LIST OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ABBREVIATIONS

1. CNN - Convolutional Neural Networks
2. ANN - Artificial Neural Networks
3. AP - Average Presion
4. FPS - Frames Per Second

1. INTRODUCTION

1.1 Overview of The Project

Agriculture is a critical sector that supports the economy of many countries and contributes significantly to the global food supply. The agriculture and food processing industry is one of the major sectors in any country, and it plays a crucial role in expanding the export quality of agricultural and food products. However, agriculture faces various challenges, including pest attacks, which lead to a degradation of crop quality.Pests, diseases, and weeds cause massive losses to crops, resulting in a low market for the final products. Therefore, finding new ways to increase efficiency is crucial for farmers to turn a profit. Pest attacks on crops can significantly hinder crop development and yield, especially for cash crops that contribute to vast quantities of production. As a result, monitoring and evaluating the losses due to insects is necessary to ensure crop quality and safety in agriculture.

Recent developments in technology have made it possible to use machine learning and computer vision to detect and classify insects under stored grain conditions, monitor crop and soil, grade fruits, and detect plant diseases and insect pests. For instance, computer vision-based quality inspection involves acquisition, segmentation, feature extraction, and classification, while moment invariant techniques are used for extracting shape features, and neural networks are developed to classify 20 types of insect images. Researchers have also proposed super-resolution models based on deep recursive residual networks for agricultural pest surveillance and detection, and deep residual learning to improve the recognition performance for ten classes of crop insects. Other methods developed for automatic classification of field crop pests include unsupervised feature learning and multi-level classification frameworks.

Insect detection using image processing has proven to be an effective technique due to its less computation cost, fast detection, and ease of distinguishing insects with similar color and shape. Clustering segmentation with descriptors has been implemented to detect pests in grapevine with different orientations and lighting environments, while contour-based and region-based segmentation are combined to detect individual moths and touching insects. These developments have shown that it is possible to use machine learning and computer vision to detect and classify insects in agriculture, improving crop quality and safety, and increasing profitability for farmers.

This project aims to develop a computer vision system to detect insects on crops in Andhra Pradesh and Telangana, India. These two states have different climatic and environmental conditions, which lead to a different set of pests attacking the crops. The proposed system will utilize a convolutional neural network (CNN) model and object detection to identify and classify the pests present on the crops. The system will be deployed using Raspberry Pi, a low-cost single-board computer, that can be used for various IoT applications. The Raspberry Pi will be placed in the field, where it will continuously monitor the crop and detect pests.Once the system detects an insect on the crop, it will send an alert message to the farmer via SMS, indicating the presence of the pest. The system will also suggest suitable pesticides that farmers can use to control the pest. The ability to operate autonomously and provide real-time data to farmers will enable them to take prompt action, improving pest management and increasing crop yield and quality.In conclusion, the proposed system is a smart pest recommendation system that employs IoT, CNN model, and object detection to detect insects on crops in Andhra Pradesh and Telangana, India. The system's ability to operate autonomously and provide real-time data to farmers enables effective pest management. The alerts sent to farmers via SMS and the system's ability to suggest suitable pesticides help in efficient pest control. This project can help farmers in Andhra Pradesh and Telangana, India, and similar areas globally, to improve crop yield and quality, thereby increasing profitability .

1.2 Feasibility Study

The proposed project aims to develop a smart pest recommendation system for smart agriculture using IoT. The system will utilize Raspberry Pi and camera modules to detect pests on crops and alert farmers in real-time. The purpose of this feasibility study is to determine the technical, economic, and social viability of the proposed system.

1.2.1 Economical Feasibility

The economic feasibility of the proposed system is also high. The system will reduce the cost of manual pest detection and control, which is currently a time-consuming and labor-intensive process. By using the system, farmers will be able to take timely and targeted action against pests, reducing crop damage and increasing yields. The initial investment cost of the system is expected to be moderate, with the cost of Raspberry Pi and camera modules being the main expenses. However, the long-term benefits of the system in terms of increased crop yields and reduced losses will outweigh the initial investment cost.

1.2.2 Technical Feasibility

The technical feasibility of the proposed system is high. Raspberry Pi is a cost-effective and powerful device that can be easily deployed in the field. The camera module can capture high-quality images that can be used for pest detection using computer vision and deep learning algorithms. The system will use a convolutional neural network (CNN) and object detection techniques for insect identification, which have been proven to be effective in previous studies. The system will use wireless connectivity to send alerts to farmers in real-time, enabling them to take timely action.

1.2.3 Social Feasibility

The social feasibility of the proposed system is also high. The system will provide farmers with a timely and accurate way to detect pests on their crops, enabling them to take targeted action and reduce crop losses. This will improve their standard of living and financial well-being. Furthermore, the system will reduce the use of harmful pesticides, which can have negative health and environmental impacts. The system will also help to ensure food safety and security, which is a critical issue for society.

1.3 Scope

The scope of this project involves the development of a smart pest recommendation system for smart agriculture using IoT and machine learning techniques. The system is designed to address the persistent challenge of pest attacks on crops in the agriculture sector. By utilizing the Raspberry Pi and camera module, the system is able to accurately detect and classify pests on crops in real-time, thereby enabling farmers to take timely and appropriate actions to prevent crop damage and loss.The project is expected to significantly enhance the efficiency and productivity of the agriculture sector by providing a cost-effective, sustainable, and scalable solution to the problem of pest attacks on crops. Moreover, the system's low cost and ease of deployment make it accessible to farmers in remote and economically disadvantaged regions, contributing to the promotion of sustainable agriculture practices and the eradication of food insecurity in such areas.In summary, the project's scope encompasses the development of a reliable and robust smart pest recommendation system that is capable of enhancing the profitability and sustainability of the agriculture sector while improving the livelihoods of farmers.

2.LITERATURE SURVEY

The 2020 paper "Insect classification and detection in field crops using modern machine learning techniques" by Kasinathan, Singaraju, and Uyyala provides a detailed analysis of various machine learning techniques, including ANN, SVM, KNN, Naive Bayes, and CNN, for insect classification and detection in field crops. The authors emphasized the importance of pre-processing, rescaling, and augmentation of the dataset for improving accuracy. The study has the potential to inspire innovative techniques for insect detection in agriculture, which could lead to improved crop yields and reduced economic losses.

The 2020 paper "Insect Pest Detection and Identification Method Based on Deep Learning for Realizing a Pest Control System" by Kuzuhara, Takimoto, Sato, and Kanagawa proposes a novel method for detecting and recognizing insect pests using CNN-based two-stage detection and identification methods. The authors employed a region proposal network for insect pest detection using YOLOv3, and a re-identification approach using the Xception model. This study has the potential to revolutionize pest control systems by providing a more accurate and efficient approach to pest detection.

The 2018 paper "Insect Detection and Classification Based on an Improved Convolutional Neural Network" by Xia, Chen, Wang, Zhang, and Xi presents a target recognition method using an improved VGG19 model for quick and accurate insect detection. The authors utilized the "MPest" dataset to conduct experiments and compared the performance of their method with existing techniques. The findings indicate that their method exhibited superior accuracy and speed, which could lead to more efficient and effective insect detection in various fields.

The paper titled "Remote Insects Trap Monitoring System Using Deep Learning Framework and IoT" presents a novel approach to insect monitoring and identification using a four-layer IoT framework and Faster RCNN ResNet object detection framework. The proposed system achieved a high accuracy rate of 96% for built environment insects and 94% for field insects with an average processing time of 0.2 seconds per image. This study showcases the potential of IoT and deep learning-based insect monitoring schemes to automate the insect identification process, and highlights their superiority over conventional insect management systems.

Overall, these papers highlight the significance of utilizing modern machine learning techniques for insect detection and classification, which could have significant implications for improving crop yields, reducing economic losses, and revolutionizing pest control systems.

2.1 Existing System & Drawbacks

Recent developments in the field of machine learning have enabled the creation of accurate models for insect classification and detection. Through the use of feature extraction techniques, various models have been developed that can classify different categories of insect images with high precision. However, identifying and classifying insects that share similar characteristics and are located in different positions within a natural environment is a challenging task.

Previous research has explored the use of models such as ANN and SVM for insect classification, and these models have shown promising results. In this study, we present a machine learning algorithm for insect pest detection that can effectively classify and detect insects in crops such as corn, soybean, and wheat. To achieve this, we compared different models such as ANN, SVM, KNN, NB, and CNN using various insect shape features.

The proposed algorithm employs image processing techniques to segment foreground insects and locate their position in the image, making it highly efficient in terms of computation time for insect detection in agricultural fields. To conduct this research, we utilized MATLAB2018a, Python, and frameworks such as Keras and TensorFlow on an Intel core i5 processor with 16 GB of RAM.

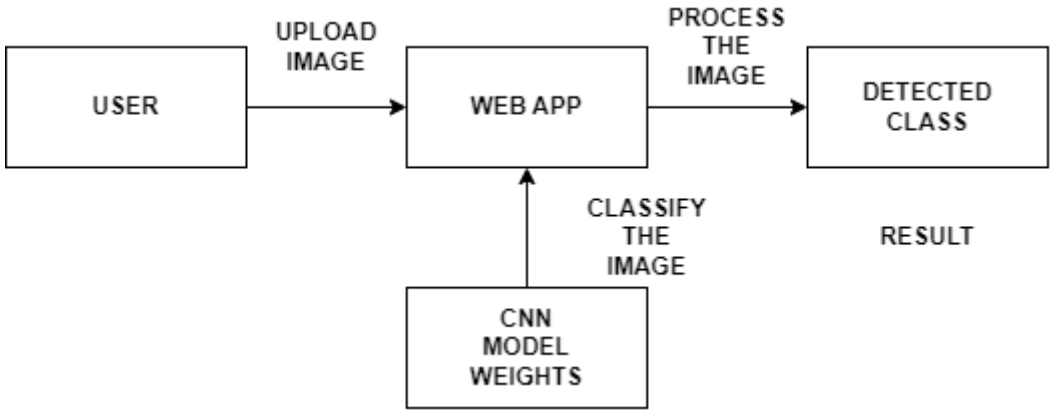

Fig 1: Block Diagram of Existing System

Limitations:

- Traditional methods for detecting and controlling pests, such as manual inspection or the use of chemical pesticides, can be time-consuming, labor-intensive, and may harm the environment.
- Some existing systems rely on satellite imagery, which may not provide sufficient resolution for accurate pest detection or may not be able to detect pests on the ground level.
- Existing systems may have limited accuracy in detecting certain types of pests or may not be effective in detecting pests during certain stages of their lifecycle.
- Some systems may be expensive to implement or require specialized equipment or training, which may not be accessible to all farmers.
- Finally, the effectiveness of some existing systems may decline over time as pests develop resistance to certain pesticides or detection methods.

2.2 Proposed System

The Smart Pest Recommendation System for Smart Agriculture using IoT is a revolutionary approach to addressing the challenges facing modern agriculture. Pest attacks on crops continue to be a major concern for farmers worldwide, leading to significant crop losses, reduced yields, and decreased profitability. In recent years, there has been a growing interest in developing innovative solutions to address this issue, and IoT and machine learning techniques have emerged as promising technologies for pest management in agriculture.

The proposed system consists of a network of IoT devices, including Raspberry Pi and camera modules, which are installed in agricultural fields to detect pests on crops. The system uses deep learning techniques, such as convolutional neural networks (CNNs), to classify images of pests and non-pests with high accuracy. When a pest is detected, the system sends an alert to farmers through various communication channels, including text messages, emails, or mobile applications, allowing farmers to take immediate actions to prevent further damage.

The system's design is scalable, allowing for the installation of multiple IoT devices in different parts of a farm, enabling real-time monitoring of pests and improved pest recommendation system. By providing real-time data on pest infestations, the system helps farmers make informed decisions about pest control measures, leading to better pest management and higher crop yields.

The Smart Pest Recommendation System for Smart Agriculture using IoT has the potential to significantly reduce the negative impact of pest attacks on crop yield and quality, leading to increased profitability for farmers. Moreover, the system's low cost and easy deployment make it accessible to farmers in remote and economically disadvantaged regions, contributing to the sustainable development of agriculture. With the proposed system, farmers can better manage pest control and optimize their crop yields, leading to a more sustainable and profitable future for agriculture.

2.2.1 Convolutional neural network

Convolutional Neural Networks (CNNs) are a class of deep neural networks that are extensively utilized for image and video recognition tasks. The primary function of CNNs is to recognize visual patterns and features in images by processing raw pixel data through several convolutional layers. These convolutional layers extract high-level features such as edges, corners, and textures from the input image, and the network then learns to classify the image based on these features.

CNNs consist of convolutional, pooling, and fully connected layers that enable them to learn features and patterns automatically from input data, making them highly effective in image and video recognition tasks. CNNs have proven to be useful in object detection, face recognition, medical imaging, natural language processing, and audio recognition due to their ability to handle large datasets.

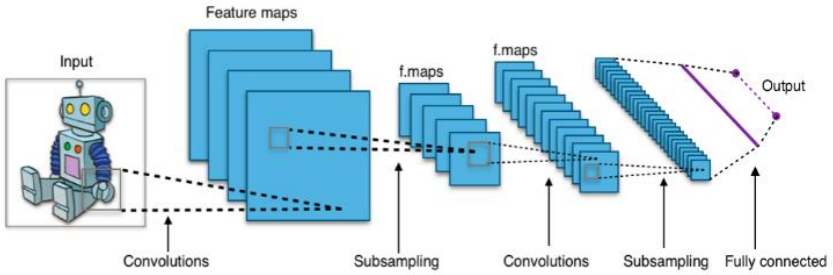

Fig 2: CNN Layers

2.2.1.2 Convolutional layer

CNNs process input data in the form of tensors, which are abstracted into feature maps through convolutional layers. This design enables data processing for each convolutional neuron's specific receptive field, reducing the number of neurons required for large inputs like high-definition photos. Convolutional layers limit the number of free parameters, enabling deeper networks, and prevent disappearing gradients and ballooning gradients issues during backpropagation. By considering spatial interactions between different features during convolution and pooling, CNNs are particularly suitable for data with a grid-like architecture, such as photographs.

2.2.1.3 Pooling layer

Pooling layers in CNNs not only reduce the dimensionality of data but also improve the robustness of the model to variations in input data. By providing translation invariance, pooling ensures that even if the position of a feature in the input image is slightly shifted, the model can still detect the feature.

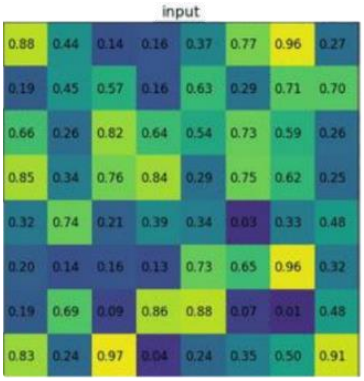

Fig 3: Image Input to CNN Classifier

The image is given as a pixel of matrix to the CNN classifier and then corresponding operations are performed like pooling, weights multiplication to obtain the filters.

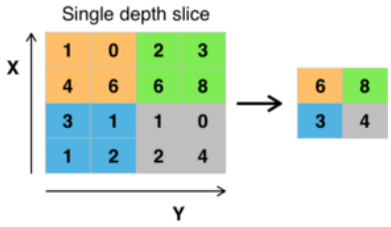

Fig 4: Pooling Layer

To apply convolutional and pooling layers in CNNs for computer vision tasks, the input data must be preprocessed, the model architecture and hyperparameters must be defined, and the model must be trained using an appropriate optimizer and loss function.

$$f_{x,y} = max_{a,b=0}^{1} S_{2X} + a, 2Y + b \qquad (1)$$

2.2.1.4 Fully connected layer

Fully connected layers in a convolutional neural network connect every neuron in one layer to every neuron in the next layer, allowing for an affine transformation through matrix multiplication and bias offset addition. This makes them crucial for the final classification task, as they capture complex relationships between the learned features from previous layers and enable the network to make a decision about the input image.

2.2.2 Image Preprocessing

Image enhancement techniques in image pre-processing are utilized to improve the quality and clarity of images, thereby increasing the accuracy of image analysis. These techniques can include sharpening the image to make it more distinct or reducing noise to improve its clarity. By employing such techniques, image quality can be improved, and insect detection and categorization can be performed with higher accuracy. In order to apply image enhancement techniques, pre-processed and segmented datasets are utilized to ensure that the images are properly prepared for analysis.

2.2.2.1 Image Augmentation

To improve the accuracy of the model and prevent overfitting, pre-processed and segmented datasets are utilized along with image augmentation techniques such as cropping flipping and rotation. These techniques expand the training set and help the model to learn a more robust set of features, leading to better performance on unseen data. By augmenting the data, the model can learn to recognize features from different perspectives, which can be useful in real-world scenarios. The insect photos are resized to a standard size, such as 227 * 227 pixels, and various techniques are used to enhance the image data.

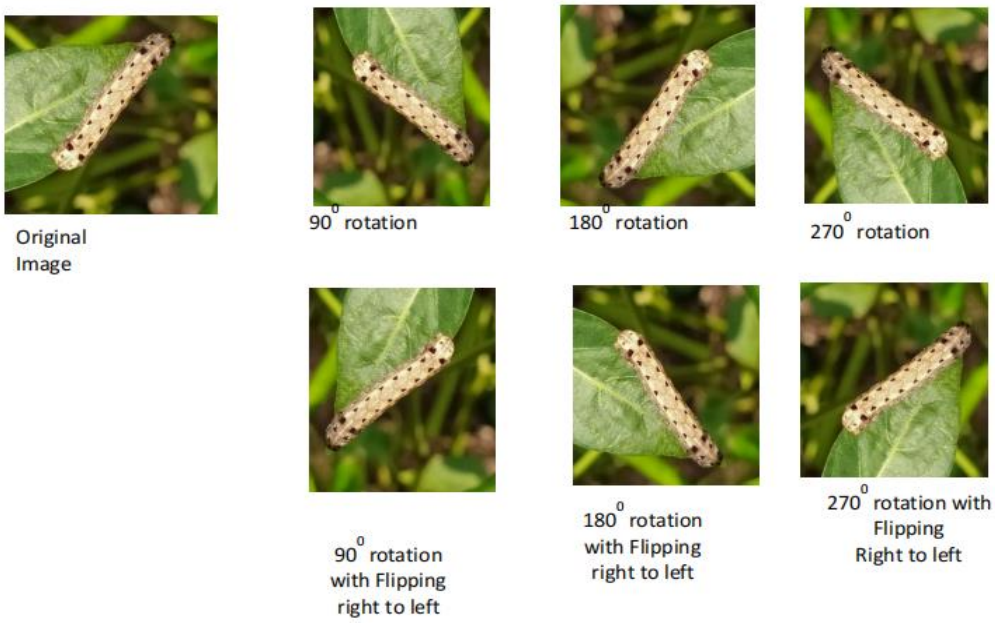

Fig 5: Data Augmentation

2.2.3 InceptionV3 – Transfer Learning

Inception-v3 is a convolutional neural network architecture that was introduced by Google in 2015. It was designed to achieve a better trade-off between accuracy and computational efficiency compared to previous deep learning models, such as AlexNet and VGGNet.

The key innovation in Inception-v3 is the Inception module, which is a type of block that uses multiple parallel convolutional layers with different kernel sizes and pooling operations. This allows the network to capture both local and global features of the input image at different scales, leading to improved accuracy.

Another important aspect of Inception-v3 is its use of factorization, which involves breaking a large convolutional layer into smaller ones with reduced depth, followed by a 1x1 convolutional layer. This reduces the number of parameters in the network and allows for better computational efficiency.

In addition to these innovations, Inception-v3 also incorporates other techniques such as batch normalization and dropout, which improve the robustness of the network and prevent overfitting. Inception-v3 has achieved state-of-the-art results on a range of image classification benchmarks, including the ImageNet Large Scale Visual Recognition Challenge (ILSVRC). Its high accuracy and computational efficiency have made it a popular choice for a variety of computer vision applications, including object detection, segmentation, and recognition in real-time systems.

Overall, Inception-v3 is a powerful deep learning model that has advanced the state of the art in computer vision and has been used in many real-world applications.

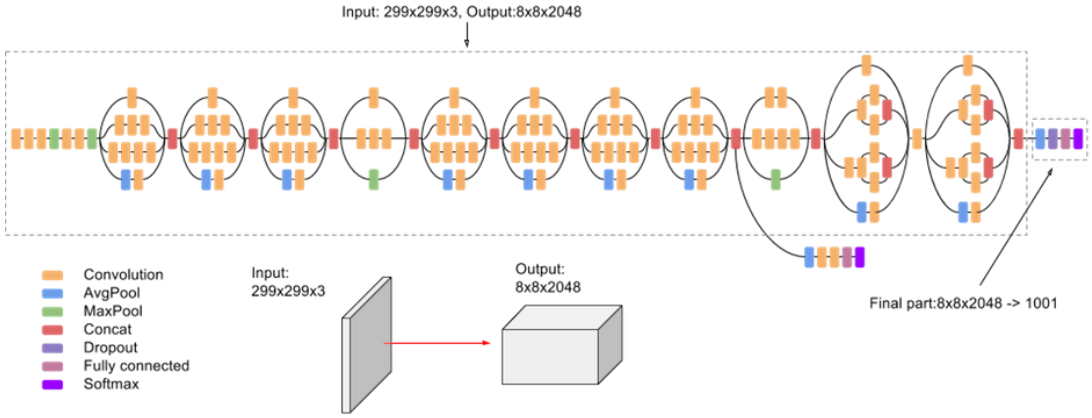

Fig 6: Architecture of Inception V3

2.2.4 Dataset and Samples for CNN model

In this study we have three classes.
They are as follows:

1. Ladha
2. Black pest
3. Leaf

We have gathered photographs from nearby Telangana and Andhra Pradesh agricultural areas for each and every class. We have 1503 pictures in total. The number of photographs we have gathered for each class is shown in the table below.

Table 1: Dataset for CNN Model

Class	No of images	Train	Test
Ladha	736	686	50
BlackPest	512	462	50
Leaf	255	205	50
Total	1503	1353	150

Some of the images are shown below:

Fig 7: Black Pest

Fig 8: Ladha

Fig 9: Leaf

2.2.5 Image Classification Methodology

The method of classifying insects comprises several processes that categorize insects according to their attributes using image processing and machine learning techniques. The following flowchart shows the process of creation of a CNN classifier model.

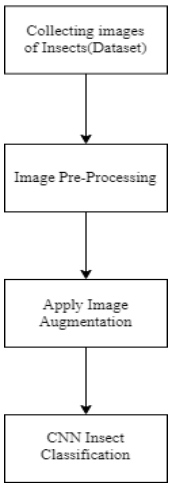

Fig 10: CNN classification flowchart

2.2.6 Raspberry pi

Raspberry Pi is a series of small single-board computers developed by the Raspberry Pi Foundation. They are low-cost, credit-card-sized computers that are designed to be easy to use and program. Raspberry Pi boards are equipped with various input/output (I/O) interfaces such as USB, HDMI, Ethernet, and GPIO pins, which allow users to interact with other devices and sensors.

GPIO (General Purpose Input/Output) pins are a type of interface that allows the Raspberry Pi to interact with external devices and sensors. These pins can be programmed to send or receive digital signals, allowing the Raspberry Pi to control external devices or receive data from sensors. The number of GPIO pins varies depending on the model of Raspberry Pi board, but typically ranges from 26 to 40 pins.

In addition to the GPIO pins, the Raspberry Pi board also has other interfaces such as USB and Ethernet ports, HDMI output, and a camera interface. These interfaces allow the Raspberry Pi to be used for a wide range of applications, including robotics, home automation, and IoT (Internet of Things) devices.

Overall, Raspberry Pi is a versatile and affordable computing platform that can be used for a wide range of projects and applications. Its GPIO pins provide an easy way to interface with external devices and sensors, making it a popular choice for DIY enthusiasts, hobbyists, and professionals alike.

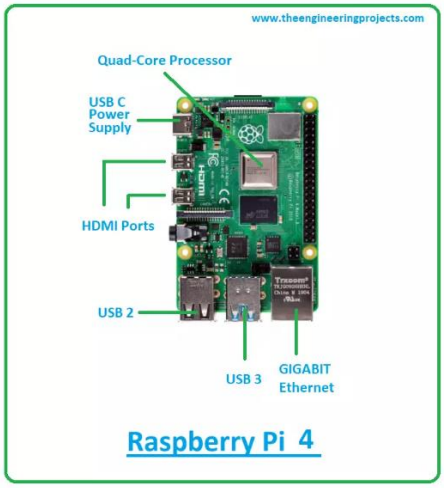

Fig 11: Raspberry Pi 4

2.2.7 Object Detection

For insect detection, a CNN was trained on a dataset of images containing various types of insects. The model was able to classify the images. However, when the model was deployed in the field, it was observed that it was classifying even unnecessary images, which could lead to unnecessary use of pesticides. To overcome this issue, object detection was used.

2.2.7.1 Object detection using TFLite Model Maker

Object detection is a technique used to indicate the objects in an image and their locations. TFLite Model Maker is a tool that simplifies the process of creating object detection models. The tool allows the user to create an object detection model by providing a few parameters and a dataset of images. The tool then trains the model and exports it in the TensorFlow Lite format, which can be used on devices with low computational power such as Raspberry Pi.

2.2.7.2 Data preparation for object detection

To train the object detection model, a dataset of insect images was utilized, and bounding boxes were used to label the locations of the insects in the images. In order to make the size of the dataset more and add variability, data augmentation techniques are helpful, such as rotation, zooming and flipping. These preprocessing steps are essential to improving the accuracy and robustness of the model during both training and testing.

Table 2: Dataset for Object Detection

Class	No of images	Train	Test
Ladha	150	100	50
BlackPest	160	110	50
Leaf	110	80	30
Total	420	290	130

2.2.7.3 Model architecture

EfficientDet-Lite2 is a smaller, computationally efficient version of the EfficientDet model that is designed for object detection tasks. The model consists of a backbone network based on a modified MobileNetV2 architecture that extracts features at multiple scales and a prediction network that performs object detection using anchor boxes. The training process is simplified using the create function of the tflite_model_maker.object_detector module, which takes in the training data, model architecture, and hyperparameters such as batch size, number of epochs, and optimizer. Object detection models predict bounding boxes and class probabilities for objects in an image. The output of the model includes a set of bounding boxes with confidence scores and corresponding class labels, which identify the type of object contained within the box.

2.2.7.4 Model training

Transfer learning was used to detect insects in an image dataset using the EfficientDet-Lite2 model. The DataLoader class from the tflite_model_maker.object_detector module was used to prepare the dataset with images labeled for insect location through bounding boxes. The dataset was split into 2 sets of training and validation, with the training set used to fine-tune the model for 20 epochs with a batch size of 4.

The create function of the tflite_model_maker.object_detector module was used to perform the training process by providing the training data, the model architecture, and other hyperparameters such as batch size, number of epochs, and optimizer.

2.2.7.5 Model evaluation

To evaluate the performance of the trained model, the evaluate function of the tflite_model_maker.object_detector module was used. The evaluate function takes as input the validation data and returns the average precision (AP) for each class.The trained EfficientDet-Lite2 model achieved an AP of 0.624 for the BlackPest class, an AP of 0.544 for the Ladha class, and an AP of 0.542 for the Leaf class on the validation set. The AP is a most used metric for evaluating the performance of object detection models. It measures the precision of the model at different levels of recall, which is the fraction of ground-truth objects that are correctly detected by the model.

3.SYSTEM ANALYSIS

3.1 Overview of System Analysis

System analysis is a critical process in the development of any software application. It involves understanding and defining the requirements, designing the system architecture, implementing the solution, testing it, and deploying it to the intended environment. In the context of the insect identification and alert system project, system analysis is necessary to ensure that the software application meets the requirements of the stakeholders and functions as expected.

Fig 12: SDLC Life Cycle

3.1.1 Requisites Accumulating and Analysis

Before the actual development process can begin, it is essential to gather and analyze the requirements of the system. The requirements can be gathered from various sources, including the stakeholders, domain experts, and industry standards. The requirements need to be documented and analyzed to ensure that they are complete, unambiguous, and feasible. The system analysis team needs to ensure that the requirements align with the goals of the project.

3.1.2 System Design

Once the requirements have been gathered and analyzed, the system design phase can begin. In this phase, the architecture of the system is defined, and the software components are identified. The design phase is crucial in ensuring that the system meets the requirements, is scalable, and can be implemented within the available resources. In the insect identification and alert system project, the design phase involved choosing the appropriate hardware and software components, defining the system architecture, and designing the user interface.

3.1.3 Implementation

After the design phase, the implementation phase begins, which involves coding and integrating the software components. The implementation phase includes tasks such as coding, unit testing, integration testing, and debugging. In the insect identification and alert system project, the implementation phase involved building the object detection and classification models using TensorFlow and Keras. It also involved integrating the models with the Raspberry Pi camera module and configuring the Twilio API for sending SMS notifications to farmers.

3.1.4 Testing

The testing phase is critical in ensuring that the software application meets the requirements and functions as expected. The testing phase includes various types of testing, such as unit testing, integration testing, system testing, and acceptance testing. In the insect identification and alert system project, the testing phase involved testing the object detection and classification models using a test dataset of images. It also involved testing the integration of the models with the Raspberry Pi camera module and the Twilio API.

3.1.5 Deployment of System

After the testing phase, the software application is ready for deployment to the intended environment. The deployment phase involves installing the software application on the target hardware, configuring the software, and ensuring that it functions correctly. In the insect identification and alert system project, the deployment phase involved installing the software application on the Raspberry Pi device and ensuring that it functions as expected in the field.

3.1.6 Maintenance

After the deployment phase, the maintenance phase begins, which involves ensuring that the software application remains functional and up-to-date. The maintenance phase includes tasks such as bug fixing, patching, and upgrading the software components. In the insect identification and alert system project, the maintenance phase involves monitoring the software application's performance and making necessary changes to ensure that it remains functional and up-to-date.

3.2 Software Used in The Project

The software used in this project played a crucial role in the success of the system. Two main tools were utilized: Google Colab and Jupyter Notebook.Google Colab was used to train the object detection model based on the Efficient LiteDet2 architecture. The object detection dataset contained around 700 images, and these images were labeled using the LabelImg software in the Pascal VOC format. The labeled dataset was then fed into Google Colab, which provided a cloud-based environment for the creation and training of the model. The efficiency of Google Colab helped in accelerating the training process and minimizing the time taken for model development.

On the other hand, Jupyter Notebook was used for the creation and training of the CNN model based on the Inception V3 architecture. The dataset for this model contained around 1500 images of three classes: Ladha, Black Pest, and damaged leaves of cotton and chili. The Jupyter Notebook environment allowed for the creation and testing of the model in a user-friendly and interactive manner.

Overall, the combination of Google Colab and Jupyter Notebook provided an efficient and user-friendly platform for the development and training of both the object detection and CNN models.

3.3 Libraries & Modules

NumPy: NumPy is a Python library used for numerical computing. It is a fundamental library for scientific computing with Python and provides support for large, multi-dimensional arrays and matrices, along with a large collection of high-level mathematical functions.

Keras: Keras is a high-level neural networks API, written in Python and capable of running on top of TensorFlow, CNTK, or Theano. It is designed to enable fast experimentation with deep neural networks and provides a user-friendly interface for designing and training neural networks.

TensorFlow: TensorFlow is an open-source machine learning framework developed by Google. It is used to build and train machine learning models and has a large ecosystem of tools and libraries that make it easy to build, deploy, and scale machine learning applications.

Pandas: Pandas is a library for data manipulation and analysis in Python. It provides data structures for efficiently storing and manipulating large datasets and includes tools for data cleaning, transformation, and analysis.

Matplotlib: Matplotlib is a data visualization library in Python. It provides a range of tools for creating static, animated, and interactive visualizations in Python.

TFLite Model Maker: TFLite Model Maker is a library that simplifies the process of building machine learning models for deployment on mobile and edge devices. It provides pre-trained models and tools for fine-tuning them on custom datasets.

TFLite Support: TFLite Support is a library that provides tools and utilities for developing machine learning applications on mobile and edge devices. It includes a range of features for optimizing and deploying machine learning models on different hardware platforms.

TFLite: TFLite is a library developed by Google to enable machine learning models to run on mobile and embedded devices with low latency and small memory footprint. It allows the deployment of models on resource-constrained devices by converting them to a format that can be executed locally. TFLite supports a wide range of hardware platforms and provides a set of tools for model optimization and conversion. It also includes an interpreter for executing the models and a runtime for accelerating the execution on compatible devices.

Micro Python: MicroPython is a software implementation of the Python 3 programming language that has been optimized to run on microcontrollers and embedded systems. It offers an interactive prompt, an extensive collection of libraries, and the ability to control hardware devices, making it an ideal choice for projects that require a small form factor and low power consumption. With MicroPython, developers can write code in a familiar language and take advantage of the flexibility and ease-of-use that Python offers.

3.4 SYSTEM REQUIREMENTS

3.4.1 Software Requirements

- Google Colab
- Jupyter Notebook
- Numpy,Keras,Tensorflow,Pandas,Matplotlib,Tflite_model_maker,Tflite_support,Tflite, Micro Python
- Operating Systems:Windows,Linux

3.4.2 Hardware Requirements

Windows

- Processor: Intel Core i5 or higher
- RAM: 8 GB or higher
- Storage: 256 GB SSD or higher

Linux – Raspberry pi

- Raspberry Pi 4 Model B
- 8 GB RAM
- 32 GB microSD card
- Raspberry Pi Camera Module V2
- 5V 3A USB-C power supply
- HDMI cable
- Ethernet cable
- Keyboard and mouse

4. SYSTEM DESIGN

4.1 Overview of System Design

The development of the insect identification system was a challenging yet exciting process that required a comprehensive approach to ensure its accuracy and reliability. By utilizing cutting-edge machine learning techniques, efficient models, and state-of-the-art software tools, I was able to build a system that provides an innovative solution to the problem of insect pest detection in agriculture.

To train the CNN and object detection models, I used a carefully curated dataset that was labeled with meticulous attention to detail using LabelImg software. The use of Efficient LiteDet2 model for object detection ensured that the system was efficient and lightweight, making it easy to deploy on small devices like the Raspberry Pi. The use of Google Colab, Jupyter Notebook, and TensorFlow allowed for an easy-to-use and interactive environment to experiment with different models and techniques.

The hardware requirements of the system were carefully considered to ensure that it is both powerful and scalable. The Raspberry Pi with a camera module, internet connection, and Twilio account for sending messages are the key components of the system. With at least 2GB of RAM and a 16GB microSD card, the Raspberry Pi has ample resources to store the models and the dataset.

The insect identification system provides farmers with an efficient and reliable tool to monitor their fields and detect insect pests with accuracy and speed. By identifying specific types of insects and suggesting appropriate pesticides, the system can help farmers prevent crop damage and save valuable resources. Overall, the insect identification system represents a major advancement in the field of agriculture and demonstrates the potential of machine learning to transform traditional farming practices.

System Design:

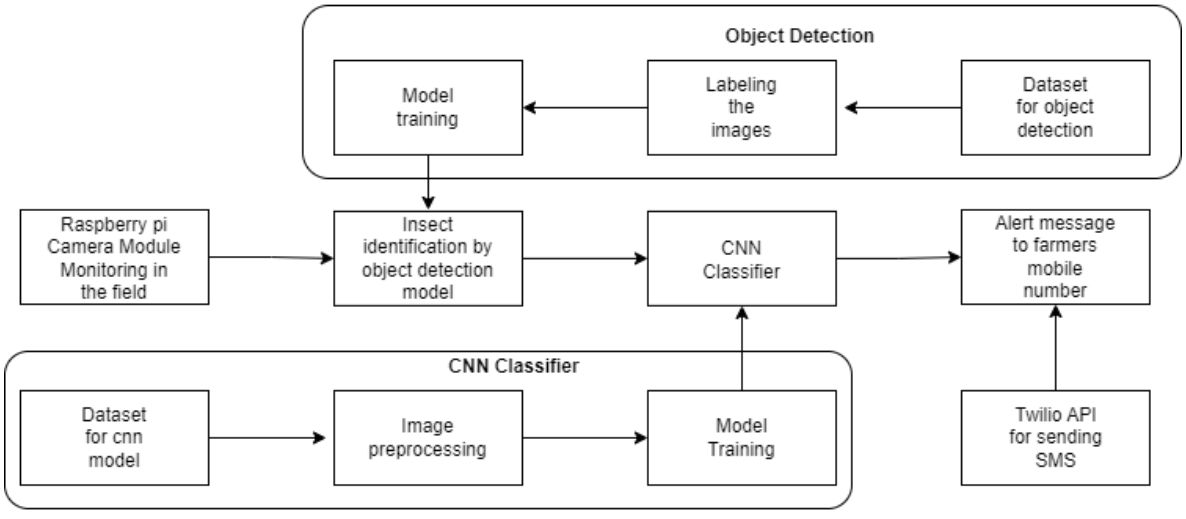

Fig 13: System Design

The system design for the insect identification and alert project involves several components working together seamlessly to achieve the desired outcome. At the heart of the system is the Raspberry Pi camera that is responsible for monitoring the fields for any insect infestations. The camera is connected to a Raspberry Pi board that runs the object detection and classification algorithms.

To perform object detection, the system uses the Efficient-LiteDet2 model, which has been trained on a dataset of approximately 700 images. The object detection dataset was created by using the LabelImg software to label the images in PascalVOC format. The model is run on a Google Colab notebook, and the output is fed to the classification model.
The classification model is based on the Inception V3 architecture, which has been converted to a lite model for deployment on the Raspberry Pi board. The classification model is trained on a dataset of 1500 images, consisting of three classes - Ladha, Black Pest, and damaged leaves of cotton and chili.

When an insect is detected, the image is saved and loaded into the classification model, which classifies the insect and sends a message to the farmer via the Twilio API. The message includes information about the type of insect and a suggestion on which pesticide to use to remove the infestation.

Overall, the system design is robust, efficient, and easy to use. It provides a cost-effective and reliable solution for detecting and identifying insects in crops, helping farmers take prompt action to protect their crops and minimize losses.

5.CODING & IMPLEMENTATION

5.1 Inception V3 CNN Classifier

```
ORG_DIR=r"C:\Users\ASUS\Downloads\final dataset\final dataset"
CLASS=['BlackPest','Ladha_chilli','Leafs_chilli']

import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
import h5py
from keras.layers import Dense,Flatten
from keras.models import Model
from keras.applications.inception_v3 import InceptionV3,preprocess_input
from keras_preprocessing.image import ImageDataGenerator,load_img,img_to_array
import keras

base_model=InceptionV3(input_shape=(256,256,3),include_top=False)

for layer in base_model.layers:
  layer.trainable=False

x=Flatten()(base_model.output)
x=Dense(units=3,activation='sigmoid')(x)

#Final Model
model=Model(base_model.input,x)

#compile the model
model.compile(optimizer='adam',loss=keras.losses.categorical_crossentropy,metrics=['accura
cy'])
#summary
model.summary()

#preprocessing
train_datagen=ImageDataGenerator(featurewise_center=True,
                                 rotation_range=0.4,
                                 width_shift_range=0.3,
                                 horizontal_flip=True,
                                 preprocessing_function=preprocess_input,
                                 zoom_range=0.4,
                                 shear_range=0.4)
train_data=train_datagen.flow_from_directory(directory=r"C:\Users\ASUS\Downloads\final
dataset\final dataset\train",
```

```
                                target_size=(256,256),
                                batch_size=36)
Found 1453 images belonging to 3 classes
train_data.class_indices
{'BlackPest': 0, 'LadhaChilli': 1, 'LeafsChilli': 2}

#model checkpoint
from keras.callbacks import ModelCheckpoint,EarlyStopping
mc=ModelCheckpoint(filepath="best_model.h5",
                   monitor="accuracy",
                   verbose=1,
                   save_best_only=True)
es=EarlyStopping(monitor="accuracy",
                 min_delta=0.01,
                 patience=5,
                 verbose=1)
cb=[mc,es]

his=model.fit_generator(train_data,
                        steps_per_epoch=10,
                        epochs=30,
                        callbacks=cb)


from keras.models import load_model
model=load_model("model_weights.h5")

#tensorflow model convertion to tflite
TF_LITE_MODEL_FILE_NAME = "tf_lite_model.tflite"

import tensorflow as tf
tf_lite_converter = tf.lite.TFLiteConverter.from_keras_model(model)
#tf_lite_converter.optimizations = [tf.lite.Optimize.OPTIMIZE_FOR_SIZE]
tf_lite_converter.optimizations = [tf.lite.Optimize.DEFAULT]
tf_lite_converter.target_spec.supported_types = [tf.float16]
tflite_model = tf_lite_converter.convert()

tflite_model_name = TF_LITE_MODEL_FILE_NAME
open(tflite_model_name, "wb").write(tflite_model)

def convert_bytes(size, unit=None):
    if unit == "KB":
        return print('File size: ' + str(round(size / 1024, 3)) + ' Kilobytes')
    elif unit == "MB":
        return print('File size: ' + str(round(size / (1024 * 1024), 3)) + ' Megabytes')
    else:
        return print('File size: ' + str(size) + ' bytes')

def get_file_size(file_path):
    size = os.path.getsize(file_path)
    return size

convert_bytes(get_file_size(TF_LITE_MODEL_FILE_NAME), "KB")
```

```
import tensorflow as tf
interpreter = tf.lite.Interpreter(model_path = TF_LITE_MODEL_FILE_NAME)
input_details = interpreter.get_input_details()
print(input_details)
output_details = interpreter.get_output_details()
print("Input Shape:", input_details[0]['shape'])
print("Input Type:", input_details[0]['dtype'])
print("Output Shape:", output_details[0]['shape'])
print("Output Type:", output_details[0]['dtype'])
```

Input Shape: [1 64 64 1]
Input Type: <class 'numpy.float32'>
Output Shape: [1 3]
Output Type: <class 'numpy.float32'>
0
30

```
interpreter.resize_tensor_input(input_details[0]['index'], (1,256,256,3))
interpreter.resize_tensor_input(output_details[0]['index'], (1,3))
interpreter.allocate_tensors()
input_details = interpreter.get_input_details()
output_details = interpreter.get_output_details()
print("Input Shape:", input_details[0]['shape'])
print("Input Type:", input_details[0]['dtype'])
print("Output Shape:", output_details[0]['shape'])
print("Output Type:", output_details[0]['dtype'])
```

Input Shape: [1 256 256   3]
Input Type: <class 'numpy.float32'>
Output Shape: [16 3]
Output Type: <class 'numpy.float32'>

```
TF_LITE_WEIGHTS_TEMP_FILE = "model_weights.h5"

path = r"C:\Users\USER\Desktop\Main Project\InsectsData\dataset\test\BlackPest\image0020.jpg"
img=load_img(path,target_size=(256,256))
i=img_to_array(img)/225
input_arr=np.array([i])
input_arr.shape
interpreter.set_tensor(input_details[0]['index'], input_arr)
interpreter.invoke()

output_data_tflite_opt = interpreter.get_tensor(output_details[0]['index'])
print(output_data_tflite_opt)

pred=np.argmax(output_data_tflite_opt)
print(pred)
plt.imshow(img)
if pred==0:
  print("blackpest")
elif pred==1:
  print("ladha")
```

```
elif pred==2:
  print("leaf")
```

5.2 Object Detection using Model Maker

```
#Google Colab NoteBook
! pip install -q tflite-model-maker-nightly
! pip install -q tflite-support-nightly

#Mount Google drive with colab notebook
from google.colab import drive
drive.mount('/content/drive')

import numpy as np
import os

from tflite_model_maker.config import ExportFormat, QuantizationConfig
from tflite_model_maker import model_spec
from tflite_model_maker import object_detector

from tflite_support import metadata

import tensorflow as tf
assert tf.__version__.startswith('2')

tf.get_logger().setLevel('ERROR')
from absl import logging
logging.set_verbosity(logging.ERROR)

!unzip -q /content/drive/MyDrive/opencv/testing.zip

# Load the dataset
train_data = object_detector.DataLoader.from_pascal_voc(
    'testing/train',
    'testing/train',
    ['BlackPest','Ladha','Leaf']
)

val_data = object_detector.DataLoader.from_pascal_voc(
    'testing/validate',
    'testing/validate',
    ['BlackPest','Ladha','Leaf']
)

#Select a model architecture
spec = model_spec.get('efficientdet_lite2')
```

```
#Train the TensorFlow model with the training data.
model = object_detector.create(train_data, model_spec=spec, batch_size=4, train_whole_mo
del=True, epochs=20, validation_data=val_data)

# Evaluate the model with the validation data.
model.evaluate(val_data)

#Export as a TensorFlow Lite model.
model.export(export_dir='.', tflite_filename='custom_object_detection.tflite')

#Evaluate the TensorFlow Lite model.
model.evaluate_tflite('custom_object_detection.tflite', val_data)


# Download the TFLite model to your local computer.
from google.colab import files
files.download('custom_object_detection.tflite')
```

5.3 Program for Raspberry pi

```
import time
import numpy as np
from PIL import Image
import tflite_runtime.interpreter as tflite
import os
import cv2

cap = cv2.VideoCapture(0)
if !cap.isOpened():
        twilio("Camera is not working")

edgetpu='0'

#====custom model and label files==================
model_dir = 'home/pi/custom_object_detection/models/custom'

model = 'custom_object_detection.tflite'


label = 'custom_labels.txt'

#============================================

model_path=os.path.join(model_dir,model)
```

```
label_path=os.path.join(model_dir,label)

#--------------------object detection----------------------------------------------------
def detect_objects(interpreter, image, score_threshold=0.3, top_k=6):
    """Returns list of detected objects."""
    set_input_tensor(interpreter, image)
    #interpreter.invoke()
    invoke_interpreter(interpreter)

    global model_dir
    if (model_dir=='models/pretrained'):
      # for pre-trained models
      boxes = get_output_tensor(interpreter, 0)
      class_ids = get_output_tensor(interpreter, 1)
      scores = get_output_tensor(interpreter, 2)
      count = int(get_output_tensor(interpreter, 3))
    else:
      # for custom models made by Model Maker
      scores = get_output_tensor(interpreter, 0)
      boxes = get_output_tensor(interpreter, 1)
      count = int(get_output_tensor(interpreter, 2))
      class_ids = get_output_tensor(interpreter, 3)


    def make(i):
        ymin, xmin, ymax, xmax = boxes[i]
        return Object(
            id=int(class_ids[i]),
            score=scores[i],
            bbox=BBox(xmin=np.maximum(0.0, xmin),
                      ymin=np.maximum(0.0, ymin),
                      xmax=np.minimum(1.0, xmax),
                      ymax=np.minimum(1.0, ymax)))

    return [make(i) for i in range(top_k) if scores[i] >= score_threshold]

import collections
Object = collections.namedtuple('Object', ['id', 'score', 'bbox'])

class BBox(collections.namedtuple('BBox', ['xmin', 'ymin', 'xmax', 'ymax'])):
    """Bounding box.
    Represents a rectangle which sides are either vertical or horizontal, parallel
    to the x or y axis.
    """
    __slots__ = ()
#-----------------------------------------------------------------------

#----------Loading Labels---------------------------------------------------
```

```
import re
def load_labels(path):
  """Loads the labels file. Supports files with or without index numbers."""

  with open(path, 'r', encoding='utf-8') as f:
    lines = f.readlines()
    labels = {}
    for row_number, content in enumerate(lines):
      pair = re.split(r'[:\s]+', content.strip(), maxsplit=1)
      if len(pair) == 2 and pair[0].strip().isdigit():
        labels[int(pair[0])] = pair[1].strip()
      else:
        labels[row_number] = pair[0].strip()
  return labels


#------------------------------------------------------------------------

#------Making Interpreter-----------------------------------------------------------
import platform

EDGETPU_SHARED_LIB = {
  'Linux': 'libedgetpu.so.1',
  'Darwin': 'libedgetpu.1.dylib',
  'Windows': 'edgetpu.dll'
}[platform.system()]

def make_interpreter(path, edgetpu):
    print (path,edgetpu)
    if(edgetpu=='0'):
        interpreter = tflite.Interpreter(model_path=path)
    else:
      path, *device = path.split('@')
      interpreter                                                                    =
tflite.Interpreter(model_path=path,experimental_delegates=[tflite.load_delegate(EDGETPU_
SHARED_LIB,{'device': device[0]} if device else {})])


    print('Loading Model: {} '.format(path))

    return interpreter

#------------------------------------------------------------------------

def input_image_size(interpreter):
    """Returns input image size as (width, height, channels) tuple."""
    _, height, width, channels = interpreter.get_input_details()[0]['shape']
    return width, height, channels
```

```
def set_input_tensor(interpreter, image):
  """Sets the input tensor."""
  image = image.resize((input_image_size(interpreter)[0:2]), resample=Image.NEAREST)
  #input_tensor(interpreter)[:, :] = image

  tensor_index = interpreter.get_input_details()[0]['index']
  input_tensor = interpreter.tensor(tensor_index)()[0]
  input_tensor[:, :] = image


def get_output_tensor(interpreter, index):
  """Returns the output tensor at the given index."""
  output_details = interpreter.get_output_details()[index]
  #print(output_details)
  tensor = np.squeeze(interpreter.get_tensor(output_details['index']))
  return tensor

def invoke_interpreter(interpreter):
  global inference_time_ms

  t1=time.time()
  interpreter.invoke()
  inference_time_ms = (time.time() - t1) * 1000
  print("****Inference time = ", inference_time_ms)

#------------------------------------------------------------------------
#--------Image Overlay-------------------------------------------------------

def overlay_text_detection(objs, labels, cv2_im, fps):
    height, width, channels = cv2_im.shape
    font=cv2.FONT_HERSHEY_SIMPLEX

    for obj in objs:
        x0, y0, x1, y1 = list(obj.bbox)
        x0, y0, x1, y1 = int(x0*width), int(y0*height), int(x1*width), int(y1*height)
        percent = int(100 * obj.score)

        if (percent>=50):
            box_color, text_color, thickness=(0,255,0), (0,0,0),2
        else:
            continue


        text3 = '{}% {}'.format(percent, labels.get(obj.id, obj.id))
        print(text3)

        try:
```

```
            cv2_im = cv2.rectangle(cv2_im, (x0, y0), (x1, y1), box_color, thickness)
            cv2_im = cv2.rectangle(cv2_im, (x0,y1-10), (x1, y1+10), (255,255,255), -1)
            cv2_im = cv2.putText(cv2_im, text3, (x0, y1),font, 0.6, text_color, thickness)
        except:
            #log_error()
            pass

    global model, inference_time_ms
    str1="FPS: " + str(fps)
    cv2_im = cv2.putText(cv2_im, str1, (width-180, height-55),font, 0.7, (255, 0, 0), 2)

    str2="Inference: " + str(round(inference_time_ms,1)) + " ms"
    cv2_im = cv2.putText(cv2_im, str2, (width-240, height-25),font, 0.7, (255, 0, 0), 2)

    cv2_im = cv2.rectangle(cv2_im, (0,height-20), (width, height), (0,0,0), -1)
    cv2_im = cv2.putText(cv2_im, model, (10, height-5),font, 0.6, (0, 255, 0), 2)
    try:
        return cv2_im,text3
    except:
        return cv2_im
#-------------------------------------------------------------------------


def main():

  interpreter = make_interpreter(model_path, edgetpu)

  interpreter.allocate_tensors()

  labels = load_labels(label_path)

  fps=1

  while True:

        start_time=time.time()

        ret, frame = cap.read()
        if not ret:
            break

        cv2_im = frame
        #cv2_im = cv2.flip(cv2_im, 0) #vertical flip
        #cv2_im = cv2.flip(cv2_im, 1) #horzontal flip
        output=frame
        cv2_im_rgb = cv2.cvtColor(cv2_im, cv2.COLOR_BGR2RGB)
        image = Image.fromarray(cv2_im_rgb)
```

```
        results = detect_objects(interpreter, image)
        cv2_im = overlay_text_detection(results, labels, cv2_im, fps)

        try:
            if cv2.waitKey(1) & 0xFF == ord('q'):
                break
        except KeyboardInterrupt:
                break
        if len(cv2_im)==2:
            cv2.imwrite('/home/pi/Desktop/image.jpg',output)
            break

        elapsed_ms = (time.time() - start_time) * 1000
        fps=round(1000/elapsed_ms,1)
        print("--------fps: ",fps,"----------------")
def Ladha():
            twilio("content about ladha")
def BlackPest():
            twilio("content about blackpest")
def Leaf():
            twilio("Content about Leaf")
def twilio(res):
            #twilio code

            from twilio.rest import Client
            account_sid="AC5a23d4ffddb140ca9e022104ffe99840"
            auth_token="2e0529811f508e1c21cdf77f9f40bd2f"
            client=Client(account_sid,auth_token)
            message=client.api.account.messages.create(
            to="+919390633141",
            from_="+19252701563",
            body=res)

if __name__ == '__main__':
            main()
            #print("picture captured")
            from keras_preprocessing.image import ImageDataGenerator,load_img,img_to_array
            import numpy as np
            import tflite_runtime.interpreter as tflite
            path = r"/home/pi/Desktop/image.jpg"
            interpreter=tflite.Interpreter(model_path=r"tf_lite_model.tflite")
            input_details = interpreter.get_input_details()
            output_details = interpreter.get_output_details()
            #print(input_details)
            interpreter.resize_tensor_input(input_details[0]['index'], (1,256,256,3))
            interpreter.resize_tensor_input(output_details[0]['index'], (1,4))
            interpreter.allocate_tensors()
```

```
img=load_img(path,target_size=(256,256))
i=img_to_array(img)/225
input_arr=np.array([i])
input_arr.shape

interpreter.set_tensor(input_details[0]['index'], input_arr)
interpreter.invoke()
out= interpreter.get_tensor(output_details[0]['index'])
print(out)
pred=np.argmax(out)
c=0
for i in out[0]:
        if i>=0.999:
        c+=1
if out[0][0]>=0.999 and c==1:
        res="BlackPest"
elif out[0][1]>=0.999 and c==1:
        res="ladha"
elif out[0][2]>=0.999 and c==1:
        res="leaf"
else:
        res="unknown"
print("Class detected")
twilio(res)
```

6.SYSTEM TESTING

6.1 Overview of Testing

System testing is a critical part of the development process for the pest detection system. It is a process of evaluating the system's functionality and performance against the specified requirements. The objective of system testing is to ensure that the system meets the customer's needs and performs as expected. The testing process involves various types of tests to ensure that the system is free from bugs and is reliable.

6.2 Types of Tests

6.2.1 Functional testing

This type of testing ensures that the pest detection system's functions work as intended. The system is tested against its functional requirements to verify that all functions work correctly. For example, testing the system to identify pests accurately and provide recommendations for pesticide treatment.

6.2.2 Performance testing

Performance testing is used to evaluate the system's ability to handle a specific workload. It measures the system's response time, speed, and scalability. In the case of the pest detection system, performance testing would involve measuring the time taken to process an image and provide a recommendation for pesticide treatment.

6.2.3 Usability testing

Usability testing ensures that the pest detection system is user-friendly and meets the user's needs. This testing is typically done by users who are not familiar with the system. The goal is to identify any issues with the system's design and navigation.

6.2.4 Integration testing

Integration testing is used to test how the pest detection system interacts with other systems or components. This testing ensures that the system works as expected when integrated with other systems, such as a database or a camera.

In conclusion, the pest detection system needs to undergo various types of system testing to ensure that it works as expected, meets the customer's needs, and is reliable. Functional testing, performance testing, usability testing, integration testing, and security testing are all critical parts of the system testing process.

7.RESULTS

Table 3: Epochs and Accuracy of CNN Classifier

Epochs	Training Accuracy	Testing Accuracy
9	99	88
13	99	90
15	100	91

From the table provided, we can see that the insect classification model was trained for different numbers of epochs, with the training and testing accuracies recorded for each training period and training accuracy values range from 0.9977 to 0.9999, while the testing accuracy values range from 0.8873 to 0.9172.

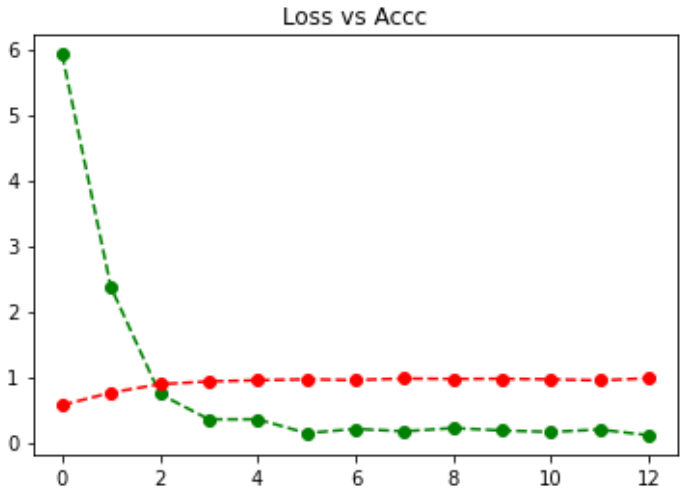

Fig 14: Graph: Loss Vs Accuracy

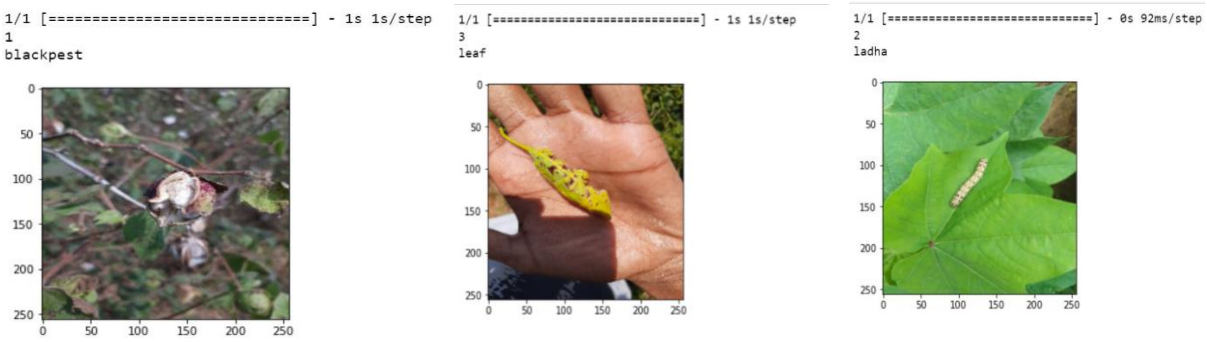

Fig 15: Results of CNN Classifier

Object Detection Evaluation Results

200/200 [==============================] - 1916s 10s/step

{'AP': 0.5616082,
'AP50': 0.97831106,
'AP75': 0.6023003,
'APs': -1.0,
'APm': 0.48134318,
'APl': 0.565245,
'ARmax1': 0.61262286,
'ARmax10': 0.6407454,
'ARmax100': 0.64136267,
'ARs': -1.0,
'ARm': 0.55,
'ARl': 0.6461,
'AP_/BlackPest': 0.63547516,
'AP_/Ladha': 0.50907195,
'AP_/Leaf': 0.5402775}

Fig 16: Device Setup in a box

Fig 17: Field Work at Ananthavaram,Krishna Dist

Fig 18: Device Setup in Framers Field

8.CONCLUSION

Our project provides farmers with a cost-effective and efficient solution for pest management, helping to improve crop yields and increase their profits. By analyzing a dataset of 1503 photos of Ladha, Black Pest, and damaged leaves from cotton and chili fields in Andhra Pradesh and Telangana, we were able to use Inception v3 and object detection to identify the presence of pests in crops. With no human intervention required, our system can detect potential pest infestations early on and provide farmers with suggestions for appropriate pesticides to use to protect their crops and prevent further damage. However, our work is not done yet. In the future, we plan to expand the scope of our project by collecting more data sets, including different insects and pests classes from different crops, and increasing the dataset size for better accuracy. We also aim to develop features that can detect multiple insects or pests in a single image and even detect partial images of insects, which would be a significant contribution to pest management research.

9.REFERENCES

[1] Thenmozhi Kasinathan, Dakshayani Singaraju, Srinivasulu Reddy Uyyala;Insect classifification and detection in field crops using modern machine learning techniques.(2020)

[2] Insect Detection and Classifification Based on an Improved Convolutional Neural Network. Denan Xia, Peng Chen, Bing Wang, Jun Zhang,and Chengjun Xie.(2018)

[3] Remote Insects Trap Monitoring System Using Deep Learning Framework and IoT. Balakrishnan Ramalingam, Rajesh Elara Mohan, Sathian Pookkuttath, Charan Satya Chandra Sairam Borusu.(2020)

[4] Insect Pest Detection and Identification Method Based on Deep Learning for Realizing a Pest Control System.Hiroaki Kuzuhara; Hironori Takimoto; Yasuhiro Sato; Akihiro Kanagawa.(2020)

[5] Nanni L, Maguolo G, Pancino F. Research on insect pest image detection and recognition based on bio inspired methods.(2019)

[6] Sharma P, Berwal YP, Ghai W. Performance analysis of deep learning CNN models for disease detection in plants using image segmentation.(2019)

[7] A large-scale benchmark dataset for insect pest recognition;2019. https://github.com/xpwu95/IP102.

[8] Shen Y, Zhou H, Li J, Jian F, Jayas DS; Detection of stored-grain insects using deep learning. (2018)

[9] Bhargava A, Bansal A. Fruits and vegetables quality evaluation using computer vision: A review. J King Saud Univ Comput Inf Sci (2018).

[10] Deng L, Wang Z, Wang C, He Y, Huang T, Dong Y, et al. Application of agricultural insect pest detection and control map based on image processing analysis. J Intell Fuzzy Syst (2020).

[11] Liu J-e, An F-P. Image classification algorithm based on deep learning-kernel function.Sci Program.(2020)
[12] A large-scale benchmark dataset for insect pest recognition; 2019. Link: https://github.com/xpwu95/IP102.

[13] Nanni L, Maguolo G, Pancino F. Insect pest image detection and recognition based on bio-inspired methods. Ecol Inform. 2020;101089.

[14] Rother C, Kolmogorov V, Blake A. ‘‘GrabCut” interactive foreground extraction using iterated graph cuts. ACM T Graphic 2004;23:309–14.

[15] Deng L, Wang Y, Han Z, Yu R. Research on insect pest image detection and recognition based on bio inspired methods. Biosyst Eng 2018;169:139–48