SASTRA
ENGINEERING · MANAGEMENT · LAW · SCIENCES · HUMANITIES · EDUCATION
DEEMED TO BE UNIVERSITY
(U/S 3 of the UGC Act, 1956)
THINK MERIT | THINK TRANSPARENCY | THINK SASTRA
THANJAVUR | KUMBAKONAM | CHENNAI

Deep Learning-Driven Pest Detection and Classification with Instant SMS Alerts for Precision Agriculture

GUIDE

Mr. Ramesh R
Srinivasa Ramanujan Centre
SASTRA Deemed to be University

Presented By:

Ch. Hemanth Sai Nag – 226003030

D. Deepak Reddy – 226003039

G. Sai Aasish – 226003050

Contents

BASE PAPER DETAILS

Title : Pest classification: Explainable few-shot learning vs. convolutional neural networks vs. transfer learning.

Journal Name : SCI-E

Published Year : 2025

ABSTRACT

- Detecting pests in agriculture under minimal labelled data and real-time constraints.
- When the datasets are small, FSL will be applied for pest detection. For large datasets, Vision Transformers (ViT).
- Graph Neural Networks (GNNs) will convert images into graphs and train the model.
- Self-Supervised Learning will fine-tune performance with small datasets.
- TensorFlow Lite will allow for optimized mobile inference and allow lightweight yet high performance pest detection.
- OpenCV will process real time video feeds, allowing immediate analysis.
- Integration with SMS notification is to ensure prompt action as this will notify the farmer about the activity of the pest.
- The application combines deep learning with real-time processing, providing a very efficient and accurate method toward field-ready management of pests.

PROBLEM STATEMENT

- Manual pest detection is slow, error-prone, and relies on expert knowledge, while existing AI models require large labeled datasets.
- Conventional deep learning models struggle with accuracy in data-scarce environments and lack interpretability, making them difficult to trust.
- Therefore, leverages Explainable Few-Shot Learning (FSL) to enhance pest classification accuracy, reduce data dependency, and provide transparent, trustworthy AI decisions.

OBJECTIVE

- Develop and evaluate an Explainable Few-Shot Learning (FSL)-based pest classification system to improve accuracy, efficiency, and transparency in smart agriculture.
- Compare FSL models with traditional deep learning approaches to achieve high classification accuracy with minimal data while integrating explainability techniques like Grad-CAM for better interpretability and trust.

LITERATURE REVIEW

Title	Author	Published Year	Methodology	Journal	Merits	Demerits
1. Detection of mulberry ripeness stages using deep learning models	Seyed-Hassan Miraei Ashtiani	2021	The study employs deep learning models, specifically Convolutional Neural Networks (CNNs), to classify mulberry ripeness stages based on image data.	IEEE	The proposed CNN-based approach achieves high accuracy in classifying mulberry ripeness stages, demonstrating the effectiveness of deep learning in agricultural applications.	• The study may face challenges related to the generalization of the model to different environmental conditions and the need for a large dataset to train the deep learning model effectively

Title	Author	Published Year	Methodology	Journal	Merits	Demerits
Machine Learning based Pest Identification in Paddy Plants.	Vivek Agnihotri	2019	The authors propose an OTA-C (Operational Transconductance	IEEE	The proposed design offers a flexible approach to implementing fractional-order filters.	The practical implementation of fractional-order capacitors can be challenging.
AMN: Attention Metric Network for One-Shot Remote Sensing Image Scene Classification.	Yonghao Xu	2020	The authors propose the Attention Metric Network (AMN), a deep learning model.	Remote Sensing	The AMN model enhances feature representation by focusing on the most relevant parts of an image	The study may face challenges in generalizing the model to diverse remote sensing datasets due to potential overfitting.

Title	Author	Published Year	Methodology	Journal	Merits	Demerits
Multi-Domain Few-Shot Learning and Dataset for Agricultural Applications	Sai Vidyaranya Nuthalapati	2021	Introduces a group-aware contrastive network using context graphs.	IEEE	One merit of this method could be its innovative approach to person re-identification using group-aware contrastive networks.	A potential demerit could be the complexity of implementing and fine-tuning such a specialized network
Insect Pest Detection and Identification Method Based on Deep Learning for Realizing a Pest Control System	Hiroaki Kuzuhara	2020	The authors propose a two-stage detection and identification method for small insect pests utilizing Convolutional Neural Networks (CNNs).	SCI-E	A significant advantage of this method is its ability to accurately detect and identify small insect pests, which are often challenging to recognize.	A potential drawback is that the two-stage process may result in increased computational complexity and longer processing times.

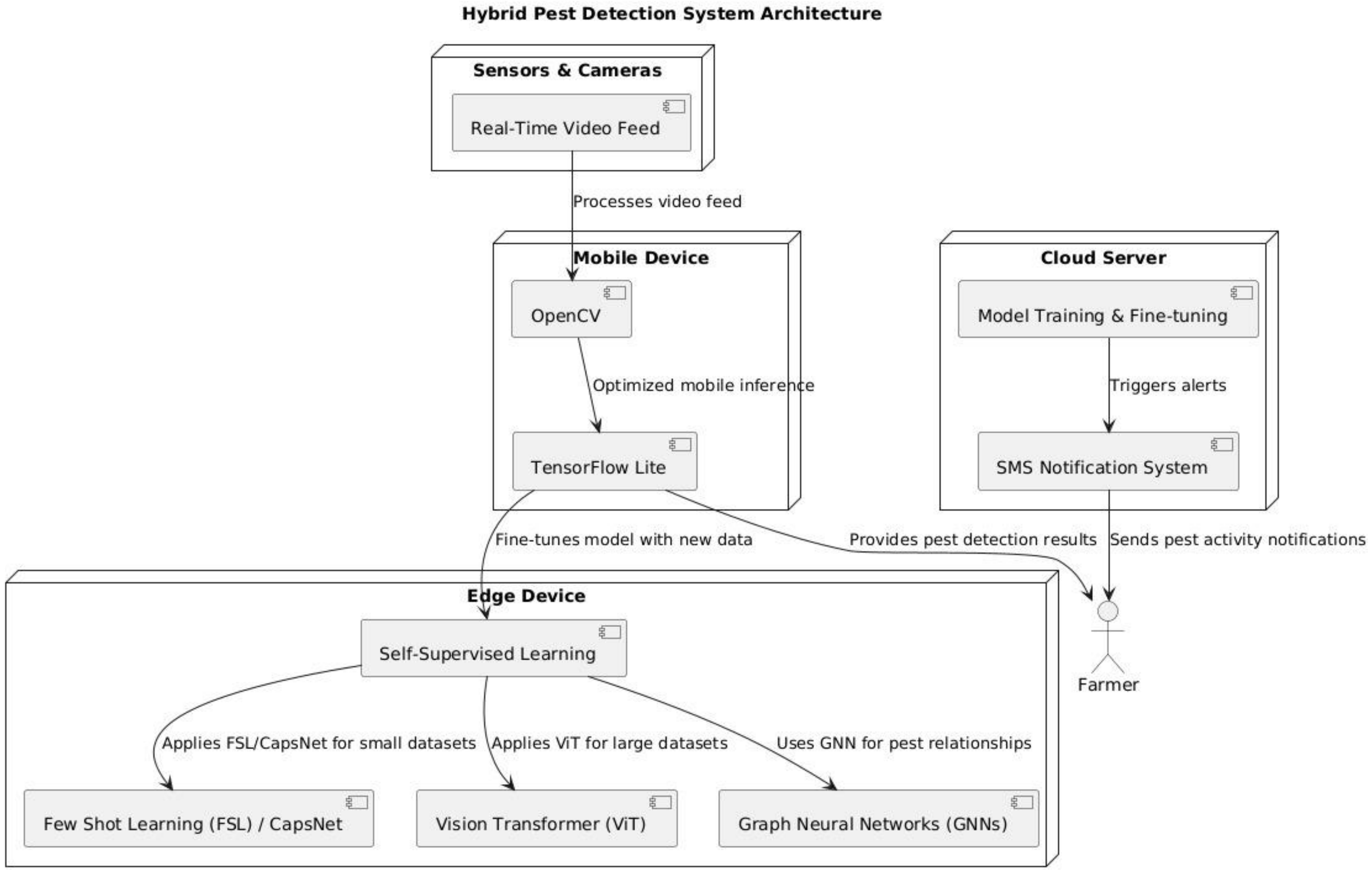
Title	Author	Published Year	Methodology	Journal	Merits	Demerits
Few-shot learning for image-based bridge damage detection	Yan Gao	2023	Few Shot learning	Elsevier	Train model with few number of images	It’s not sure we can get more accuracy than general deep learning model.
Crop pest classification based on deep convolutional neural network and transfer learning	K. Thenmozhi	2019	Convolution Neural Network(CNN)	Elsevier	Training is very easy to implement.	We can’t get best efficiency by this model

Title	Author	Published Year	Methodology	Journal	Merits	Demerits
Vision transformer-based visual language understanding of the construction process	Bin Yang	2024	Vision transformer	Elsevier	Vision Transformers apply this architecture to image data, treating the image as a sequence of patches instead of a grid of pixels.	They still underperform CNNs when training data is very limited. Without sufficient data, the attention model tends to overfit.

Work Plan

Timeline/ Process	1-10 days	10-20 days	20-30 days	30-45 days	45-60 days
Data Collection					
Data Pre-processing					
Split the data					
Train FSL, GNN, Vit					
Train SSL and combine model					
OpenCV					
Final Output					

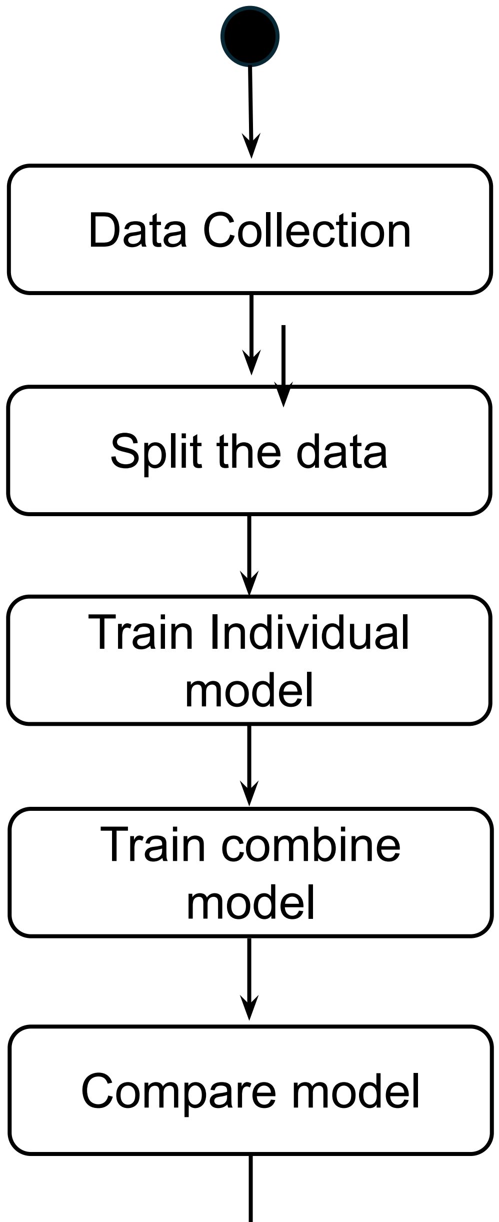
ARCHITECTURE DIAGRAM

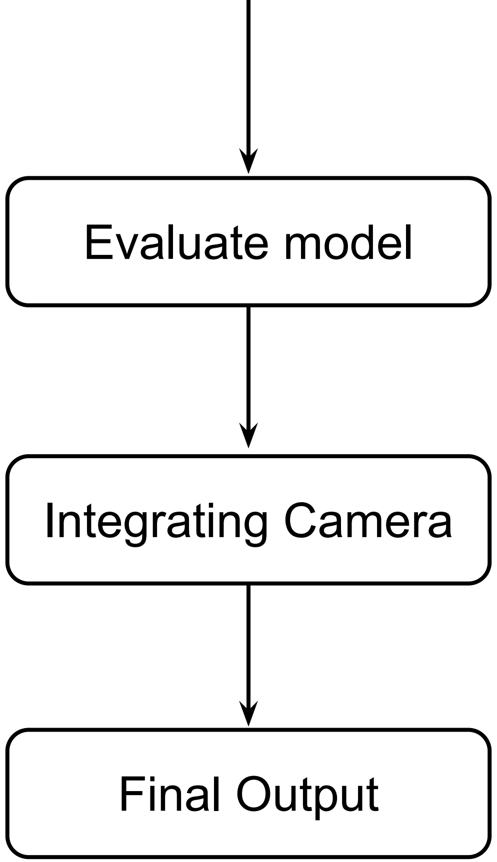


Dataset Description

```
pest/
 |── train/
        |-aphids
        |- armyworm
        |- beetle
        |- bollworm
        |- grasshopper
        |- mites
        |- mosquito
        |- sawfly
        |- stem borer
 |── test/
        |-aphids
        |- armyworm
        |- beetle
        |- bollworm
        |- grasshopper
        |- mites
        |- mosquito
        |- sawfly
        |- stem borer
```

WORK FLOW DIAGRAM

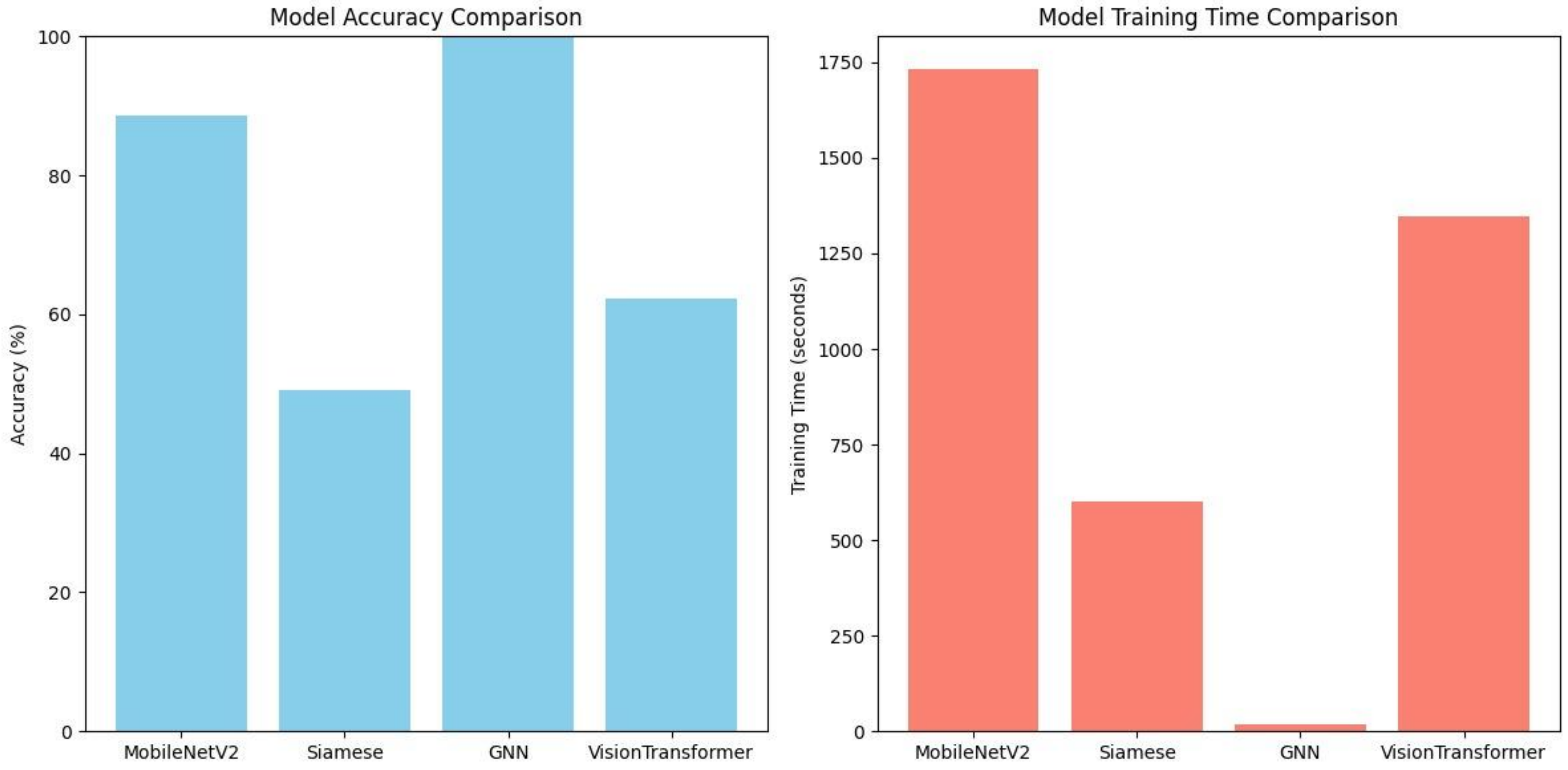



Evaluate model
Integrating Camera
Final Output

FULL IMPLEMENTATION

```python
# Define the necessary global variables if not already defined
IMG_SIZE = (224, 224)
BATCH_SIZE = 32
train_dir = 'Dataset/pest/train'
test_dir = 'Dataset/pest/test'
```

Python



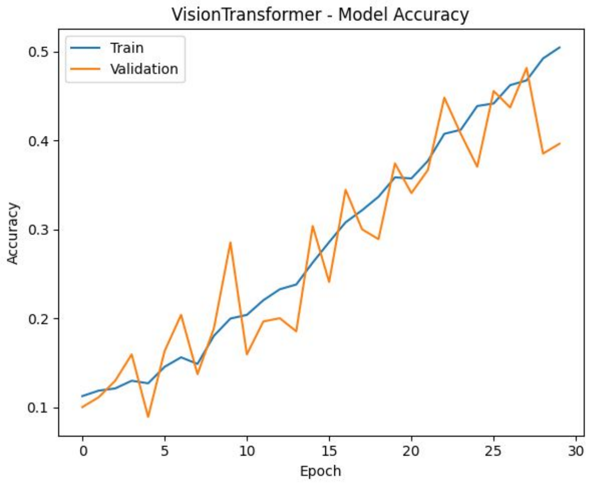
```python
# Common data loading for TensorFlow models
def load_tf_data(shuffle=True):
    train_datagen = tf.keras.preprocessing.image.ImageDataGenerator(
        zca_epsilon=1e-06,
        rotation_range=30,
        width_shift_range=0.1,
        height_shift_range=0.2,
        shear_range=20,
        zoom_range=0.8,
        fill_mode="nearest",
        horizontal_flip=True,
        vertical_flip=True,
        validation_split=0.1,
        rescale=1./255
    )

    test_datagen = tf.keras.preprocessing.image.ImageDataGenerator(rescale=1./255)

    training = train_datagen.flow_from_directory(
        train_dir,
        batch_size=BATCH_SIZE,
        target_size=IMG_SIZE,
        subset="training",
        shuffle=shuffle
    )

    validing = train_datagen.flow_from_directory(
        train_dir,
        batch_size=BATCH_SIZE,
        target_size=IMG_SIZE,
        subset='validation',
        shuffle=shuffle
    )

    testing = test_datagen.flow_from_directory(
        test_dir,
        batch_size=BATCH_SIZE,
        target_size=IMG_SIZE,
        shuffle=shuffle
    )

    num_classes = len(training.class_indices)
    class_labels = list(training.class_indices.keys())

    return training, validing, testing, num_classes, class_labels
```

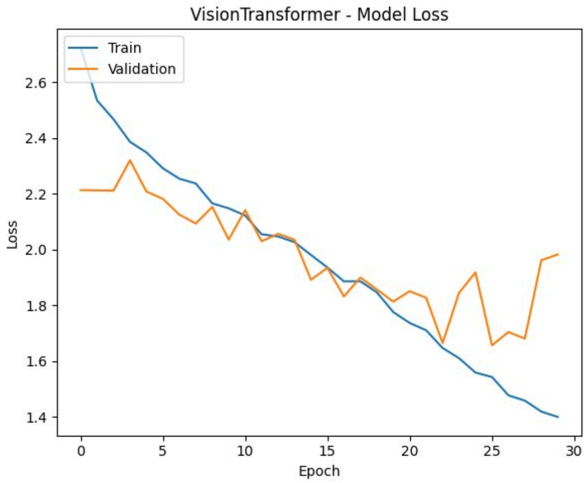
```python
def main():
    # Load data
    training, validing, testing, num_classes, class_labels = load_tf_data()
    print(f"Number of classes: {num_classes}")
    print(f"Class labels: {class_labels}")
    # Dictionary to store results
    results = {}
    # Train MobileNetV2 model
    mobilenetv2_model, mobilenetv2_acc, mobilenetv2_time = train_mobilenetv2(training, validing, testing, num_classes)
    results["MobileNetV2"] = {"accuracy": mobilenetv2_acc, "training_time": mobilenetv2_time}
    # Visualize prediction for a sample image
    sample_img = 'Dataset/pest/test/beetle/jpg_33.jpg'
    if os.path.exists(sample_img):
        visualize_prediction(mobilenetv2_model, sample_img, class_labels, "MobileNetV2")
    # Train Siamese Network
    # Note: You may need to update this function to accept num_classes if needed
    siamese_model, siamese_acc, siamese_time = train_siamese_network()
    results["Siamese"] = {"accuracy": siamese_acc, "training_time": siamese_time}
    # Train GNN model if possible
    try:
        # Note: You may need to update this function to accept num_classes if needed
        gnn_model, gnn_acc, gnn_time = train_gnn_model()
        if gnn_model is not None:
            results["GNN"] = {"accuracy": gnn_acc, "training_time": gnn_time}
    except Exception as e:
        print(f"Could not train GNN model: {e}")
    # Train Vision Transformer model
    vit_model, vit_acc, vit_time = train_vit_model(training, validing, testing, num_classes)
    results["VisionTransformer"] = {"accuracy": vit_acc, "training_time": vit_time}
    if os.path.exists(sample_img):
        visualize_prediction(vit_model, sample_img, class_labels, "VisionTransformer")
    # Compare model performances
    print("\n----- Model Performance Comparison -----")
    for model_name, metrics in results.items():
        print(f"{model_name}: Accuracy = {metrics['accuracy'] * 100:.2f}%, Training Time = {metrics['training_time']:.2f} seconds")
    # Plot comparison chart
    plt.figure(figsize=(12, 6))
    # Accuracy comparison
    plt.subplot(1, 2, 1)
    model_names = list(results.keys())
    accuracies = [results[model]["accuracy"] * 100 for model in model_names]
    plt.bar(model_names, accuracies, color='skyblue')
    plt.ylabel('Accuracy (%)')
    plt.title('Model Accuracy Comparison')
    plt.ylim([0, 100])
    # Training time comparison
    plt.subplot(1, 2, 2)
    training_times = [results[model]["training_time"] for model in model_names]
    plt.bar(model_names, training_times, color='salmon')
    plt.ylabel('Training Time (seconds)')
    plt.title('Model Training Time Comparison')
    plt.tight_layout()
    plt.savefig('model_comparison.png')
    plt.close()
    print("Comparison chart saved as 'model_comparison.png'")
```

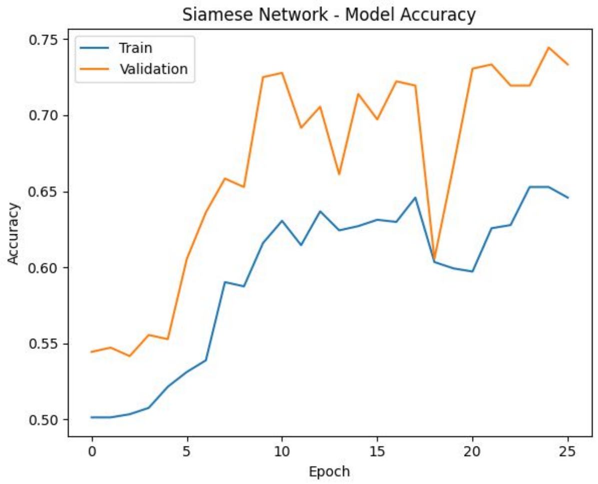
MODULES USED

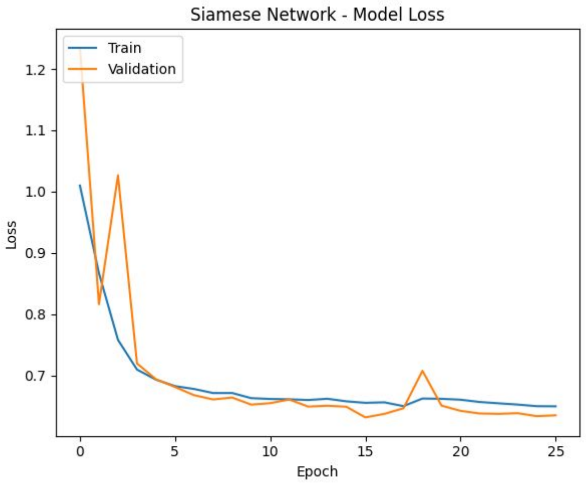
- Load TensorFlow data
- Build mobilenetv2 model, train mobilenetv2 model
- Euclidean distance, build Siamese base, build Siamese model,
- Create pair, load image from directory.
- Prepare pair of models, classify with Siamese, train Siamese network
- Train GNN model.
- Vit model.
- Main module
- OpenCV for using camera for classifying pest
- Twillo for sending messags

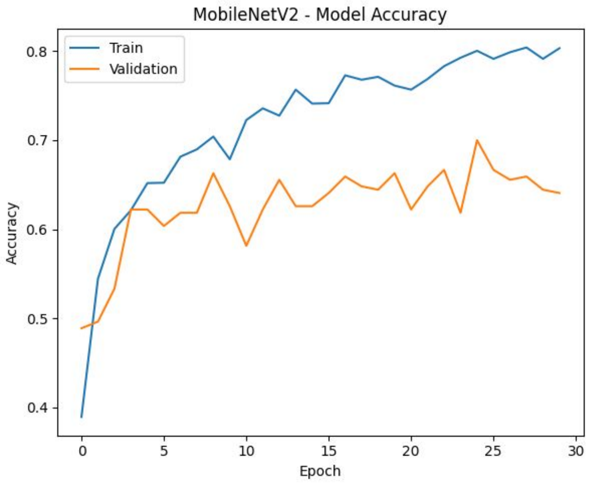
Results


VisionTransformer - Model Accuracy
Train
Validation
Accuracy
Epoch


VisionTransformer - Model Loss
Train
Validation
Loss
Epoch


Siamese Network - Model Accuracy
Train
Validation
Accuracy
Epoch


Siamese Network - Model Loss
Train
Validation
Loss
Epoch


MobileNetV2 - Model Accuracy
Train
Validation
Accuracy
Epoch

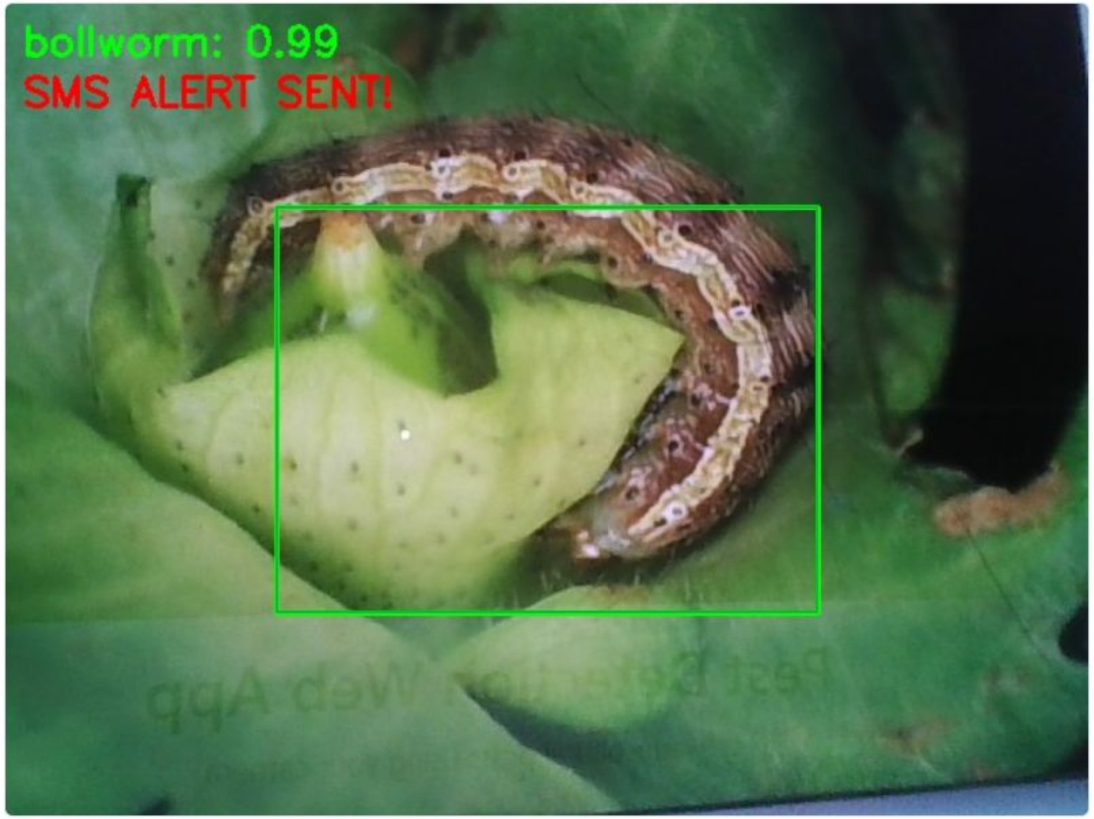

MobileNetV2 - Model Loss
Train
Validation
Loss
Epoch

Pest Detection Web App

Identify agricultural pests using your camera

Ready

Camera View



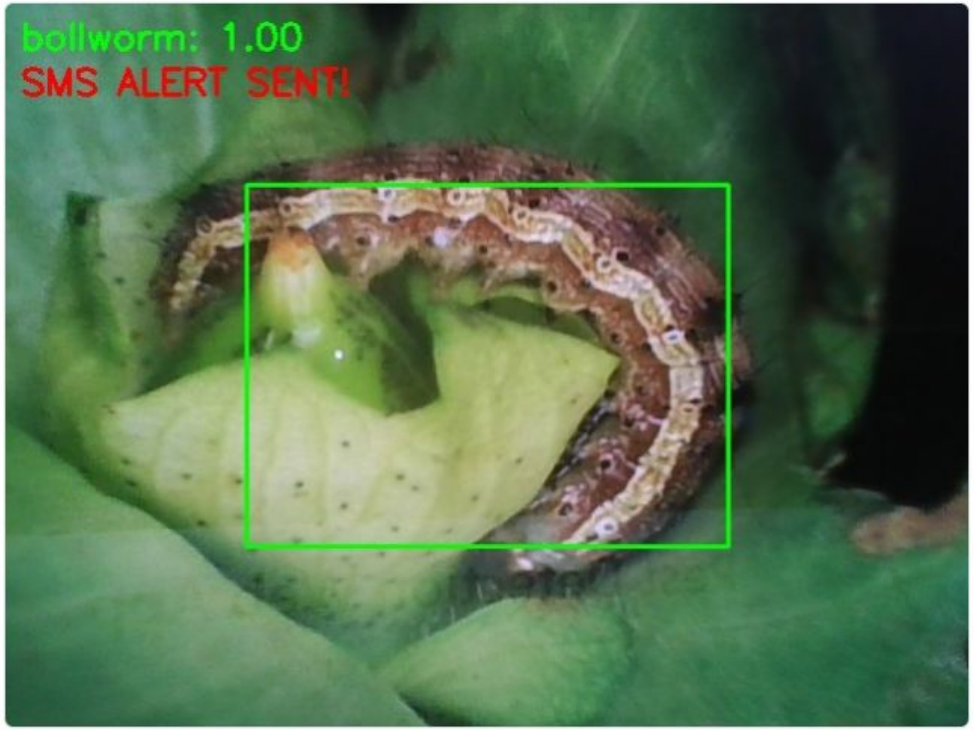
Capture & Analyze

Instructions

1. Allow camera access when prompted
2. Position the pest inside the green box
3. Click "Capture & Analyze" for detailed results
4. Review the pest identification results below

Detection Results

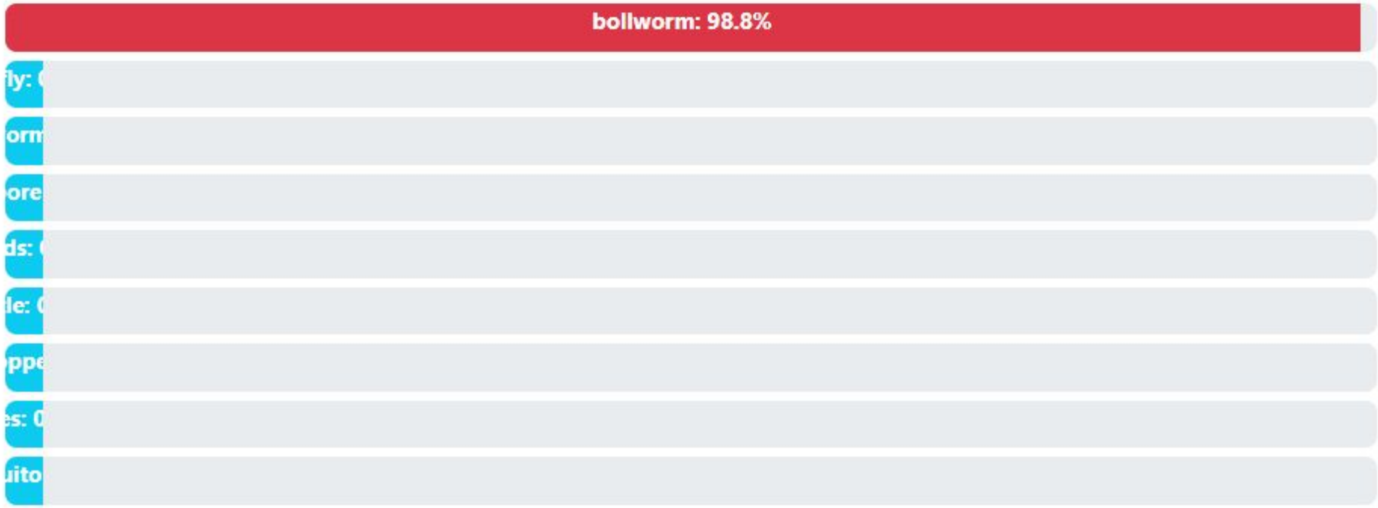
Captured Image



Detected Pest: bollworm

Confidence: 98.8%

All Class Probabilities



Conclusion

Implementing Few-Shot Learning (FSL) under limited computational resources often leads to reduced accuracy due to its reliance on complex training dynamics and meta-learning strategies. In contrast, **Self-Supervised Learning (SSL)** proves to be more effective in such scenarios. When combined with lightweight architectures like **MobileNetV2**, SSL enables the model to learn meaningful patterns from unlabeled data and adapt quickly with minimal labels — all while running efficiently on devices with low CPU/GPU capabilities.

This approach not only improves detection performance but also supports **real-time, on-device pest monitoring**, making it a more practical alternative to traditional deep learning methods in agriculture. Therefore, the integration of SSL with MobileNetV2 provides a cost-effective and scalable solution for pest detection in the field.

THANK YOU!!