proc. IEEE/CVF Conf. Computer Vision and Pattern Recognition (CVPR), 2019, pp. 8787–8796, doi:10.1109/CVPR.2019.00899.

[3] H. Bay, A. Ess, T. Tuytelaars, and L. Van Gool, "Speeded-up robust features (SURF)," Comput. Vis. Image Underst., vol. 110, no. 3, pp. 346–359, 2008, doi:10.1016/j.cviu.2007.09.014.

[4] L. Deng, Y. Wang, Z. Han, and R. Yu, "Research on insect pest image detection and recognition based on bio-inspired methods," Biosyst. Eng., vol. 169, pp. 139–148, 2018, doi:10.1016/j.biosystemseng.2018.02.008.

[5] C. Bao, L. He, and Y. Wang, "Linear spatial pyramid matching using non-convex and non-negative sparse coding for image classification," 2015, arXiv:1504.06897. [Online]. Available: https://arxiv.org/abs/1504.06897

[6] K. Wang, W. Li, X. Wu, J. Xu, Z.-W. Wang, and S. Yang, "AP162: A large-scale dataset for agricultural pest recognition," Comput. Electron. Agric., vol. 237, Art. no. 110520, Oct. 2025, doi:10.1016/j.compag.2025.110520.

[7] R. Wang, L. Liu, C. Xie, P. Yang, R. Li, and M. Zhou, "AgriPest: A large-scale domain-specific benchmark dataset for practical agricultural pest detection in the wild," Sensors, vol. 21, no. 5, Art. no. 1601, Feb. 2021, doi:10.3390/s21051601.

[8] GeeksforGeeks, "Fast RCNN — ML," GeeksforGeeks, Feb. 27, 2020. https://www.geeksforgeeks.org/machine-learning/fast-r-cnn-ml/

[9] S. Ren, K. He, R. Girshick, and J. Sun, "Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks," arXiv.org, Jun. 04, 2015. https://arxiv.org/abs/1506.01497

[10] "Thresholds and the confusion matrix," Google for Developers, 2024. https://developers.google.com/machine-learning/crash-course/classification/thresholding

[11] P. Zhou, J. Feng, C. Ma, C. Xiong, S. Hoi, and S. Research, "Towards Theoretically Understanding Why SGD Generalizes Better Than ADAM in Deep Learning." Available: https://proceedings.neurips.cc/paper/2020/file/f3f27a324736617f20abbf2ffd806f6d-Paper.pdf