Safeguarding Agriculture:

AI-Enabled Prognostication of Farm Insect Threats

PROJECT REPORT

Date of Submission: 09.11.2023

Team ID: Team-592713

Team Members:

Sushmetha S R - 21BAI1162

Shuvam J - 21BAI1131

Gaddam Krithisha - 21BCE1491

Raghav S - 21BPS1181

CONTENTS

1. Introduction

1.1 Project Overview

In recent years, the agriculture industry has witnessed a growing need for innovative solutions to address the challenges associated with pest and insect control on farms. The presence of harmful insects can lead to substantial crop losses, making it essential to accurately identify and manage these pests. This project endeavours to provide a novel and efficient solution by harnessing the power of deep learning and computer vision techniques to classify insects found on farms.

Our approach involves building a Convolutional Neural Network (CNN) model trained on a comprehensive dataset comprising images of 15 different types of insects commonly found in agricultural settings. The model will enable farmers and agricultural professionals to quickly and accurately identify insect species present on their farms, paving the way for precise pest management strategies.

1.2 Purpose

The primary purpose of this project is to develop a practical tool that aids farmers in insect classification and provides targeted recommendations for insecticide usage. This tool not only simplifies the identification process but also enhances the overall efficiency of pest management on farms. The key objectives of our project are as follows:

- To create a deep learning model using Convolutional Neural Networks that can accurately classify insects from images.
- To integrate this model into a user-friendly web application accessible to farmers.
- To empower farmers with the ability to upload insect images and receive instant insect species identification.
- To recommend suitable insecticides and pest control measures based on the identified insect species, thereby aiding in informed decision-making for pest management.

Through this project, we aim to contribute to the sustainable and efficient management of pests on farms, reduce crop losses, and ultimately enhance agricultural productivity. The integration of technology into agriculture is expected to bring substantial benefits, not only in terms of time and cost savings but also in reducing the environmental impact of indiscriminate pesticide use.

2. Literature Survey

2.1 Existing Problem

The agriculture industry faces a significant challenge in dealing with pest and insect infestations that can cause extensive crop damage and result in substantial economic losses. The lack of timely and accurate insect identification systems often hinders effective pest management. Traditional methods of insect identification are labor-intensive, time-consuming, and often error-prone. Additionally, misidentifying insect species can lead to the improper use of pesticides, which has environmental implications and may exacerbate the issue of pesticide resistance among insect populations.

2.2 References

Dangerous Farm Insects Dataset: https://www.kaggle.com/datasets/tarundalal/dangerous-insects-dataset **2.3**

Problem Statement Definition

The central problem addressed by this project is the need for an efficient, accurate, and accessible insect classification and pest management system in agriculture. How might we utilize Convolutional Neural Networks (CNN) and existing insect image datasets to create an AI-driven early detection and prevention system for agricultural insect threats? This would safeguard food production, mitigate extensive crop damage, and protect the livelihoods of farmers who face substantial economic losses due to insect infestations.

By developing an AI-driven solution that leverages the power of deep learning and computer vision, we aim to revolutionize the way insects are identified and managed on farms. This technology has the potential to significantly enhance the accuracy of insect identification, reduce the environmental impact of pesticide use, and provide farmers with valuable insights for more effective pest control strategies.

3. Ideation and Proposed Solution

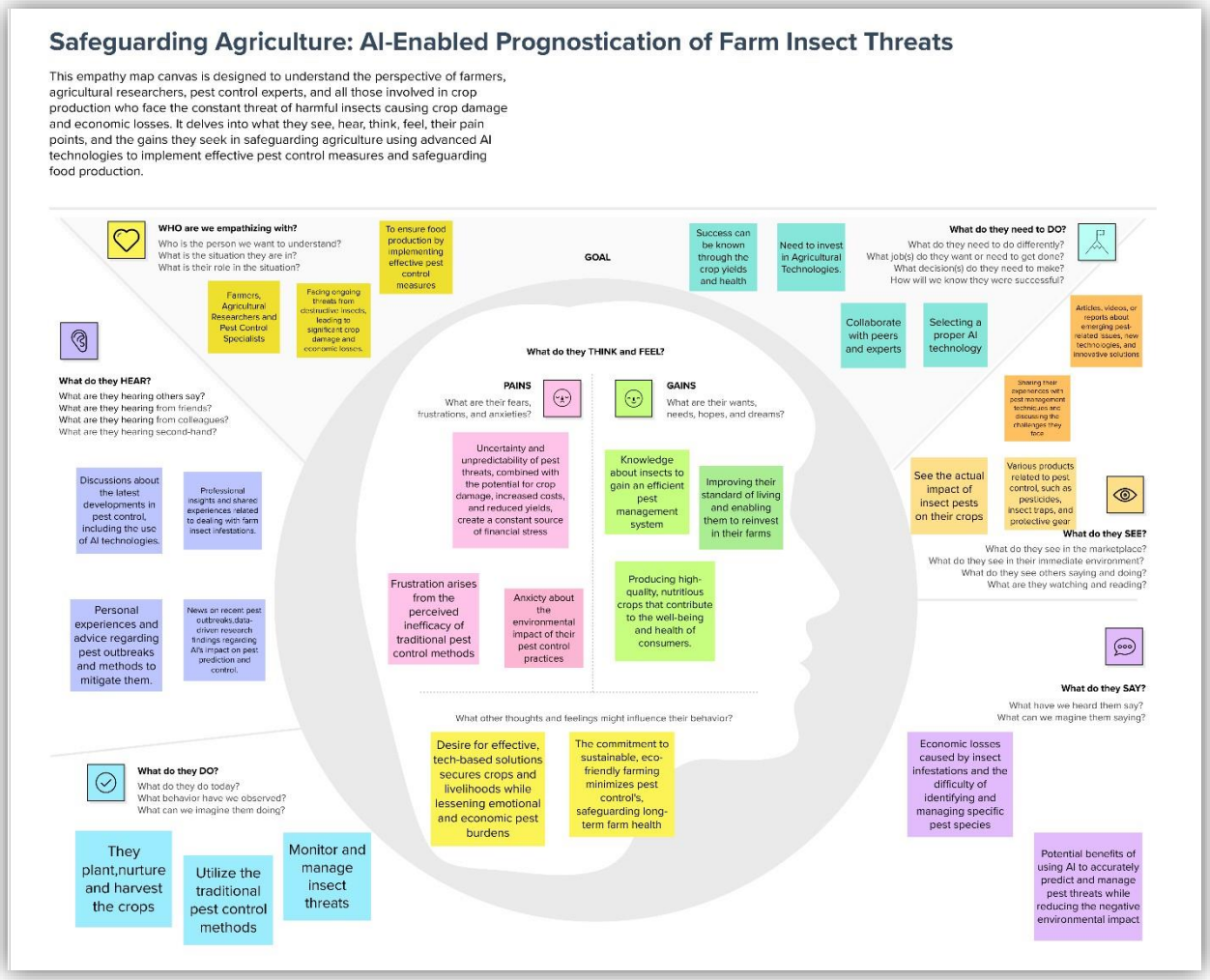
3.1 Empathy Map Canvas

Empathy is at the core of designing solutions that truly address the needs of end-users, in this case, farmers facing insect infestations. To better understand the perspectives and pain points of our target users, we've created an Empathy Map Canvas, which consists of the following key components:

- Says: What are the farmers saying? This involves understanding their challenges and concerns when it comes to insect identification and pest management.
- Thinks: What are the thoughts and feelings of the farmers? This includes their anxieties, uncertainties, and expectations regarding pest control.

- Does: What actions do farmers take? Understanding the existing practices and methods they use to identify and manage insects.
- Feels: What emotions do farmers experience? This encompasses their frustrations, fears, and desires related to insect infestations.

By considering these aspects, we aim to create a solution that aligns with the real-world needs of our endusers and provides them with a valuable tool for insect classification and pest management.



3.2 Ideation and Brainstorming

In the process of conceiving this solution, our team engaged in a thorough brainstorming exercise, considering the multifaceted challenges posed by insect threats in modern agriculture. The brainstorming session led to the development of a solution that combines innovation and practicality to empower farmers and safeguard food production. Here are the key elements that emerged from our ideation:

- Highly Accurate CNN Classification System: The cornerstone of our solution is a Convolutional Neural Network (CNN) classification system designed to deliver unparalleled accuracy in identifying and classifying insect threats. This component ensures that the system can provide precise information to farmers, enabling them to take proactive measures against insect infestations.

- User-Friendly Web Application: The web application component was conceived with usability in mind. We aimed to create a platform that is not only technologically advanced but also accessible to a wide range of users, including those with limited technical expertise. Farmers can upload images of their crops or insects with ease, making the system practical for real-world agricultural settings.
- Real-Time Results: One of the core ideas that emerged from our brainstorming sessions was the importance of real-time results. This feature empowers farmers with immediate feedback, allowing them to take prompt action to address insect threats as they arise.
- Comprehensive Recommendations: Beyond identification, the system goes a step further by recommending suitable insecticides and providing information on their potential effects. This aspect of the solution ensures that farmers receive not only identification but also actionable insights for effective pest management.
- Empowerment of Farmers: We recognized the need to empower farmers with knowledge and tools. By providing them with accurate and actionable information, our solution enhances their ability to make informed decisions, ultimately leading to greater self-reliance and confidence.
- Accessibility for All: Inclusivity was a key consideration during the ideation process. We aimed to create a solution that could be used by a broad spectrum of agricultural workers, regardless of their technological expertise. This approach promotes sustainable and data-driven agriculture while benefiting agricultural communities.

The ideation and brainstorming phase led to the development of a comprehensive and innovative solution that addresses the complex issue of insect threats in agriculture. By combining advanced technology with user-friendly features, our solution aims to contribute to crop protection, food security, and the long-term sustainability of farming operations.

4. Requirement Analysis

4.1 Functional Requirements

The functional requirements outline the specific functionalities that the system must possess to meet its objectives effectively. The core functionalities of the proposed system are as follows:

- Image Upload and Processing: Farmers should be able to upload images of insects found on their farms. The system should process the uploaded images to identify the insect species.
- Insect Classification: The system must use a Convolutional Neural Network (CNN) to classify insects accurately. It should categorize insects into one of the 15 predefined insect types.
- Recommendation System: After classification, the system should provide recommendations for suitable insecticides and pest control measures based on the identified insect species.These recommendations should consider factors like the type of insect, its life stage, and the crop affected.
- User Authentication and Management: The system should support user authentication to ensure data privacy and access control. Farmers should be able to create and manage their accounts.
- Database Management: The system should store and manage insect images, classification data, and user information for future reference and analysis.

- User-Friendly Interface: The web application should offer an intuitive and user-friendly interface to facilitate ease of use for farmers.
- Real-Time Functionality: The system should provide real-time results, allowing farmers to make immediate decisions for pest management.

4.2 Non-Functional Requirements

Non-functional requirements describe the qualities or attributes that the system should have, such as performance, security, and scalability. The non-functional requirements for this project include:

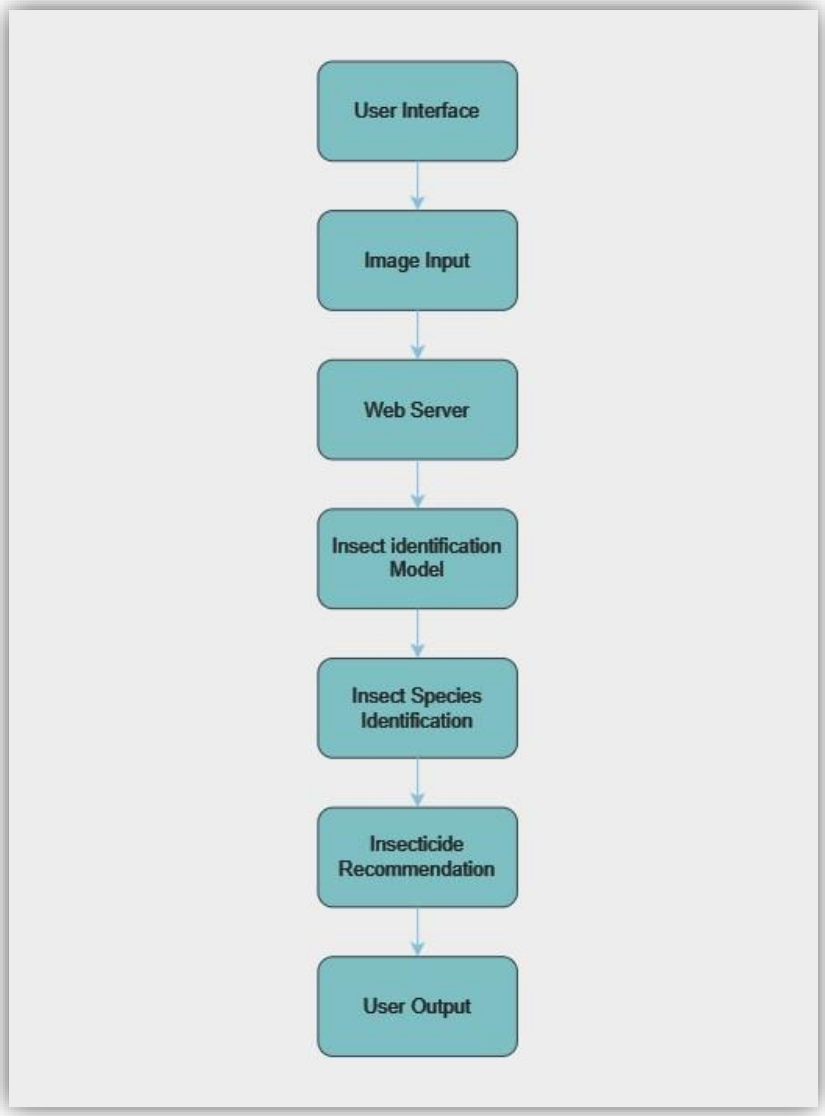
- Accuracy: The deep learning model should achieve a high level of accuracy in insect classification, minimizing misclassifications.
- Response Time: The system should deliver results within seconds to provide farmers with timely information for pest management.
- Security: User data, including uploaded images and personal information, must be securely stored and protected to maintain privacy and confidentiality.
- Scalability: The system should be designed to accommodate a growing number of users and an expanding dataset of insect images.
- Usability: The web application should be intuitive and easy to use, requiring minimal training for farmers.
- Compatibility: The application should be compatible with various web browsers and mobile devices to ensure accessibility.
- Reliability: The system should be dependable and available, with minimal downtime or service interruptions.

These requirements will serve as the foundation for the design and development of the insect classification and recommendation system.

5.2 Solution Architecture

Data Flow Diagram

The proposed solution's architecture involves a series of interconnected components that collectively enable the identification and management of insect threats in agriculture. The data flow diagram illustrates the flow of information and interactions between these components:



This data flow diagram illustrates the flow of data and interactions within the solution, starting from user input and concluding with user feedback.

User Stories

The development of this solution involves addressing specific user stories to fulfill the needs and expectations of different user types:

- Farmers and Agricultural Researchers (Mobile user):

  User Story 1 (USN-1): Users can interact with the user interface (UI) to choose an image for analysis.

  User Story 2 (USN-2): The UI includes an "Upload" button for image submission.

- Developers:

  User Story 3 (USN-3): Chosen images are sent to a Flask application for analysis.

  User Story 4 (USN-4): The application employs CNN models for image analysis.

- Data Scientist:

  User Story 5 (USN-5): Data scientists download and preprocess the dataset for model training.

  User Story 6 (USN-6): Data scientists build a VGG16 model for image analysis.

  User Story 7 (USN-7): Data scientists train the VGG16 model using a training dataset and *ImageDataGenerator* for image augmentation.

  User Story 8 (USN-8): Data scientists evaluate the model's performance on a testing dataset.

- Developer:

  User Story 9 (USN-9): Developers save the trained VGG16 model for future use and deploy it in real-world applications.

This collection of user stories reflects the various responsibilities and tasks related to the development and use of the solution.

5.2 Solution Architecture

The architecture of our insect identification and management system comprises several key components, each playing a crucial role in the process. Here are the detailed descriptions of these components:

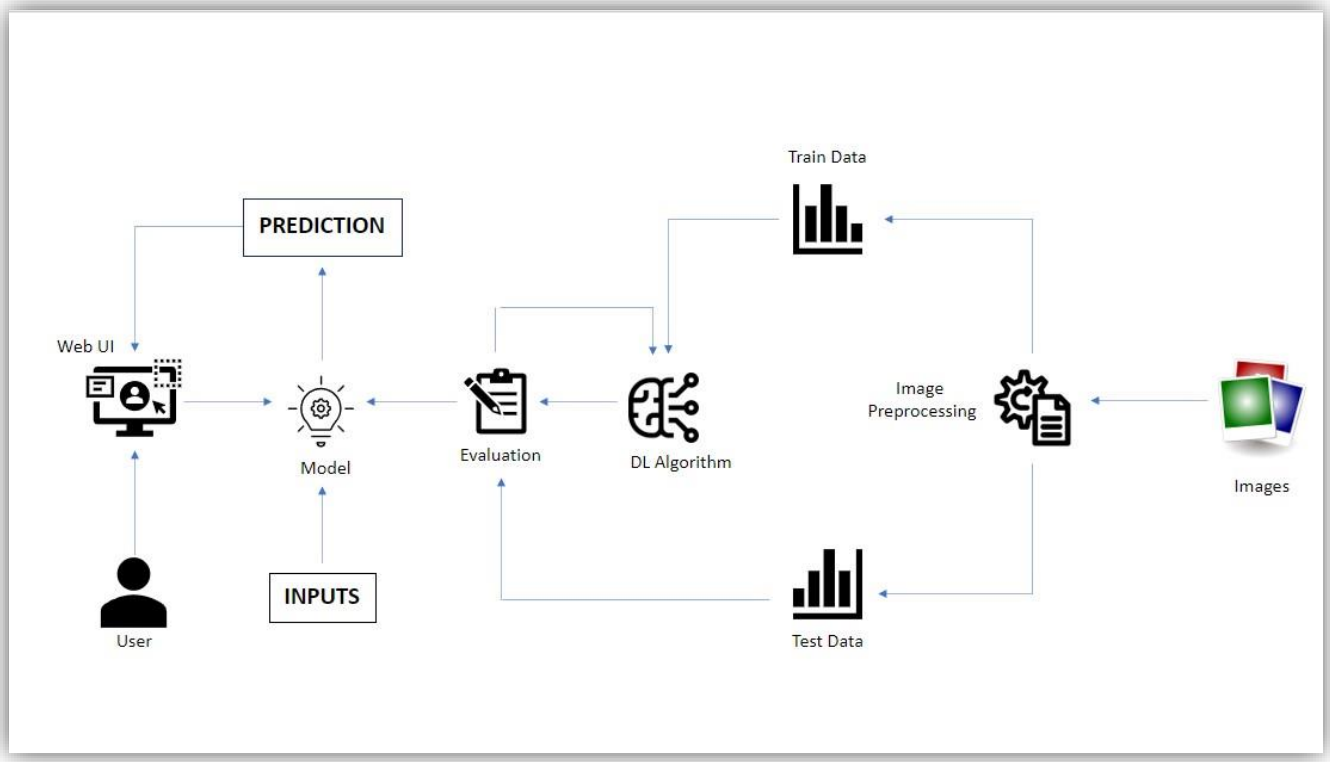
- User Interface (UI): The user interface, often accessed through web browsers, provides a userfriendly platform for users, including farmers and agricultural researchers. It serves as the entry point for interacting with the system.
- Web UI Feedback: The web interface allows users to provide feedback, enhancing user engagement and enabling continuous improvements in system performance and user experience.
- Image Input: Users can upload images of crops or insects they encounter, which serve as the primary input for the system. This component ensures the seamless flow of data into the system.
- Deep Learning Algorithm (DL Algorithm): At the core of our solution lies a deep learning algorithm, specifically a Convolutional Neural Network (CNN), which is responsible for processing the uploaded images and identifying insect species. This component is the engine of insect threat identification.
- Training Data and Testing Data: Data scientists curate and preprocess training and testing datasets, which are fundamental to the training and evaluation of the CNN model. These datasets ensure the model's accuracy and its ability to classify insect species effectively.
- Classification Model: The classification model, based on the VGG16 architecture, has been meticulously trained on the provided data. It excels in analyzing image features and accurately identifying the insect species, making it a critical component in the architecture.
- Database: The database stores important data, including insect images, classification information, and user feedback. This data repository facilitates data management, analysis, and continuous improvement of the system.

- User Output: The system provides real-time feedback to users, displaying the results of insect identification and generating recommendations for suitable insecticides. This component is essential for empowering users to make informed decisions.

Architecture Highlights

- Real-Time Processing: The architecture excels at real-time image processing, ensuring that users receive instant results and actionable recommendations.
- Data-Driven Approach: Leveraging carefully preprocessed training and testing data, the CNN model effectively classifies insect threats based on image features and characteristics.
- User-Centric Design: The user interface is intuitive and accessible, designed to cater to users with varying levels of technological expertise.

This architecture merges cutting-edge deep learning technology with a user-focused approach, providing users with the tools and knowledge needed to protect their crops, enhance food security, and promote sustainable agricultural practices.



6. Project Planning

6.1 Technical Architecture

The technical architecture of our insect identification and management system is underpinned by a combination of robust tools and technologies, ensuring efficient operation and seamless user interaction. The architecture is divided into two primary components: the Classification Algorithm and the Web Application.

- Classification Algorithm: The classification algorithm is the heart of the system, responsible for insect species identification. It is developed using Python, harnessing the power of deep learning libraries and frameworks such as TensorFlow and Keras. The choice of Python and Google Colab for model development ensures flexibility and ease of collaboration among the development team.
- Web Application: The user interface and interaction are facilitated through a web application, which is built using a stack of web development technologies. HTML, CSS, and JavaScript form the foundation for the frontend, providing a user-friendly and responsive interface. Flask, a Python web framework, is used on the backend to handle image processing and communication with the classification algorithm.

The use of these technologies guarantees a robust and accessible system for users while enabling efficient data processing and real-time insect identification.

6.2 Sprint Planning and Architecture

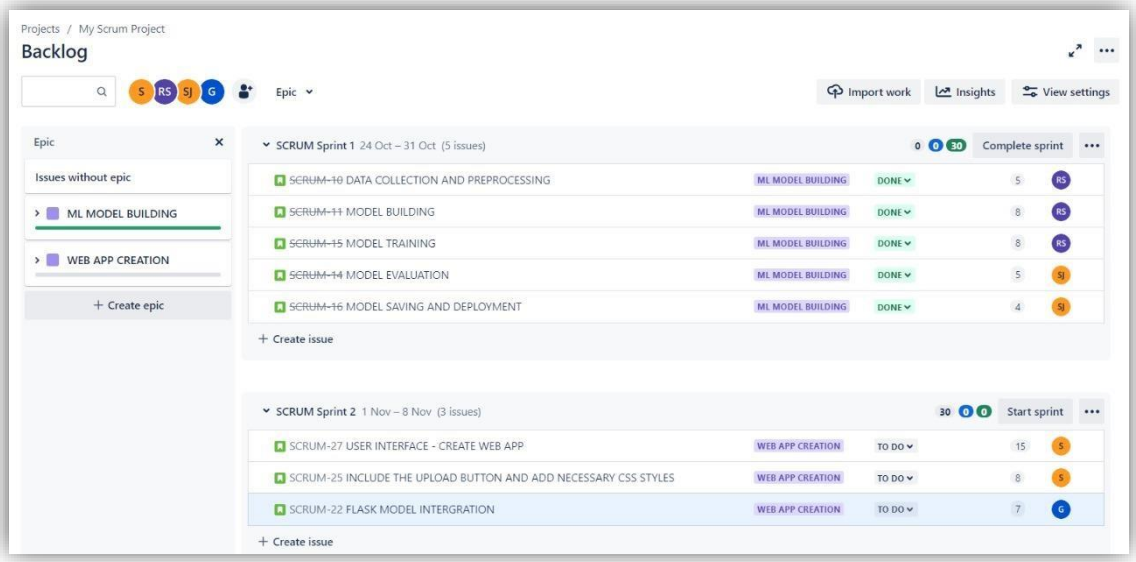
Our project follows an agile development methodology, which involves iterative sprints to deliver incremental functionality. Each sprint is carefully planned, and the architecture of the system evolves with each cycle. Sprint planning and architecture considerations are as follows:

- Iterative Development: Our project is divided into sprints, each spanning a defined timeframe. During these sprints, specific goals and tasks are outlined and executed, ensuring that the project progresses incrementally.
- Continuous Feedback: User feedback and internal evaluations play a pivotal role in shaping the architecture. They guide the implementation of new features, enhancements, and optimizations in subsequent sprints.
- Flexibility and Scalability: The architecture is designed to be flexible, allowing for the integration of additional features and technologies as the project evolves. This approach ensures scalability and adaptability to meet future requirements.

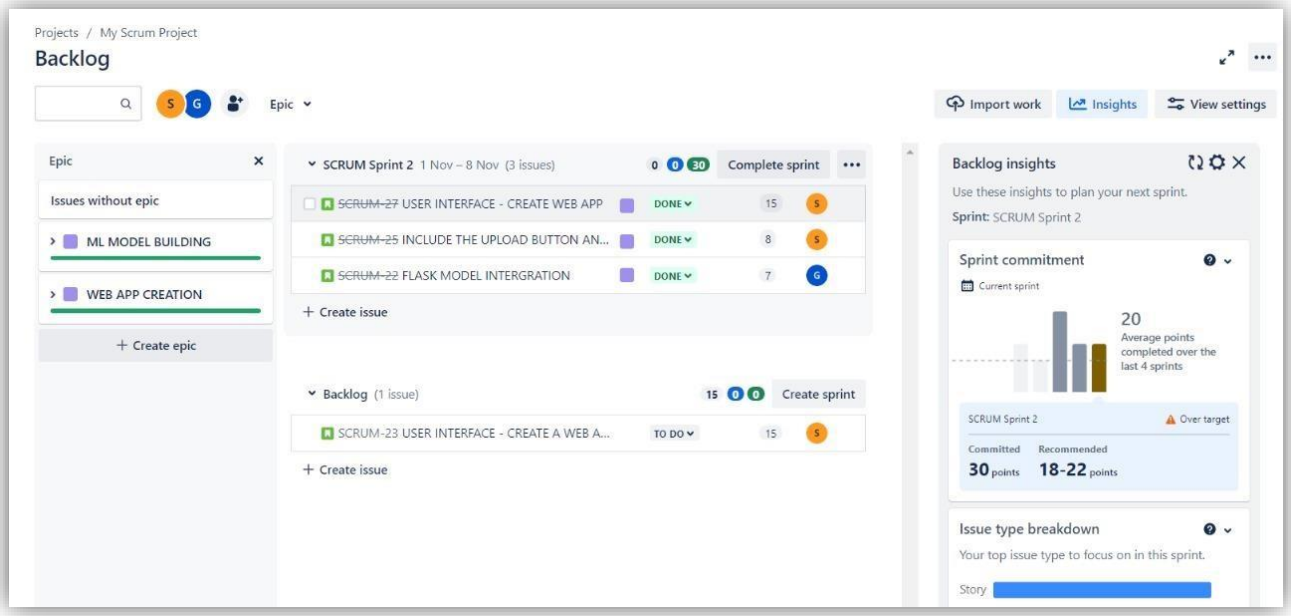
6.3 Sprint Delivery Schedule

Our project follows a structured sprint delivery schedule to ensure timely achievement of milestones and consistent progress. While the specific sprint durations and deliverables may vary, the overall schedule is as follows:

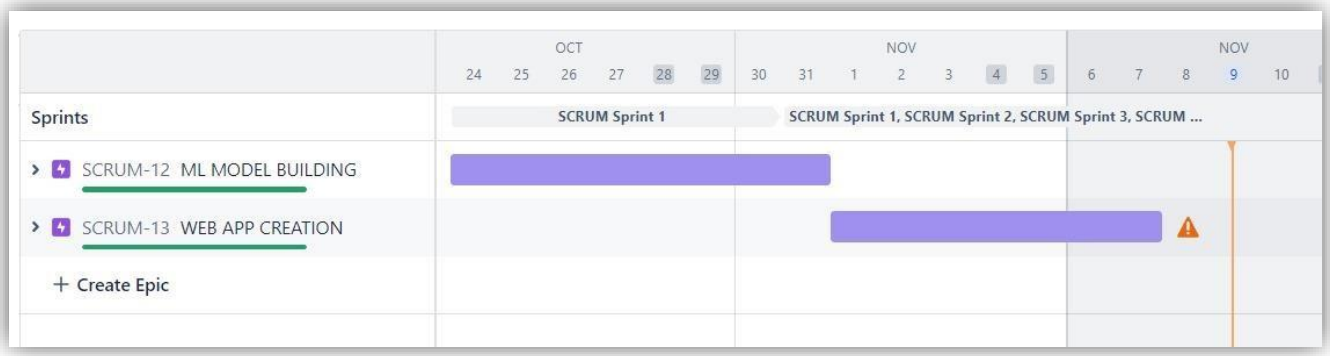
- Sprint 1 - Data Collection, Model Building, and Evaluation (Duration: 7 days) - USN-5: Data collection and preprocessing. High priority.
  - USN-6: Building a VGG16 model for image analysis. High priority.
  - USN-7: Training the VGG16 model using the training dataset and ImageDataGenerator for image augmentation. High priority.
  - USN-8: Evaluating the performance of the trained VGG16 model on the testing dataset. High priority.
  - USN-9: Model saving and deployment. Medium priority.



- Sprint 2 - User Interface and Model Integration (Duration: 7 days)
  - USN-1: Enabling user interaction with the UI for image selection. High priority.
  - USN-2: Adding an "Upload" button to the web interface for image submission. High priority.
  - USN-3: Integrating the chosen image with a Flask application for analysis. Medium priority. - USN-4: Using CNN models to analyze the chosen image. High priority.



This sprint delivery schedule outlines the completion of specific tasks and objectives in each sprint. The project's agile development approach ensures a systematic progression towards the project's goals and milestones.



7.Coding and Solutioning

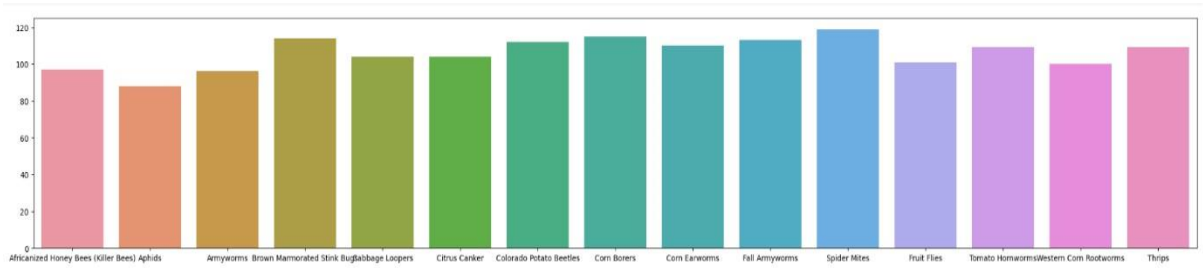
The dataset used in the code was originally obtained from Kaggle and later uploaded to Google Drive for access and use in our project.

- **Import Necessary Libraries:** This cell imports the required libraries, including os, Counter, glob, matplotlib, seaborn, and some functions from Keras.
- **List and Count Species:** It reads images from the specified directory in your Google Drive, extracts the species names, and counts the number of images per species using Counter.
- **Generate a List of Species:** It lists the species found in the training data.

```
['Africanized Honey Bees (Killer Bees)',
 'Aphids',
 'Armyworms',
 'Brown Marmorated Stink Bugs',
 'Cabbage Loopers',
 'Citrus Canker',
 'Colorado Potato Beetles',
 'Corn Borers',
 'Corn Earworms',
 'Fall Armyworms',
 'Spider Mites',
 'Fruit Flies',
 'Tomato Hornworms',
 'Western Corn Rootworms',
 'Thrips']
```

- **Display Barplot of Classes:** This cell generates and displays a bar plot of the classes (species) in your training data using Matplotlib and Seaborn.

- **Create Image Data Generators:** Image data generators are created for training and validation data. These generators perform data augmentation and preprocess the images for feeding into the neural network. '**train_gen**' rescales images, applies shear, zoom, and horizontal flip transformations, and splits data for validation. '**train_batch**' and '**valid_batch**' are the generator objects for training and validation data, respectively.



- **<u>Import VGG16 Pre-trained Model:</u>** In this part, you import the VGG16 model, a pre-trained convolutional neural network (CNN), from Keras. The model has been pre-trained on a large dataset known as ImageNet, making it capable of recognizing various objects and patterns in images. The VGG16 model has multiple layers, including convolutional and pooling layers, and it's suitable for feature extraction from images.
  You also print the number of layers in the VGG16 model to get an idea of its architecture. The VGG16 model typically has 16 layers, as the name suggests.

- **<u>Create a Custom Model:</u>** The VGG16 model, pre-trained on ImageNet, is loaded and modified to suit the specific project requirements. The input shape of the model is set to (300, 300, 3). Finetuning is performed by freezing the first 15 layers of the VGG16 base model, making them nontrainable, and then adding additional layers on top for the specific task.The added layers include a global average pooling layer, batch normalization, dropout layers, and dense layers with ReLU activation functions. The model is compiled using the Adam optimizer with a learning rate of 0.0001 and categorical crossentropy as the loss function. The model is trained using the fit method on the training data (train_batch) for 40 epochs, with a batch size of 8. Additionally, a ReduceLROnPlateau callback is used to adjust the learning rate dynamically during training based on the validation loss, with a patience of 5 epochs and a minimum learning rate of 0.00001.This fine-tuning approach is beneficial for transfer learning, where a pre-trained model is adapted to a specific task, leveraging the knowledge gained from the original task (ImageNet classification in this case). The added layers and the fine-tuning process enable the model to learn task-specific features while retaining the knowledge from the pre-trained model. The training history is stored in the history variable, allowing for analysis of the model's performance over epochs.

```
Number of layers in VGG16: 19
Model: "sequential_3"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 vgg16 (Functional)          (None, 9, 9, 512)         14714688

 global_average_pooling2d_3  (None, 512)               0
  (GlobalAveragePooling2D)

 dropout_9 (Dropout)         (None, 512)               0

 dense_9 (Dense)             (None, 512)               262656

 dropout_10 (Dropout)        (None, 512)               0

 dense_10 (Dense)            (None, 256)               131328

 dropout_11 (Dropout)        (None, 256)               0

 dense_11 (Dense)            (None, 15)                3855

=================================================================
Total params: 15112527 (57.65 MB)
Trainable params: 7477263 (28.52 MB)
Non-trainable params: 7635264 (29.13 MB)
_________________________________________________________________
```

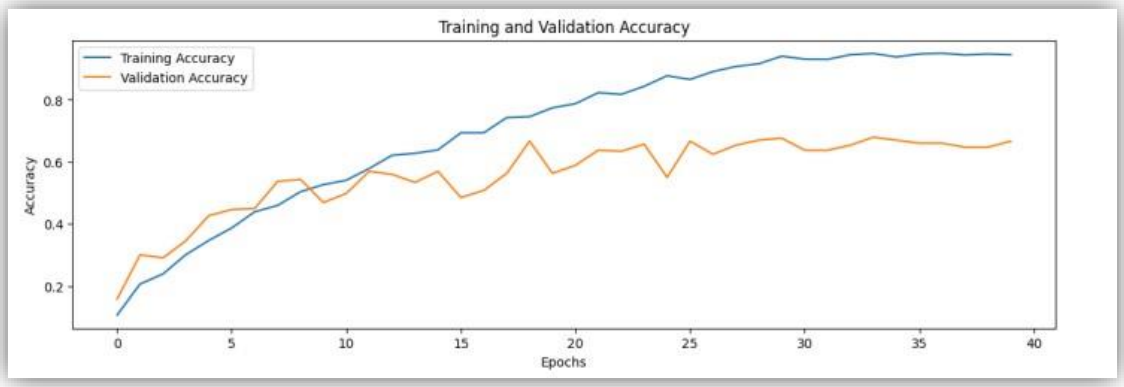
- **Compile the Model:** Once the custom model is defined, you need to compile it before training. Compiling the model involves specifying various settings for the training process. We use the Adam optimizer, which is a popular optimization algorithm for updating the model's weights during training. The loss function is set to categorical cross-entropy, which is appropriate for multi-class classification problems like yours. The model's performance during training is evaluated using accuracy as the metric. The assignment to the variable vgg in this cell is unnecessary and can be removed as it doesn't affect the model's training.

- **Train the Model:** In this step, you train the compiled model using your training data. You use the fit function, which takes the training data generator, batch size, number of training epochs, and the validation data generator as input. The model is trained for a specified number of epochs, and the training history, including metrics like loss and accuracy, is stored in the history variable. This history is useful for monitoring how the model's performance evolves during training.

```
Epoch 1/30
13/80 [===>.....................] - ETA: 39s - loss: 2.9169 - accuracy: 0.0769
/usr/local/lib/python3.10/dist-packages/PIL/Image.py:996: UserWarning: Palette images with Transparency expressed in bytes should be converted to RGBA images
  warnings.warn(
80/80 [==============================] - 75s 807ms/step - loss: 2.7619 - accuracy: 0.0567 - val_loss: 2.6913 - val_accuracy: 0.0712
Epoch 2/30
80/80 [==============================] - 63s 790ms/step - loss: 2.6809 - accuracy: 0.0850 - val_loss: 2.5875 - val_accuracy: 0.1133
Epoch 3/30
80/80 [==============================] - 60s 751ms/step - loss: 2.6133 - accuracy: 0.1087 - val_loss: 2.5653 - val_accuracy: 0.1780
Epoch 4/30
80/80 [==============================] - 71s 891ms/step - loss: 2.5724 - accuracy: 0.1283 - val_loss: 2.5085 - val_accuracy: 0.2006
Epoch 5/30
80/80 [==============================] - 61s 760ms/step - loss: 2.5290 - accuracy: 0.1512 - val_loss: 2.4375 - val_accuracy: 0.2233
Epoch 6/30
80/80 [==============================] - 60s 750ms/step - loss: 2.4665 - accuracy: 0.1843 - val_loss: 2.4133 - val_accuracy: 0.2395
Epoch 7/30
80/80 [==============================] - 60s 747ms/step - loss: 2.4292 - accuracy: 0.2071 - val_loss: 2.2779 - val_accuracy: 0.2621
Epoch 8/30
80/80 [==============================] - 60s 746ms/step - loss: 2.3356 - accuracy: 0.2386 - val_loss: 2.2532 - val_accuracy: 0.2913
Epoch 9/30
80/80 [==============================] - 63s 781ms/step - loss: 2.2001 - accuracy: 0.2803 - val_loss: 2.0775 - val_accuracy: 0.3883
Epoch 10/30
80/80 [==============================] - 60s 754ms/step - loss: 2.0905 - accuracy: 0.3031 - val_loss: 1.9349 - val_accuracy: 0.4013
Epoch 11/30
80/80 [==============================] - 61s 758ms/step - loss: 1.9207 - accuracy: 0.3622 - val_loss: 1.8471 - val_accuracy: 0.4207
Epoch 12/30
80/80 [==============================] - 61s 765ms/step - loss: 1.7594 - accuracy: 0.4118 - val_loss: 1.7970 - val_accuracy: 0.4498
Epoch 13/30
80/80 [==============================] - 61s 766ms/step - loss: 1.6874 - accuracy: 0.4457 - val_loss: 1.6649 - val_accuracy: 0.4660
...
Epoch 29/30
80/80 [==============================] - 60s 744ms/step - loss: 0.4053 - accuracy: 0.8543 - val_loss: 1.4306 - val_accuracy: 0.6440
Epoch 30/30
80/80 [==============================] - 59s 744ms/step - loss: 0.3723 - accuracy: 0.8638 - val_loss: 1.7192 - val_accuracy: 0.6408
```

8.Performance Testing:

- **<u>Plot Training and Validation Metrics:</u>** These cells generate plots to visualize the training and validation metrics (loss and accuracy) over the training epochs. The loss plot helps you see how well the model is learning. It should generally decrease over time, but if validation loss starts increasing, it might indicate overfitting.
  The accuracy plot provides insight into the model's ability to correctly classify images. You want to see this metric increase over the training epochs for both training and validation data, indicating that the model is learning to make accurate predictions.

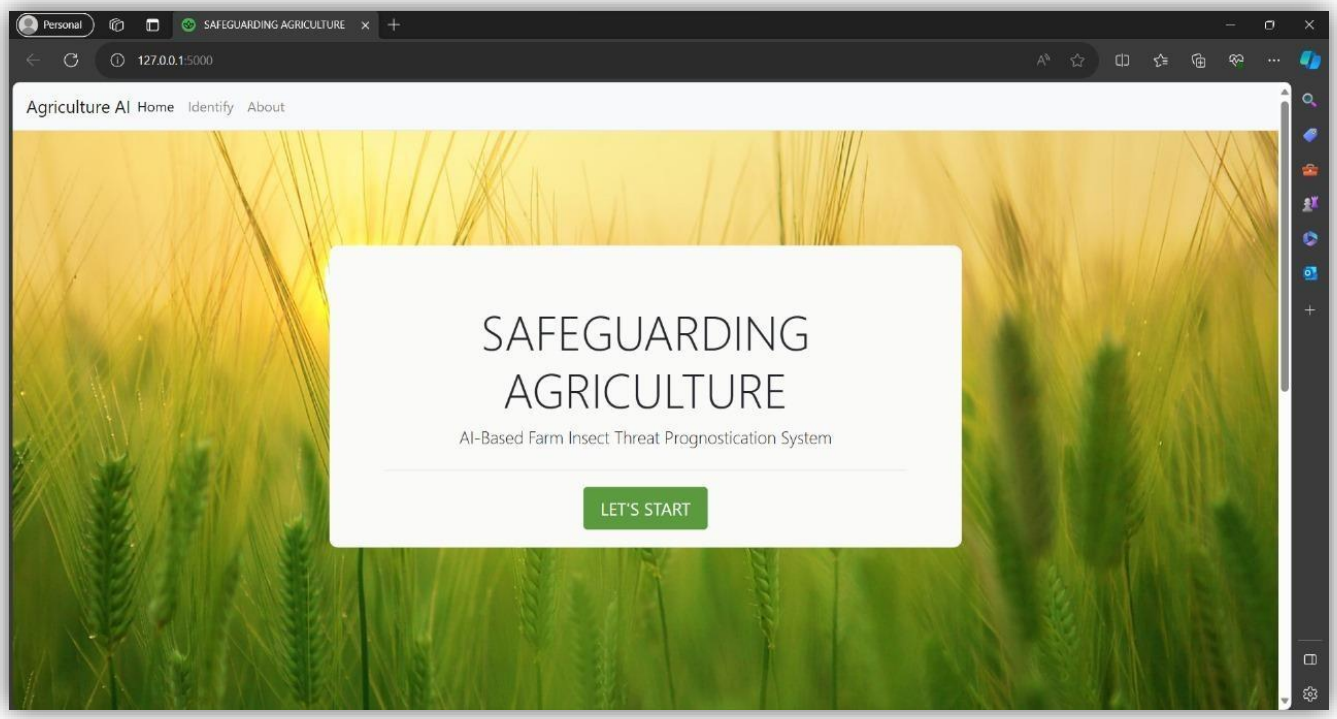


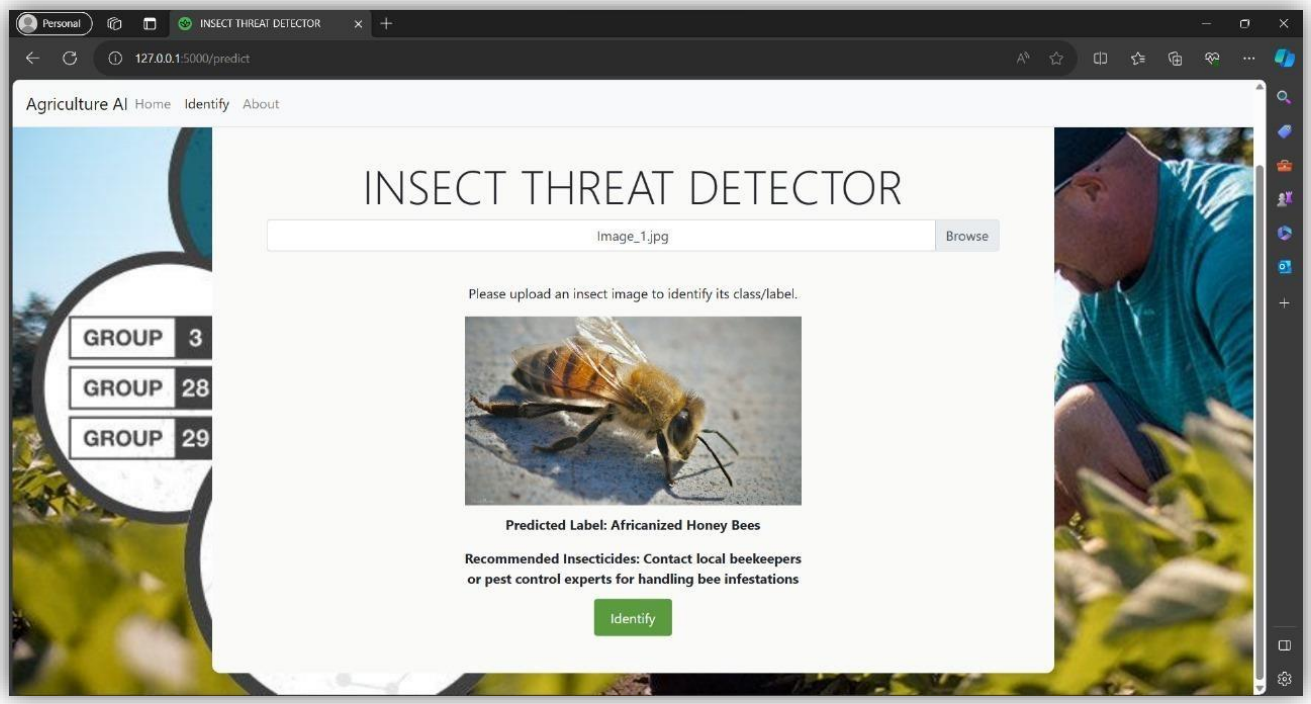
- **Make a Prediction:** In this cell, you load an image, preprocess it to match the model's input shape, make a prediction using the trained model, and identify the predicted class. It also prints the recommended insecticide for the predicted class.

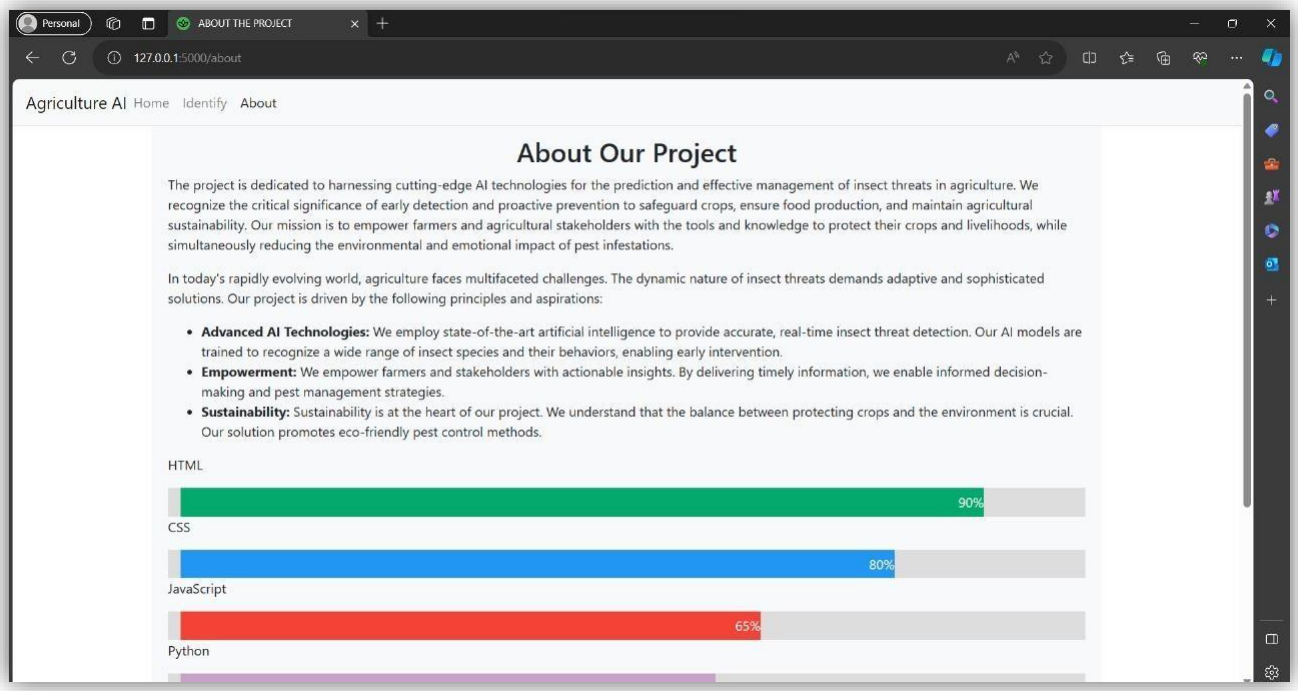
Here's an explanation of the last part (the prediction and recommended insecticide): It loads an image from the specified file. Resizes the image to the model's input shape (300x300 pixels). Converts the image to a NumPy array. Expands the dimensions of the image to match the expected input shape. Uses the trained model to make a prediction on the image. Determines the class with the highest predicted probability. Retrieves the class name from a predefined list. Prints the predicted class name and the recommended insecticide if available in the insecticides dictionary.

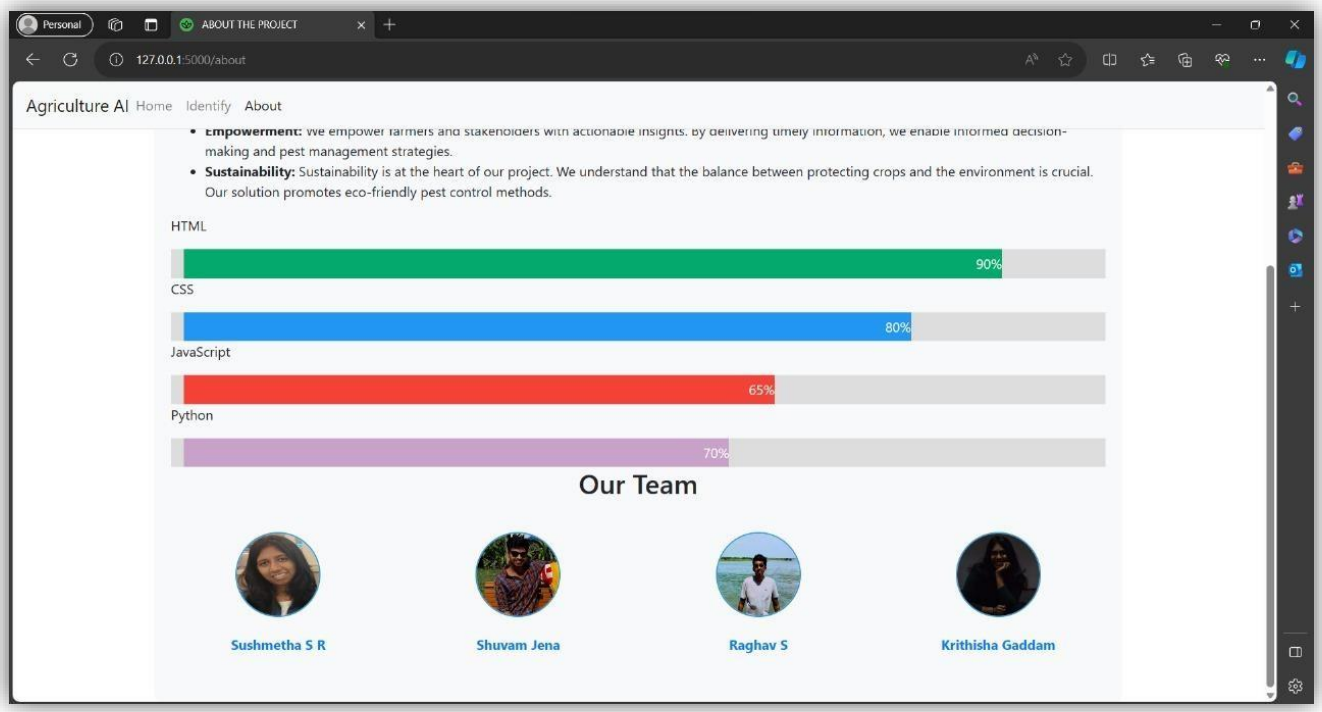
9.Output Classification:

```
1/1 [==============================] - 0s 21ms/step
Predicted Class Name: Corn Borers
For Corn Borers, recommended insecticides are: Bacillus thuringiensis (Bt), neonicotinoids
```


SAFEGUARDING AGRICULTURE
127.0.0.1:5000
Agriculture AI Home Identify About
SAFEGUARDING AGRICULTURE
AI-Based Farm Insect Threat Prognostication System
LET'S START


INSECT THREAT DETECTOR
127.0.0.1:5000/predict
Agriculture AI Home Identify About
INSECT THREAT DETECTOR
Image_1.jpg
Browse
Please upload an insect image to identify its class/label.
GROUP 3
GROUP 28
GROUP 29
Predicted Label: Africanized Honey Bees
Recommended Insecticides: Contact local beekeepers or pest control experts for handling bee infestations
Identify





10. Advantages and Disadvantages Advantages:

- Early Insect Threat Detection: The system enables early detection of insect threats in crops, allowing farmers to take timely action to prevent extensive damage.
- Accurate Identification: Leveraging a deep learning model, the system provides highly accurate insect species identification, reducing misclassification errors.
- Real-Time Results: The architecture is optimized for real-time processing, offering users instant feedback and recommendations for insect control.

- Data-Driven Recommendations: The system not only identifies insects but also recommends suitable insecticides and provides information on potential effects, empowering farmers with data-driven solutions.
- User-Friendly Interface: The user interface is designed to be accessible and intuitive, catering to users with varying technological expertise.
- Cost Savings: By promoting efficient pesticide use and preventative measures, the system contributes to cost savings for farmers.
- Food Security: The solution enhances food security by safeguarding crop yields and preventing substantial economic losses caused by insect infestations.

Disadvantages:

- Data and Model Training: The accuracy of the system relies on the quality and quantity of the training data. Acquiring and preprocessing suitable data can be time-consuming and resourceintensive.
- Technology Access: Some farmers may face challenges accessing and using the technology due to limited internet connectivity or device availability.
- System Maintenance: Ongoing maintenance and updates are required to ensure the system's accuracy and functionality.
- Data Privacy: Safeguarding user data and images is crucial, and any security breaches could lead to privacy concerns.
- Resource Intensiveness: Training deep learning models and maintaining databases can be resourceintensive and may require dedicated personnel.
- User Education: Ensuring that farmers understand and use the system effectively may require additional efforts in user education and support.
- Environmental Impact: The use of insecticides, even when recommended, can have environmental consequences, and sustainable pesticide management is essential.

It's essential to recognize both the strengths and limitations of the system to inform decision-making, address challenges, and maximize the advantages for farmers and agricultural professionals.

11. Conclusion

The development of the insect identification and management system represents a significant leap forward in the realm of modern agriculture. This solution is a testament to the power of technology and data-driven approaches in addressing the critical challenges posed by insect threats in farming. As we conclude, several key points come to the forefront:

- Empowering Farmers: The system empowers farmers and agricultural professionals by providing them with a user-friendly tool to identify and manage insect threats effectively. It not only enhances their ability to safeguard crops but also promotes self-reliance and confidence.
- Early Detection and Prevention: The ability to detect insect threats in the early stages is a gamechanger. Timely identification and data-driven recommendations mitigate extensive crop damage, ultimately contributing to food security and economic stability.
- Accuracy and Efficiency: The integration of deep learning and real-time processing ensures the accuracy of insect species identification and recommendations for insecticides. This not only saves costs but also optimizes pesticide usage.
- Inclusivity: The system is designed to be accessible to a wide range of users, promoting inclusivity in agricultural practices, regardless of technological expertise.
- Continuous Improvement: Feedback mechanisms and user education are essential for ongoing system enhancement. Regular updates, maintenance, and user support contribute to the system's long-term effectiveness.
- Environmental Considerations: While insecticides are recommended, it's crucial to underscore the importance of sustainable pesticide management and environmental protection.

In conclusion, our insect identification and management system offers a holistic and data-driven solution to a pressing challenge in modern agriculture. It represents a valuable tool for farmers and agricultural researchers, ultimately enhancing crop protection, food security, and the sustainability of farming practices.

12. Future Scope

The insect identification and management system, as it stands today, represents just the beginning of what is possible in the realm of agricultural technology. There is a vast scope for future developments and enhancements in this domain. Here are some avenues for future exploration:

- Machine Learning Advancements: Continuous research and development in machine learning and deep learning algorithms can lead to even more accurate and efficient insect identification models. Exploring advanced architectures and incorporating state-of-the-art techniques can further improve system performance.
- Integration with IoT: The integration of Internet of Things (IoT) devices can enhance real-time monitoring and data collection in the field. IoT sensors can provide valuable environmental data and enable proactive pest management.
- Crop Disease Detection: Expanding the system's capabilities to detect and manage plant diseases in addition to insect threats can offer a comprehensive solution for crop protection.
- Localization and Multilingual Support: Adapting the system to different regions and providing multilingual support can make it accessible to a broader range of farmers across the globe.
- Community Engagement: Engaging with local agricultural communities and gathering feedback can lead to tailored solutions that address specific regional challenges.

- Environmental Sustainability: Focusing on sustainable pest management practices and integrating eco-friendly solutions is essential for minimizing the environmental impact of insecticide use.
- Public-Private Partnerships: Collaborations with government agencies, research institutions, and agricultural organizations can drive research and development, and enhance the system's reach and impact.
- Education and Training: Developing training programs and resources to educate users on the system's capabilities and best practices for insect and pest management.
- Data Sharing and Research: Sharing anonymized data with researchers and government agencies can contribute to broader insights into pest behavior, environmental factors, and agricultural practices.
- AI in Agriculture: The integration of artificial intelligence into various aspects of agriculture, including crop yield prediction, soil health analysis, and climate modeling, offers immense potential for comprehensive farm management.

In essence, the future scope of the insect identification and management system is vast and promising. With the continued evolution of technology, an emphasis on sustainability, and collaboration with agricultural stakeholders, the system can play a pivotal role in promoting food security and sustainable farming practices.

13. Appendix

Source Code:

Classification Model:

```
[ ]  from google.colab import drive
     drive.mount('/content/drive')
```

```
[ ]  import os
     from collections import Counter
     path="/content/drive/MyDrive/farm_insects"
     names=[name.replace(' ','_').split('_')[0] for name in os.listdir(path)]
     classes=Counter(names)
```

```
[ ]  species_list=os.listdir(path)
     species_list
```

```
[ ]  import glob as gb
     import matplotlib.pyplot as plt
     import seaborn as sns
     class_names = []
     class_count = []
     train_examples = 0
     for f in os.listdir(path):
         files = gb.glob(pathname=str(path + "//" + f + "/*"))
         class_names.append(f)
         class_count.append(len(files))
         train_examples += len(files)
     plt.figure(figsize=(30, 5))
     sns.barplot(x=class_names, y=class_count)
     plt.show()
```

```
[ ]  from keras.preprocessing.image import ImageDataGenerator
     train_gen=ImageDataGenerator(
         rescale=1./255,
         shear_range=0.2,
         zoom_range=0.05,
         horizontal_flip=True,
         validation_split=0.2
     )
```

```
[ ]  train_batch=train_gen.flow_from_directory(
         directory=path,
         target_size=(300,300),
         batch_size=16,
         subset='training',
         class_mode='categorical',
         seed=64
     )
```

```
[ ]  valid_batch=train_gen.flow_from_directory(
         directory=path,
         target_size=(300,300),
         batch_size=16,
         subset='validation',
         class_mode='categorical',
         seed=64
     )
```

```
[ ]  from tensorflow.keras.applications.vgg16 import VGG16
     from tensorflow.keras.layers import GlobalAveragePooling2D, Dense,Dropout,BatchNormalization
     from tensorflow.keras.models import Sequential

     # Create a Sequential model
     vgg_model = Sequential()

     # Load the VGG16 base model with pre-trained weights
     vgg_base_model = VGG16(include_top=False, weights="imagenet", input_shape=(300, 300, 3))

     print(f'Number of layers in VGG16: {len(vgg_base_model.layers)}')  # Corrected parentheses

     # Set the base model to be non-trainable
     vgg_base_model.trainable = True  # Changed to True, if you want to fine-tune some layers, change this accordingly

     # Freeze the first 15 layers of the base model
     for layer in vgg_base_model.layers[:15]:
         layer.trainable = False
```

```
vgg_model.add(vgg_base_model)
vgg_model.add(GlobalAveragePooling2D())
vgg_model.add(BatchNormalization())
vgg_model.add(Dropout(0.5))
vgg_model.add(Dense(units=512, activation='relu'))
vgg_model.add(BatchNormalization())
vgg_model.add(Dropout(0.3))
vgg_model.add(Dense(units=256, activation='relu'))
vgg_model.add(Dropout(0.3))
vgg_model.add(Dense(units=15, activation='softmax'))

# Display model summary
vgg_model.summary()
```

```
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.callbacks import ReduceLROnPlateau

vgg_model = Sequential()

vgg_base_model = VGG16(include_top=False, weights="imagenet", input_shape=(300, 300, 3))

vgg_base_model.trainable = True

for layer in vgg_base_model.layers[:15]:
    layer.trainable = False

vgg_model.add(vgg_base_model)
vgg_model.add(GlobalAveragePooling2D())
vgg_model.add(BatchNormalization())
vgg_model.add(Dropout(0.5))
vgg_model.add(Dense(units=512, activation='relu'))
vgg_model.add(BatchNormalization())
vgg_model.add(Dropout(0.3))
vgg_model.add(Dense(units=256, activation='relu'))
vgg_model.add(Dropout(0.3))
vgg_model.add(Dense(units=15, activation='softmax'))

vgg=vgg_model.compile(optimizer=Adam(learning_rate=0.0001), loss='categorical_crossentropy', metrics=['accuracy'])
reduce_lr = ReduceLROnPlateau(monitor='val_loss', factor=0.2, patience=5, min_lr=0.00001)
history = vgg_model.fit(train_batch, batch_size=8, epochs=40, validation_data=valid_batch, callbacks=[reduce_lr])
```

```
[ ]  # Loss plot
     plt.figure(figsize=(30, 4))
     plt.subplot(1, 2, 1)
     plt.plot(history.history['loss'], label='Training Loss')
     plt.plot(history.history['val_loss'], label='Validation Loss')
     plt.legend()
     plt.xlabel('Epochs')
     plt.ylabel('Loss')
     plt.title('Training and Validation Loss')
```

```
[ ]  # Accuracy plot
     plt.figure(figsize=(30, 4))
     plt.subplot(1, 2, 2)
     plt.plot(history.history['accuracy'], label='Training Accuracy')
     plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
     plt.legend()
     plt.xlabel('Epochs')
     plt.ylabel('Accuracy')
     plt.title('Training and Validation Accuracy')
     plt.show()
```

```
[ ]  vgg_model.save("insect.h5")
```

```
[ ]  from tensorflow.keras.preprocessing.image import load_img, img_to_array
     import numpy as np

     # Load an image
     img = load_img('/content/drive/MyDrive/farm_insects/Armyworms/Image_1.jpg')

     # Resize the image to match the model's input shape
     img = img.resize((300, 300))

     # Convert the image to a NumPy array
     X = img_to_array(img)

     # Expand the dimensions to match the model's input shape
     X = np.expand_dims(X, axis=0)

     # Make a prediction using the vgg_model
     y_pred = vgg_model.predict(X)

     # Get the class index with the highest probability
     class_idx = np.argmax(y_pred, axis=1)[0]
```

```python
class_names = [
    'Africanized Honey Bees (Killer Bees)',
    'Aphids',
    'Armyworms',
    'Brown Marmorated Stink Bugs',
    'Cabbage Loopers',
    'Citrus Canker',
    'Colorado Potato Beetles',
    'Corn Borers',
    'Corn Earworms',
    'Fall Armyworms',
    'Fruit Flies',
    'Spider Mites',
    'Thrips',
    'Tomato Hornworms',
    'Western Corn Rootworms'
]

class_name = class_names[class_idx]
print("Predicted Class Name:", class_name)
```

```python
#Recommendation of Insecticide
insecticides = {
    'Africanized Honey Bees (Killer Bees)': 'Contact local beekeepers or pest control experts for handling bee infestations',
    'Aphids': 'Neem oil, insecticidal soap',
    'Armyworms': 'Bacillus thuringiensis (Bt)',
    'Brown Marmorated Stink Bugs': 'Pyrethroids, neonicotinoids',
    'Cabbage Loopers': 'Bacillus thuringiensis (Bt)',
    'Citrus Canker': 'Copper-based sprays, antibiotics for citrus trees',
    'Colorado Potato Beetles': 'Insecticides with active ingredients like neonicotinoids',
    'Corn Borers': 'Bacillus thuringiensis (Bt), neonicotinoids',
    'Corn Earworms': 'Bacillus thuringiensis (Bt)',
    'Fall Armyworms': 'Bacillus thuringiensis (Bt)',
    'Fruit Flies': 'Fruit fly traps, pyrethroids',
    'Spider Mites': 'Miticides, neem oil',
    'Thrips': 'Neem oil, insecticidal soap',
    'Tomato Hornworms': 'Bacillus thuringiensis (Bt)',
    'Western Corn Rootworms': 'Seed treatment with neonicotinoids'
}
if class_name in insecticides:
    if class_name=='Africanized Honey Bees (Killer Bees)':
      print(f'For {class_name}, contact local beekeepers or pest control experts for handling bee infestations.')
    else:
      recommended_insecticides = insecticides[class_name]
      print(f'For {class_name}, recommended insecticides are: {recommended_insecticides}')
```

```
from sklearn.metrics import confusion_matrix, accuracy_score, classification_report
y_pred = vgg_model.predict(valid_batch)
y_true = valid_batch.classes
# Calculate the confusion matrix
confusion_mat = confusion_matrix(y_true, y_pred.argmax(axis=1))
# Calculate the accuracy score
accuracy = accuracy_score(y_true, y_pred.argmax(axis=1))
# Generate a classification report
class_report = classification_report(y_true, y_pred.argmax(axis=1))
print("Confusion Matrix:\n", confusion_mat)
print("Accuracy Score:", accuracy)
print("Classification Report:\n", class_report)
```

GitHub Link: https://github.com/sushniaa/Farm_Insect_Threats

Project Demo Link:

https://drive.google.com/file/d/1lO13b1ejbRUA0xRepB4kx3yOkwXklOqT/view?usp=sharing