BugBot: Project Methodology

Shirley Fong, Keegan Veazey, Le Ju

March 10, 2025

Abstract

Insects are a common nuisance, with many species appearing similar, making identification challenging without expert assistance. Many homeowners in America rely on professional pest control services, which can cost thousands of dollars. Rising costs have led homeowners to attempt pest identification themselves. To address this issue, this research focuses on developing an image classification model to identify common New England household pests using transfer learning with convolutional neural networks (CNNs). Data was collected through programmatic web scraping and supplemental sources such as Google, Reddit, and iNaturalist. The final dataset contains 11 pest classes with approximately 160 photos per class, and was preprocessed using manual filtering, image hashing, resizing, recoloring, and padding. The dataset was split into training, validation, and test sets, of which data augmentation was applied on the training set. For classification, a CNN was trained on 11 pest classes. We utilize three pre-trained models for transfer learning: DenseNet201, MobileNetV2, and Xception. We also employ batch normalization, dropout layers, L2 regularization, and Bayesian/random search for hyperparameter optimization. Performance was assessed using accuracy, precision, recall, F1 score, confusion matrices, loss and accuracy curves, ROC curves, and one-versus-rest scores. The workflow is illustrated in Figure 1 and details the interactions between the various components of this research.

1 Purpose of the Methodology

A common challenge people face daily is dealing with insects. Insects can infest homes, causing structural damage and posing public health risks, as some species are poisonous or toxic to humans. To address the challenge of pest detection in New England, we have proposed an effective transfer learning based pest detection model to detect common New England pests. Transfer learning enables our model to generalize well even with a relatively small dataset, as these pre-trained networks have already learned feature representations from large datasets like ImageNet. By using a pre-trained model, we were able to develop a classifier without the need to train one entirely from scratch, making the process more efficient. In our proposed system we have utilized three popular pre-trained CNN models: DenseNet201, MobileNetV2, and Xception. Our goal is to develop a model with a high performance, requiring evaluation of multiple models to assess trade-offs in diverse metrics including accuracy, One versus Rest score, and other performance metrics. By comparing these models, we gain insight into which model achieves best performance for pest classification, accounting for model speed, comparative model results, and effects of more complex architectures.

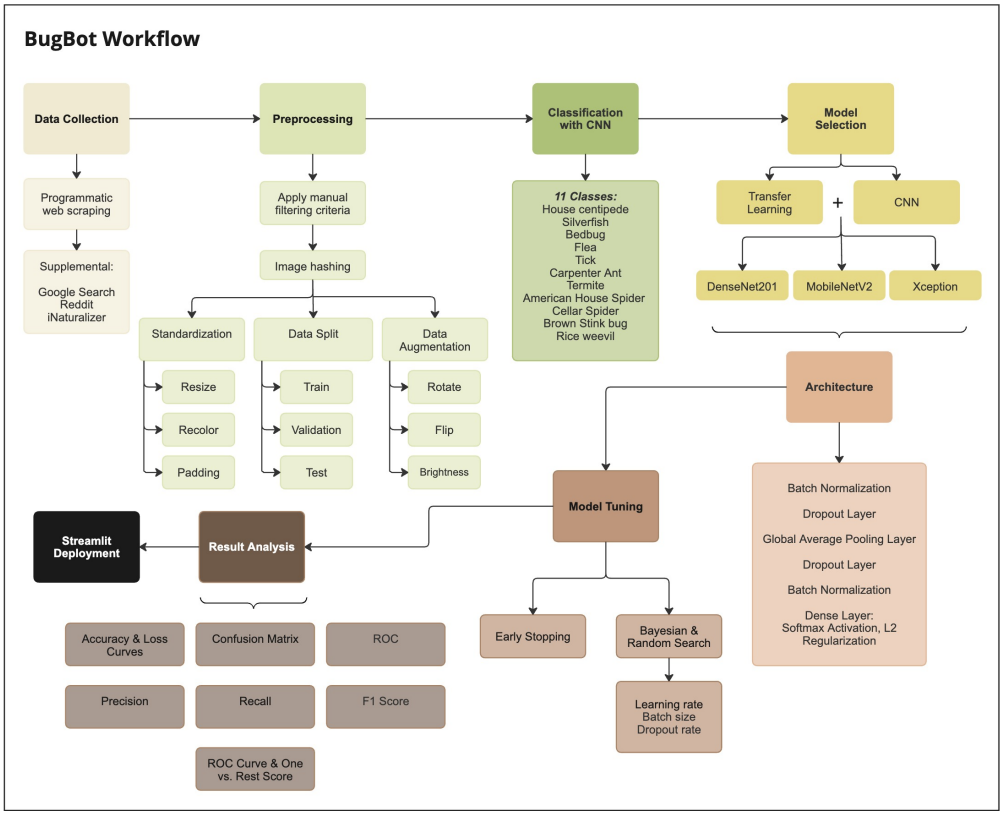


Figure 1: BugBot Workflow

2 Problem Statement

Our project deals with a machine learning multi-classification problem involving 11 insects. Household pest identification remains a significant challenge for New England residents, impacting individuals and communities such as property owners, renters, and building managers. Currently, there is a strong agricultural pest focus in existing research, and which are reliant on controlled laboratory images, ineffective for everyday use. Our contribution is a unique dataset applicable to everyday users that has real-world images and a user-friendly tool that provides users with accurate classification of household pests.

2.1 Related Work

Recent studies have demonstrated the effectiveness of CNNs in pest classification across various contexts. A relevant study is the research paper "An Efficient Deep Learning Approach for Jute Pest Classification Using Transfer Learning" by Muhammad Tanvirul Islam and Md. Sadekur Rahman. The primary similarity between their work and our proposed approach lies in the pest-classification specific image topic as well as the use of a CNN model use. However, their research specifically targets Jute pests, which pose significant challenges to Jute crop production in Bangladesh (Islam & Rahman, 2024). In contrast, our study will use a completely different dataset and will be used to classify common household pests found in New England. Another key difference between their work and ours is that Islam and Rahman investigate a variety of pre-trained models, while we plan to build the network from scratch while also exploring the potential of

using ensemble methods.

In our review, we found the lack of diversity in training data to be notable factors across the literature. For example, Turkoglu et al.'s PlantDiseaseNet study in the paper "PlantDiseaseNet: Convolutional Neural Network Ensemble for Plant Disease and Pest Detection" utilized a highly controlled dataset, comprised of RGB images captured with standardized equipment and consistent resolution of 4000 x 6000 pixels. This contrasts significantly with our proposed methodology, which will incorporate web-scraped images with varying resolutions and background contexts. Furthermore, PlantDiseaseNet uses an ensemble of Support Vector Machines (SVMs) as the primary classifying component, only using a CNN as an averaging model for the final classification across multiple SVM outputs (Turkoglu et al., 2021). Meanwhile, our approach uses the CNN directly by feeding in the image data as the first layer.

The literature also reveals a consistent emphasis on data processing to improve model performance. For instance, Rehman et al.'s work on brain tumor classification in the paper "A Deep Learning-Based Framework for Automatic Brain Tumors Classification Using Transfer Learning", while in a different domain, provides valuable insight into image pre-processing. Their application of image flipping and rotation techniques Rehman et al. (2019) aligns with our planned approach, though our implementation must account for the greater variability in pest imagery compared to standardized medical imaging.

The research paper "Crop pest classification based on deep convolutional neural network and transfer learning" by the authors K. Thenmozhi and U. Srinivasulu Reddy also implements image processing for improved model performance, applying reflection, scaling, and rotation techniques. Thenmozhi and Reddy's research also represents one of the more comprehensive studies in the field, achieving over 95% classification accuracy across three different datasets of crop pests (Thenmozhi & Srinivasulu Reddy, 2019). They further compare custom CNN models against pre-trained architectures including AlexNet and ResNet. We intend to adapt their comparison approach and to use similar augmentation techniques while acknowledging the different challenges presented by our more diverse dataset.

For contrast, Rajan et al.'s work in the research paper "Detection And Classification Of Pests From Crop Images Using Support Vector Machine" demonstrates the limitations of simpler classification approaches. Their SVM-based system, while effective for their use case of binary classification between whiteflies and aphids, highlights the need for more complex learning approaches when handling multiple pest classes. Rajan et al.'s approach relies on manually designed feature extraction to detect and recognize pests. Manual feature extraction limits the accuracy and adaptability of pest detection by failing to capture varied image characteristics (Guo et al., 2024). Furthermore, their study's reliance on controlled greenhouse imagery further underscores the importance of our more comprehensive approach using diverse image sources.

Our research aims to bridge the gap between these agricultural-focused studies and household pest identification needs by incorporating the successful methodological elements identified in the literature such as image rotation and mirroring, and by comparing our CNN model to similar projects such as Islam and Rahman (2024) jute pest study. Our planned use of web-scraped images and potential ensemble method investigation rep-

resents a novel approach that addresses everyday people's needs while building upon established technical foundations.

3 Data Collection and Preparation

The majority of the BugBot dataset was scraped using a publicly available API for scraping images from Bing search queries. A manual review and filtering process was then applied to the Bing-based dataset and was subject to the following standards:

1. The insect must be present in the image
2. The image must show at least 75% of the insect's body
3. The photo is not a drawing, cartoon, or an AI generated image
4. The image depicts the adult/matured insect

For all collected images, it is also essential to eliminate duplicates or near-duplicates to prevent data leakage. Visual inspection in addition to image hashing are implemented. Since no temporal or spatial dependencies exist in the dataset, random splitting for training, validation, and test subsets is appropriate. The validation set will guide hyper-parameter tuning, while the test set will provide an unbiased assessment of the model's performance.

Furthermore, it is important to note the unique nature of many pests that usually infest home environments in groups or colonies, including ants, termites, and bed bugs. Therefore, a direct effort was made to collect additional group images of insect infestations to be included in the dataset. After filtering, a range of approximately 16%-76%, depending on the insect class, was kept from the original scraped data based on the above criteria.

After web scraping, additional data was collected by performing manual Google searches for image results using insect keywords, and by taking advantage of community postings on websites such as Reddit to collect home-environment-specific images. After combining the scraped images and manual supplement, the resulting dataset includes 160 raw images across each of the 11 insect classes.

Since our dataset consists of RGB images with x and y coordinates, the only features are the image pixels themselves. However, additional hidden features will be extracted through preprocessing and modeling the data and includes visual characteristics, such as texture, RBG value, and pixel sequence. Furthermore, when the complete image matrix is flattened into a 1-dimensional feature vector, each pixel element becomes a feature of the overall image.

The target variable in this supervised learning project is the insect class, representing one of the 11 common household pests. It is a well-defined categorical variable with clear labels derived from the dataset. Since the dataset is designed with a balanced distribution of 160 images per class (100 for training, 40 for validation, and 20 for testing), there is no inherent class imbalance, and resampling techniques are not necessary. This ensures that the model will not favor any single class during training or evaluation.

Due to the dataset design and manual filtering criteria, we have curated a dataset that adequately represents every insect class to ensure equal distribution of training, validation, and test data to reduce bias. We are also aware of potential bias in web scraping data due to the prevalence of scientific images which may not reflect the targeted end user – an everyday homeowner classifying an insect in their home. To mitigate this we have further supplemented the dataset with manually selected images with diverse backgrounds and contexts including images from Reddit and iNaturalist.

In terms of resources required, the dataset, consisting of 1,760 images, is relatively small and is manageable to process with standard resources. The dataset size even after augmentation does not exceed the limits of local storage or memory. Therefore, the dataset does not require advanced techniques such as distributed processing (e.g., using Apache Spark or Dask) since the computational and storage demands are minimal. Although no distributed processing will be required, the data will require multiple pre-processing steps which include:

1. Removing duplicates
2. Standardizing the data to a fixed image size (e.g., 224x224)
3. Normalizing the pixel values
4. Color standardization (i.e., RGBA $\rightarrow$ RGB)
5. Adding padding to the images
6. Applying data augmentation techniques including rotation, height and width shift, and brightness adjustments to expand the training dataset

It is important to acknowledge the limitations of this dataset which could include a lack of sufficient data, potentially leading to over-fitting during training and data imbalance. When this case occurs, it could skew model predictions and affect the model's accuracy. This will be addressed and mitigated through data augmentation, mentioned above, through techniques including image rotation, horizontal and vertical flips, and cropping of images. By ensuring that each insect class contains 160 unique images and by expanding the dataset to include varied versions of all images, we effectively diversify the data set to mitigate over-fitting and enhance the model performance.

4 Selection of Models

Selecting the appropriate machine learning model is crucial for optimizing classification performance, especially given the complexity of pest identification from real-world images. Our goal was to build a robust classifier that generalizes well across diverse everyday pest images.

4.1 Model Selection Criteria

We based our model selection on the following criteria:

- Pretrained model performance: Transfer learning significantly accelerates training and improves accuracy, as models trained on large datasets like ImageNet learn generalized feature representations.
- Computational efficiency: Since our dataset consists of only 1,760 images across 11 classes, we prioritized models that offer high accuracy while remaining computationally efficient.
- Overfitting mitigation: Given the relatively small dataset, we selected image specific models which use regularization techniques such as dropout, batch normalization, and L2 regularization.

4.2 Selected Models

To evaluate model effectiveness, we initially ran six different models: CNN without transfer learning, EfficientNet, ResNet50, VGG16, MobileNetV2, DenseNet201, and Xception. Based on the validation metrics, we decided to use the following three CNN architectures: DenseNet201, MobileNetV2, and Xception. DenseNet201 had the best performance in regards to evaluation metrics and, with its ability to strengthen feature propagation and reduce redundant parameters, it makes it highly effective for classification tasks with limited data. MobileNetV2 had the second-best performance and was selected for its lightweight architecture, which balances accuracy and efficiency, making it ideal for real-time applications. Lastly, Xception had the third-best performance in accuracy.

4.3 Justification for Model Comparison

Instead of relying on a single model, we systematically compared multiple architectures to identify the best-performing model in terms of accuracy, precision, recall, F1 score, and One versus Rest score. Moreover, it assesses how architectural complexity influences performance, speed, and generalization. Subsequently, we determined whether lightweight models like MobileNetV2 can achieve results comparable to deeper networks like DenseNet201.

4.4 Transfer Learning Strategy

Training a CNN from scratch would require a significantly larger dataset and resources. Therefore, transfer learning was used with pre-trained weights from ImageNet. The base layers of each model were frozen, and a custom classification head was added, which is discussed in Section 5.1: Model Architecture and Configuration.

4.5 Implementation and Evaluation

To further refine model performance, we conducted hyperparameter tuning using Bayesian optimization and random search, optimizing learning rates, batch sizes, and dropout rates. The evaluation metrics and final model selection are discussed later in the report, specifically in Section 6: Evaluation and Comparison.

5 Model Development and Training

This section discusses the architecture, training process, and hyperparameter-tuning methodology for BugBot. As discussed in the previous section (Section 4: Model Selection), the three aforementioned pre-trained models were employed to accomplish the transfer learning portion of our modeling. By using these models for transfer learning, we leverage the generalized knowledge pre-trained on ImageNet's diverse 1.4 million images for each model. This approach reduces computational resources and training time and is accomplished by freezing the base model's 20 million parameters and training our custom classification head. By employing transfer learning, we are able to focus further resources on hyperparameter optimization rather than designing complex networks from scratch.

5.1 Model Architecture and Configuration

To configure each transfer learning model with a custom classification head, the following structure was used:

For each baseline model (DenseNet201, MobileNetV2, Xception):

- Use preprocessed training data as input (see Figure 1: BugBot Workflow)
- Set baseline model as an initial base, freezing ImageNet weights
- Append customized classification head to the base model consisting of:
  - First batch normalization layer
  - First dropout layer with tunable rate for regularization
  - Global Average Pooling layer to reduce spatial dimensions while preserving channel information
  - Second dropout layer with tunable rate for regularization
  - Second batch normalization layer
  - Dense output layer with softmax activation, output neuron count matching the number of pest classes (11), and uses L2 regression as a kernel regularizer

5.2 Data Preprocessing and Data Split

Before training, the data is preprocessed using the preprocessing steps in Figure 1, and importantly includes standardization measures of resizing, scaling, recoloring, and removing duplicates via image hashing with the Undouble library (see Section 3: Data Collection and Preparation). The data is then divided into three separate directories: training, validation, and testing sets following the split of 100 training images, 40 validation images, and 20 test images for each class. The training set then undergoes data augmentation, including image mirroring, brightness adjustment, and rotation using PIL Image, ImageEnhance, and Numpy libraries. Notably, for each dataset directory, Keras ImageDataGenerator library is used to create custom data generators for efficiency throughout the model training and evaluation process:

- `TRAIN_GENERATOR`: Provides shuffled batches for model training

- `VAL_GENERATOR`: Provides validation data during training and hyperparameter tuning
- `TEST_GENERATOR`: Provides unshuffled data for final model evaluation
- `EVAL_VAL_GENERATOR`: Provides unshuffled validation data for metric calculation

5.3 Mitigating Overfitting

AS previously mentioned, when training a CNN on a small dataset like ours, the risk of overfitting can be high since the model can memorize the training data instead of generalizing it to unseen data. Therefore, mitigating overfitting and performing hyperparameter tuning for better model performance and efficiency was essential in our model experiments. To do this, the models are configured with batch normalization, dropout, L2 regularization, and early stopping. Batch normalization helps by keeping the activations of each layer stable during training. Dropout helps by randomly dropping neurons during training, preventing reliance on specific features. We employed two layers of dropout, each of which is tuned using Keras Tuner, described in section 5.4. L2 Regularization (ridge regression) helps by penalizing large weights without zeroing out features as L1 regularization does. Early stopping, unlike the other methods mentioned, is not a layer application and instead contributes to mitigating overfitting by preemptively stopping the training process when the validation loss stops improving. Besides specifying validation loss as the metric to keep an eye on, there are two parameters we also set: patience (set to 3) and min_delta (set to 0.001). Patience is the number of epochs with no improvement allowed before stopping training, and min_delta is the minimum change in the monitored metric required to consider it an improvement.

5.4 Hyperparameter Tuning and Optimization Algorithms

Two hyperparameter tuning methodologies were implemented for each model using Keras Tuner: Bayesian Optimization and RandomSearch. Notably, grid search was initially implemented for our models. However, we found the implementation to be inefficient for our limited resources. The inefficiency of grid search is due to its exhaustive approach to exploring all possible hyperparameter combinations, resulting in high computational costs. In contrast, Bayesian optimization and random search explore the hyperparameter search space more efficiently by focusing on probable combinations. Essentially, these algorithms make smarter and more efficient decisions than grid search, requiring fewer trials and less computational resources to identify optimal hyperparameters.

In BugBot, Bayesian optimization is used as the main optimization algorithm and is specifically set to optimize combinations for validation accuracy. Random search is used to help verify the robustness of hyperparameters and to provide other potential hyperparameter combinations to explore if results differ from the Bayesian optimization result. Since neither algorithm explores the entire search space, setting the max_trials parameter, which defines the maximum number of combinations the algorithm will attempt, is important. For each model, the learning rate, dropout rate, and batch size are tuned, resulting in the following hyperparameter space:

- Learning rate: [0.01, 0.001, 0.0001]

- Dropout rate: [0.2, 0.3, 0.4, 0.5]
- Batch size: [16, 32, 64]

Given three choices for learning rate, four for dropout, and three for batch size, the maximum number of trials is 36 (3 * 4 * 3). Therefore, we set max_trials to 20, representing 20/36. This means the algorithms will, at most, search about 55% of the original search space compared to grid search, which would explore all 36.

5.5 Hyperparameter Optimization Workflow

The hyperparameter tuning is implemented as the following process for each model:

For each algorithm (Bayesian and Random Search):

1. Multiple trials are conducted with different hyperparameter combinations
2. Each model is evaluated on validation data from `VAL_GENERATOR`
3. Early stopping halts training as needed with the last epoch being saved
4. The best-performing hyperparameter combination is identified and saved to a dedicated dictionary for the current model and current optimization algorithm
5. A final model is constructed with the best parameters identified in step 4
6. The model is trained for the optimal number of epochs identified in step 3
7. Evaluation metrics (described in section 6) are calculated on both validation and test sets and saved to CSV for comparison with other models

Multiple independent tuning runs were conducted to ensure the robustness of the model, particularly concerning early stopping parameters. The model pipelines were tested using the argparse library for efficient parameter modification through the command line, allowing for adjustments of patience values to [3, 5, 10] and min_delta values to [0.0001, 0.001, 0.01].

The systematic methodology we use for BugBot combines transfer learning, custom classification heads, and dual hyperparameter optimization algorithms for robust pest identification that employs different overfitting mitigation techniques.

6 Evaluation and Comparison

In this section, the results of the three different customized pre-trained models are compared and analyzed. To evaluate the performance and choose the final model, we used a comprehensive set of key metrics: accuracy, loss curve, confusion matrix, precision, recall, and F1 score. The final model decision was further supported by supplemental metrics, comprising of receiving operating curves (ROC) and one versus rest scores, which provided additional insights into model performance.

6.1 Evaluation Metrics

- **Accuracy**: A metric to correctly detect pests. It is given by:

$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN}$$

- **Precision**: A metric to determine the number of successful positive predictions the model has created. It is given by:

$$\text{Precision} = \frac{TP}{TP + FP}$$

- **Recall**: A metric that counts the number of accurate positive predictions made out of all possible positive predictions. It is given by:

$$\text{Recall} = \frac{TP}{TP + FN}$$

- **F1 Score**: A metric that is the harmonic mean of the recall and precision scores, used to compare two classifiers. A score of 1 indicates a flawless classifier. It is given by:

$$F1\ Score = 2 \times \frac{\text{precision} \times \text{recall}}{\text{precision} + \text{recall}} \times 100$$

- **ROC AUC**: A plot of the false positive rate versus the true positive rate. The area under the curve (AUC) is used to compare different models.

- **One-versus-rest score**: A classification strategy that trains one binary classifier per class. For each classifier, the target class is treated as positive, while all other classes are treated as negative. This approach is commonly used for multiclass classification problems.

- **Macro Average**: The arithmetic mean of each class related to precision, recall, and F1 score. This evaluation is used for the overall performance of multiclass classification. It is given by:

$$\text{Macro Average} = \frac{1}{N} \sum_{i=1}^{N} \text{measure in class}_i$$

The loss and accuracy curves of our three pre-trained models are depicted in Figures 2 and 3. Based on these curves, there is a gap between training and validation accuracy, which suggests overfitting. As shown in Figure 3, the models show a steady increase in training accuracy from epochs 0 to 15, reaching over 85%. However, the validation accuracy stays relatively low for MobileNetV2 and Xception, fluctuating between 65% to 70% throughout the training. The overall trend for validation accuracy for those two models is inconsistent and does not match the increase in training accuracy. In contrast, DenseNet201 shows slightly better validation accuracy performance with fewer variations. Although there is still a noticeable gap between training and validation accuracy, DenseNet201 appears to generalize better than MobileNetV2 and Xception, with fewer strong fluctuations in its validation accuracy curve. The loss curves further support overfitting. In Figure 2,

the training loss for all the models decreases steadily throughout the epochs, indicating effective learning on the training data. However, the validation loss remains higher than the training loss and shows inconsistent fluctuations across the epochs. This is evident in MobileNetV2 and Xception, where the training loss rapidly declines and stabilizes at a low value, while the validation loss remains considerably higher without a consistent downward trend. The results of these curves indicate that the models quickly learn patterns in the data , reaching saturation early in training. DenseNet201 follows similar patterns as the two other models, except it has a smaller gap between training and validation loss. This indicates that DenseNet201, despite achieving high training accuracy, is also overfitting to the training data. However, compared to MobileNetV2 and Xception, the validation loss for DenseNet201 shows slightly less fluctuation, which indicates better generalization. Based on the curves displayed in Figures 2 and 3, DenseNet201 appears to be the most effective model during training.

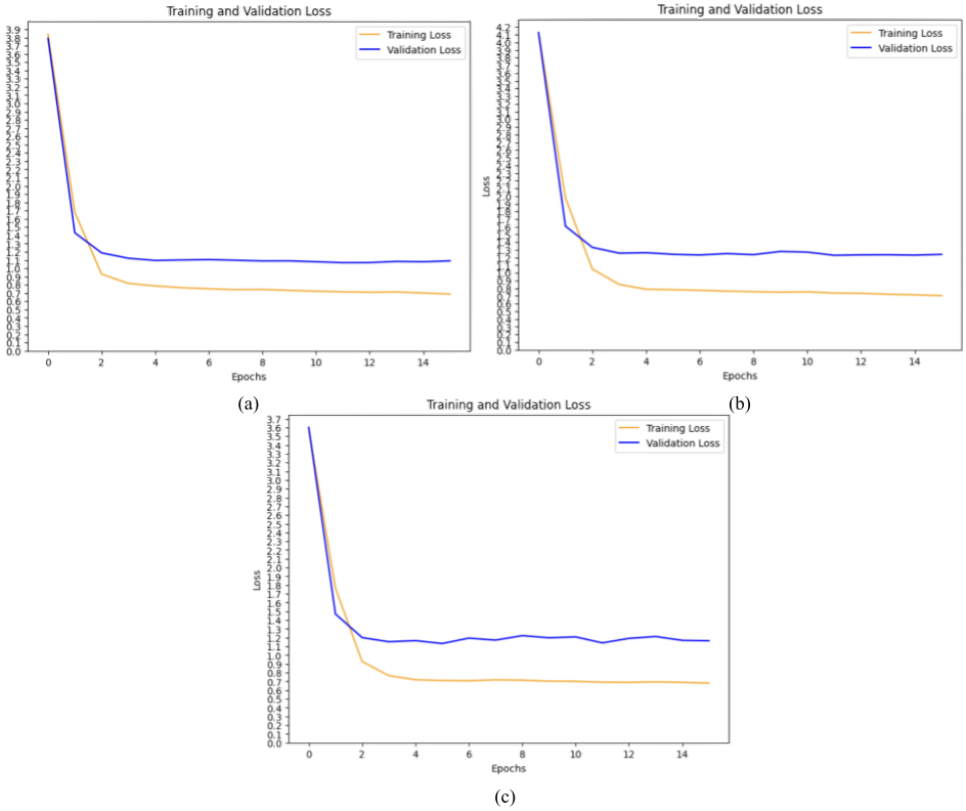


Figure 2. Loss curve (a) DenseNet201; (b) MobileNetV2; (c) Xception.

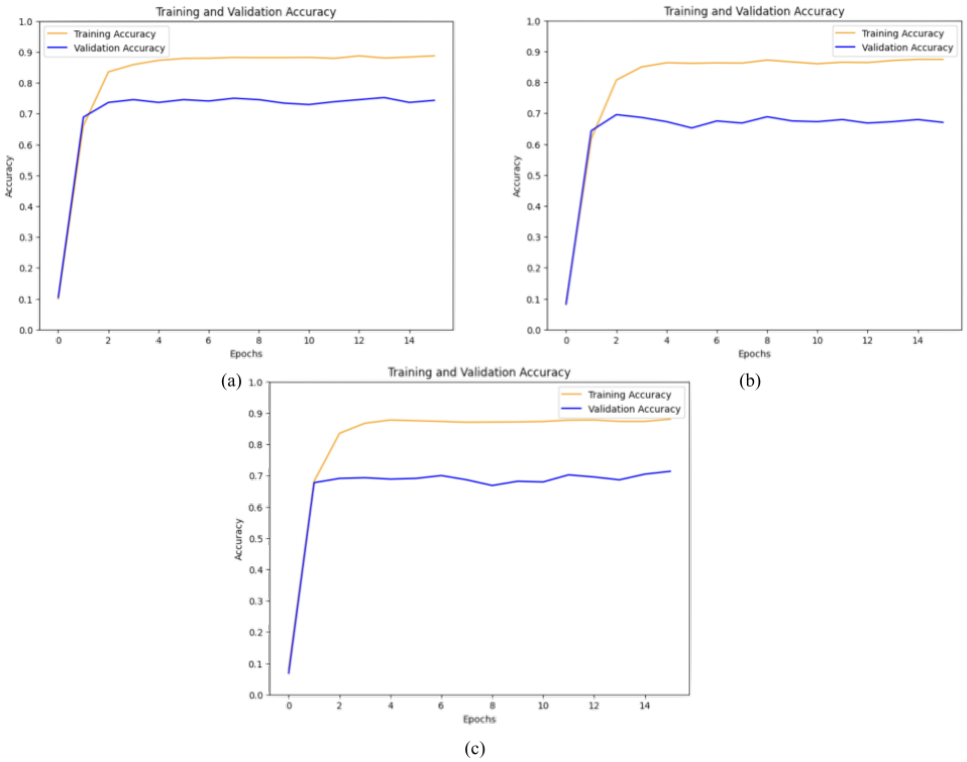


Figure 3. Accuracy curve (a) DenseNet201; (b) MobileNetV2; (c) Xception.

After training, all models were evaluated using the testing dataset. The evaluation process used multiple approaches to assess model performance. The confusion matrix provided insights into the number of correct and incorrect classifications. The confusion matrix for each model is presented in Figure 4, displaying the counts of true positives, true negatives, false positives, and false negatives, for the predicted values. After analyzing the confusion matrices, it was observed that Xception and MobileNetV2 produced similar classification results. DenseNet201 had the best performance in classifying pests, with fewer misclassifications compared to MobileNetV2 and Xception. A notable result from the confusion matrix is that bed bugs were often misclassified as fleas, highlighting that all three models had trouble distinguishing between these two classes, which is evident in Tables 1, 2, and 3. As shown in Figure 4, the models can be ranked based on their classification ability in the following order: DenseNet201, MobileNetV2, and Xception.

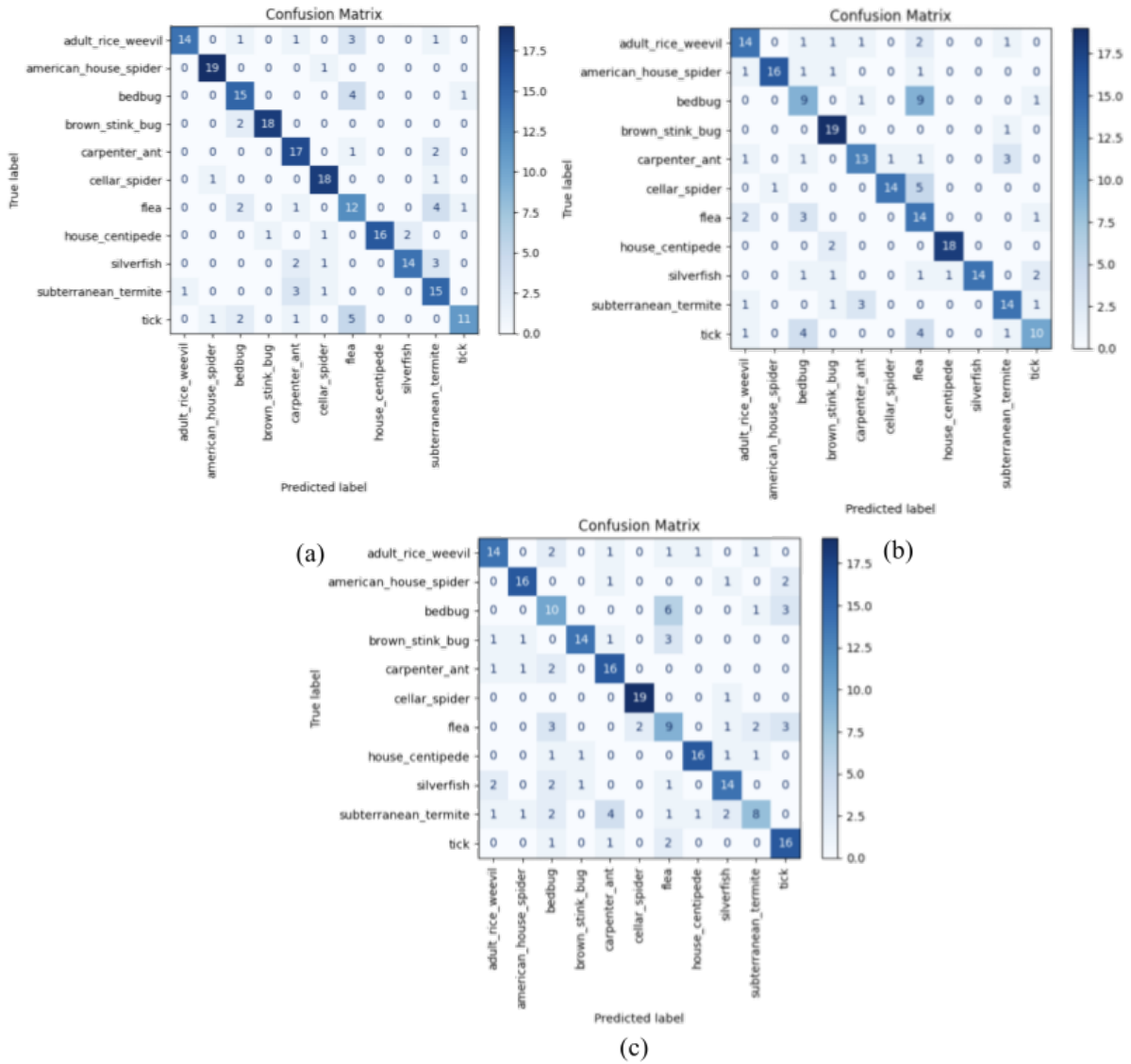


Figure 4. Confusion matrix (a) DenseNet201; (b) MobileNetV2; (c) Xception.

Following the analysis of the models using the confusion matrix, we evaluated their performance by calculating precision, recall, F1 score, and accuracy. Tables 1, 2, and 3 present these metrics for each model across different pest categories, while Table 4 summarizes the overall accuracy, including one-versus-rest score. In Tables 1-3, Xception and MobileNetV2 displayed moderate performance, with some inconsistencies across certain pest classes, such as fleas and bed bugs. DenseNet201 consistently outperformed both models, delivering higher values for all three metrics across most pest classes. Viewing the macro-average measures in Tables 4, Xception and MobileNetV2 achieved competitive scores, but DenseNet201 demonstrated the highest scores for all the metrics. The ROC curves (located in the Appendix) further reinforce these findings, as DenseNet201 achieved the highest true positive rates. Based on these results, this research concludes that DenseNet201 is the most effective and reliable model for accurately identifying common household pests.

Class Name	Precision (%)	Recall (%)	F1 Score (%)
Rice weevil	74	70	72
American house spider	84	80	82
Bed bug	43	50	46
Brown stink bug	87	70	78
Carpenter ant	67	80	72
Cellar spider	90	95	93
Flea	39	45	42
House centipede	89	80	85
Silverfish	70	70	70
Subterranean termite	61	40	49
Tick	67	80	73

Table 1. Performance of the customized Xception model on individual classes.

Class Name	Precision (%)	Recall (%)	F1 Score (%)
Rice weevil	70	70	70
American house spider	94	80	87
Bed bug	45	45	45
Brown stink bug	76	95	84
Carpenter ant	72	65	68
Cellar spider	93	70	80
Flea	38	70	49
House centipede	95	90	92
Silverfish	100	70	82
Subterranean termite	70	70	70
Tick	67	50	57

Table 2. Performance of the customized MobileNetV2 model on individual classes.

Class Name	Precision (%)	Recall (%)	F1 Score (%)
Rice weevil	93	70	80
American house spider	90	95	92
Bed bug	68	75	71
Brown stink bug	95	90	92
Carpenter ant	68	85	75
Cellar spider	81	90	85
Flea	48	60	53
House centipede	100	80	89
Silverfish	87	70	78
Subterranean termite	58	75	65
Tick	85	55	67

Table 3. Performance of the customized DenseNet201 model on individual classes.

Model Name	Precision (%)	Recall (%)	F1 Score (%)	Accuracy (%)	OVR Score (%)
Xception	70	69	69	69	94
MobileNetV2	74	70	71	70	95
DenseNet201	79	77	77	77	97

Table 4. Overall performance (Macro Avg.) of the customized pre-trained model in our dataset.

7 Conclusion and Future Work

The identification of common household pests is crucial for homeowners to maintain a pest-free environment and ensure proper measures are taken to eradicate these insects. To simplify this process, artificial intelligence, specifically image recognition techniques, has proven to be an effective tool. This research explores the use of deep convolutional neural networks and transfer learning to classify common household pests. Through experimentation and a comparative analysis of various deep learning architectures, the proposed model was able to successfully identify multiple pest classes. With a high One vs Rest score, the classifier has the ability to distinguish each class from all others with high confidence. The combination of web scraping and data augmentation allowed us to generate more training data, thus enhancing the model performance. Among the tested models, DenseNet201 demonstrated the highest accuracy over the other techniques, achieving 77% classification performance, and high 97% OVR score. The effectiveness of the approach was validated using precision, recall, F1 score, confusion matrices, loss curves, and accuracy curves, confirming the potential of deep learning for automated household pest detection.

7.1 Limitations

Given our available resources, our team focused on developing a household pest identification model that balances efficiency, accessibility, and accuracy. To ensure the models could run smoothly on local devices, we prioritized using computationally efficient architectures, making the system more practical and easier for users to replicate without requiring advanced hardware. The limitations of this study are as follows:

- The dataset size was relatively small, therefore it does not cover the full diversity of household pests, as certain species might not have sufficient images available for training, leading to an incomplete representation.
- The system might not perform equally well in all lighting conditions as pest images from smartphones can vary in terms of quality.
- Despite efforts to prevent overfitting, the model may still perform suboptimally when applied to unseen pest images and would benefit from more training images.
- While the goal is to create an accessible system, achieving high accuracy often requires computationally intensive models and, therefore, could cause computational constraints.

To address the current limitations, future work could focus on expanding the dataset by collecting more diverse and high-quality images of household pests. Implementing additional data augmentation techniques could further enhance model robustness. Additionally, exploring more advanced deep learning architectures that balance performance and computational efficiency could improve classification accuracy without sacrificing accessibility.

References

Guo, B., Wang, J., Guo, M., Chen, M., Chen, Y., & Miao, Y. (2024). Overview of pest detection and recognition algorithms. *Electronics*, *13*, 3008–3008. https://doi.org/10.3390/electronics13153008

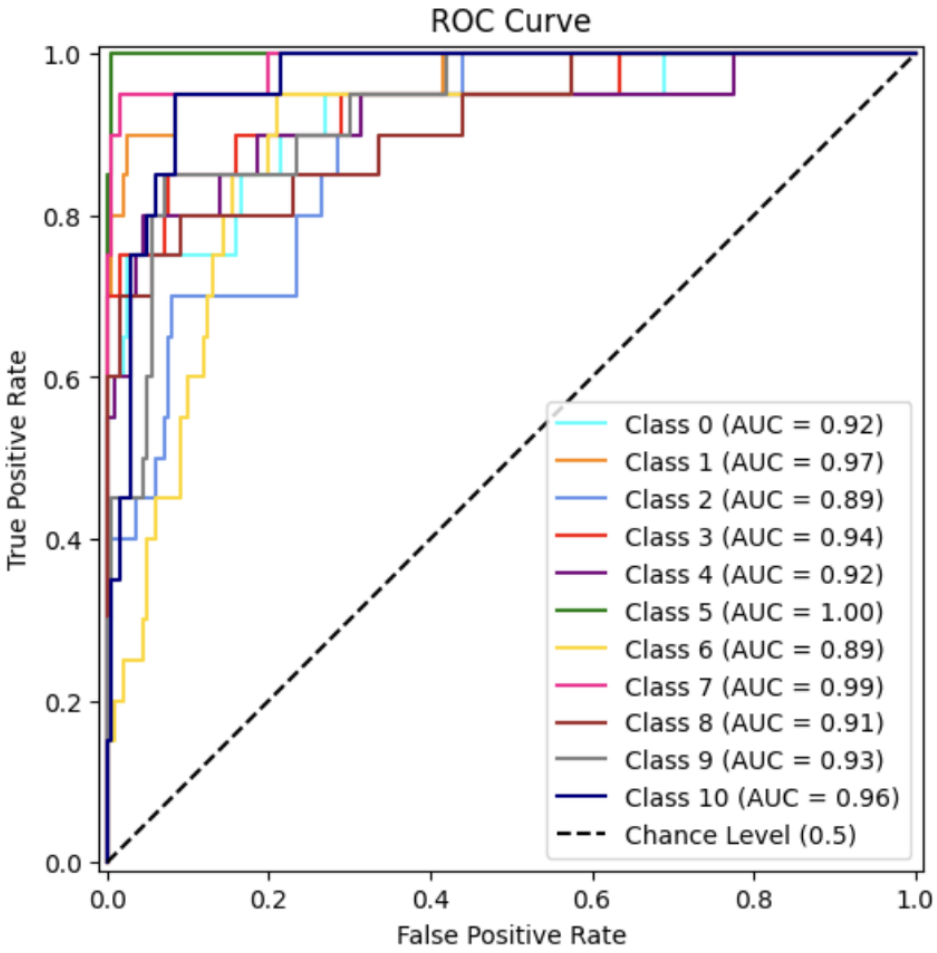
Islam, M. T., & Rahman, M. S. (2024). An efficient deep learning approach for jute pest classification using transfer learning, 1473–1478. https://doi.org/10.1109/iceeict62016.2024.10534395

Rehman, A., Naz, S., Razzak, M. I., Akram, F., & Imran, M. (2019). A deep learning-based framework for automatic brain tumors classification using transfer learning. *Circuits, Systems, and Signal Processing*, *39*, 757–775. https://doi.org/10.1007/s00034-019-01246-3

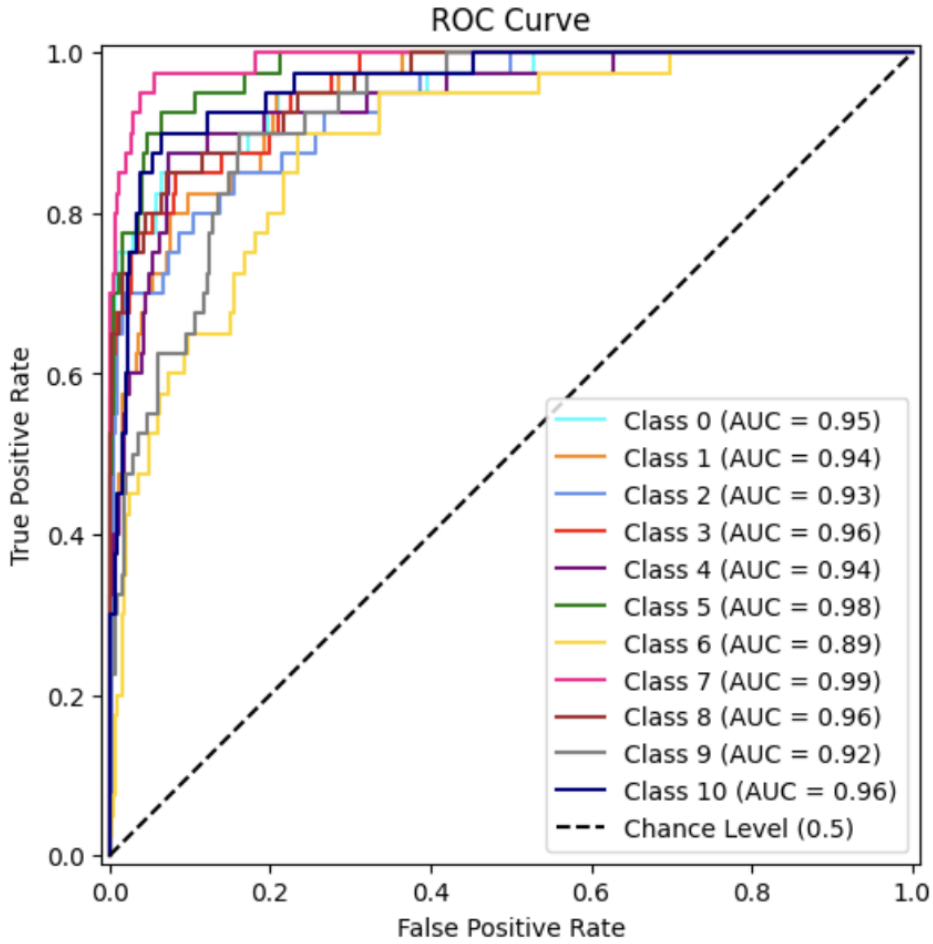
Thenmozhi, K., & Srinivasulu Reddy, U. (2019). Crop pest classification based on deep convolutional neural network and transfer learning. *Computers and Electronics in Agriculture*, *164*, 104906. https://doi.org/10.1016/j.compag.2019.104906

Turkoglu, M., Yanikoğlu, B., & Hanbay, D. (2021). Plantdiseasenet: Convolutional neural network ensemble for plant disease and pest detection. *Signal, Image and Video Processing*, *16*. https://doi.org/10.1007/s11760-021-01909-2

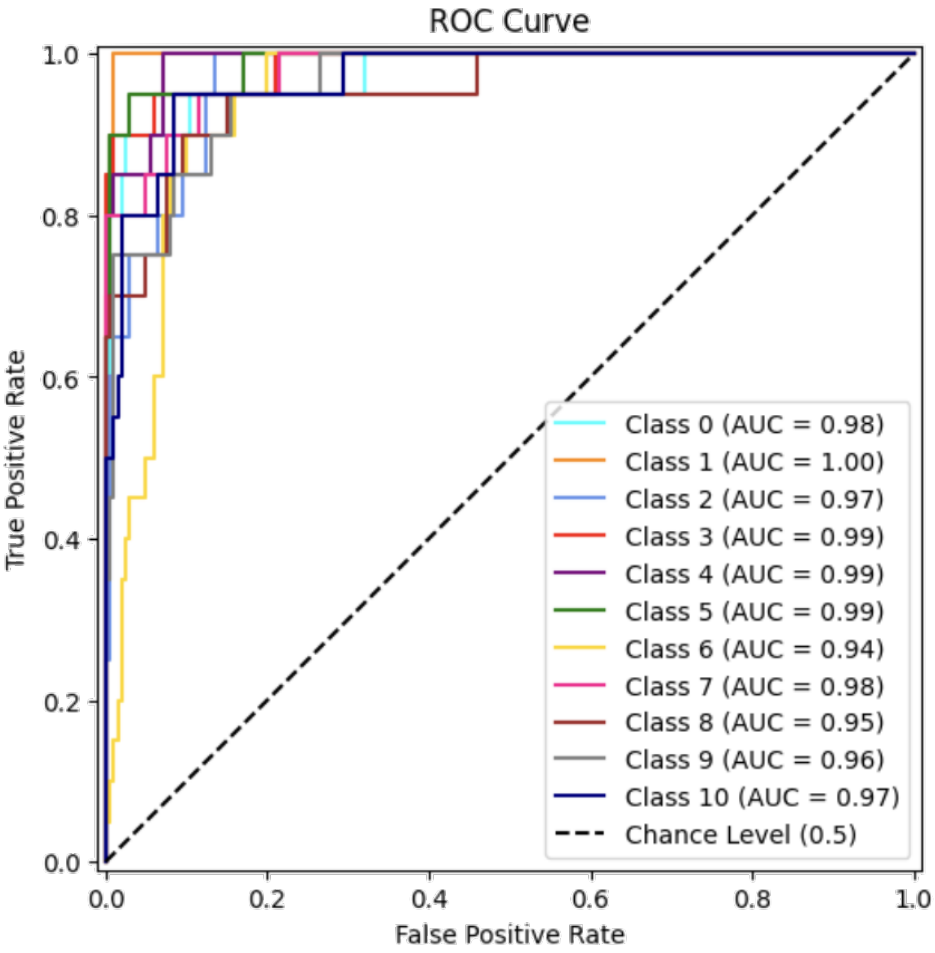
Appendix



Appendix-1: ROC curve of Xception



Appendix-2: ROC curve of MobileNetV2



Appendix-3: ROC curve of DenseNet201