IJCRT.ORG ISSN : 2320-2882

INTERNATIONAL JOURNAL OF CREATIVE RESEARCH THOUGHTS (IJCRT)

An International Open Access, Peer-reviewed, Refereed Journal

Timely Detection Of Stem Borer Pest Infestation Through Convolutional Neural Network

Sofiya Rani N, Yashmin Unnisha M K, Ms.P.Dhanya
Student, Student, Assistant Professor

Computer Science And Engineering
Aalim Muhammed Salegh College Of Engineering, Avadi,Chennai.

Abstract: The Yellow Stem Borer (YSB), Scirpophaga incertulas (Walker), is an important pest of rice throughout tropical South and Southeast Asia. The highest incidence of this pestis primarily observed in tropical lowland rice fields and deep-water rice cultivation. Theyield loss caused by the YSB is estimated to be 20% in early-planted rice crops and 80%in late-planted crops. In this paper, we developed a method to detect and classify the forms of YSB using a Convolutional Neural Network (CNN) and then model the infestation migration patterns of YSB in several rice-growing regions by using a CNN learning model. A dedicated CNN architecture is designed, and optimized for its ability to extract features and discern spatial hierarchies indicative of pest presence. Transfer learning techniques, utilizing pre-trained models, enhance the model's capability to recognize subtle patterns associated with pest infestations. The dataset is carefully annotated and augmented to ensure robust model training, with an emphasis on realworld variability. These models can help detect, classify, and model the infestations of other Agricultural pests, improving food security for rice.

Keywords : Yellow Stem Borer, Precision agriculture tools, Pest Detection,
Food Security.

I. Introduction

Rice, one of the most important cereal crops worldwide, is a crucial element in ensuring global agriculture and food security. The advancements in rice breeding and cultivation techniques have led to the development of more water-efficient varieties, such as drought-resistant and flood-tolerant strains. There is substantial scope to increase current rice yields as farmers around the world continue to adopt innovative agricultural practices and leverage technological advancements. Rice, being a vital staple food for a significant portion of the global population, faces increasing demands due to population growth and changing dietary preferences. To meet these demands sustainably, farmers are increasingly turning to modern farming techniques, such as precision agriculture, where data-driven decisions help optimize the use of resources, including water and fertilizers. The yellow stem borer, a common rice crop pest, can inflict significant harm during various growth stages. During the vegetative stage, the central shoot of the rice plant is highly susceptible, often leading to a "dead heart" condition as the larvae bore into the stem, causing it to dry up and turn yellow. When rice plants enter the ear-bearing stage, the yellow stem borer continues to wreak havoc, resulting in a "white ear-head." Additionally, the yellow stem borer's damage can extend to the lower regions of the plant, affecting the panicle. Young rice plants are the preferred targets of the yellow stem borer, underscoring the importance of effective pest management to mitigate yield losses and maintain crop health.

1.1 YSB DETECTION

The methodology for YSB detection via Convolutional Neural Network (CNN) follows a systematic approach. Initially, a diverse dataset is compiled, encompassing images of both healthy crops and those affected by yellow stem borer. This dataset undergoes meticulous preprocessing, including resizing and normalization, with data augmentation techniques such as rotation and flipping employed to enhance model generalization. The CNN architecture is then crafted, incorporating convolutional layers for feature extraction and fully connected layers for classification. Training involves partitioning the dataset into training and validation sets, with the model adjusting weights via backpropagation using a selected loss function. Hyperparameters are fine-tuned to optimize performance, and the model is evaluated using metrics like accuracy and F1 score on a separate test dataset. Further refinement may be conducted based on evaluation results. A deployment interface is developed for user-friendly access, and the model's effectiveness is validated in real-world scenarios. Comprehensive documentation of the methodology, including dataset specifics and model architecture, is essential, along with clear communication of the model's capabilities to stakeholders for practical implementation in agriculture.

1.2 SCOPE OF PROJECT

The project centers on employing Convolutional Neural Networks (CNN) for the identification of yellow stem borer in crops. Beginning with the assembly of a diverse dataset containing images of both healthy and infested crops, meticulous attention is given to data preprocessing to ensure consistency. The subsequent phase entails devising and executing a CNN architecture specifically tailored for image classification, taking into account factors like depth, layers, and filters. Upon training the model on the curated dataset, its performance undergoes thorough evaluation using metrics such as accuracy, precision, recall, and F1 score on a separate test dataset. Parameter fine-tuning follows suit to optimize the model, thereby enhancing its accuracy and ability to generalize. Practical deployment is facilitated by developing a user-friendly interface or application to enhance accessibility for stakeholders. The model's validation in real-world scenarios ensures its efficacy across diverse conditions. Comprehensive documentation and clear communication of findings to stakeholders, including farmers and researchers, are integral facets of this endeavor.

LITERATURE SURVEY

[1] Y. Fan, T. Wang, Z. Qiu, J. Peng, C. Zhang and Y. He, "Fast detection of striped stem-borer

(Chilo suppressalis Walker) infested rice seedling based on visible/near-infrared hyperspectral imaging system", Sensors, Vol. 17, No. 11, p.2470, 2017.

Fan et al. (2019) suggested using a Back Propagation Neural Network (BPNN) to create models for infection evaluation based on textural traits, distinctive wavelengths, and whole spectra. Global BPNN models that include distinctive wavelengths and texture features yield optimal results, surpassing 95% in calibration and prediction classification accuracy. In order to identify Striped Stem Borer (SSB) colonization early on and assess the level of infection, our work shows that hyperspectral imaging approaches are feasible. This work's primary benefit lies in its ability to address challenges such as herbicide tolerance, insect resistance, drought tolerance, stress resilience, enhanced quality, and increased yield. However, the cost and the maintenance of the setup are high and have a limited detection range [1].

[2] R. Z. Ahmed and R. C. Biradar, "Energy-aware routing in WSN for pest detection in coffee plantation", In 2016 International Conference on Advances in Computing, Communications and Informatics (ICACCI) pp. 398-403. IEEE, (2016)

Ahmed and Biradar (2016) presented a novel idea for halting the development of the disease Coffee White Stem Borer (CWSB) in coffee plantation fields: Wireless Sensor Networks (WSNs). WSN installed in the coffee field monitors every coffee stem. plants, and it sends such information if it detects CWSB. The authors developed a pest detection system, Early Remote Warning Pest Detection (ERWPD) to detect the presence of CWSB in Coffee Arabica plants by means of the sink node's Cluster-Heads (CHs) are nodal centers. The CH compiles the information obtained from the main nodes and transmits it by creating paths to the sink [2].

[3] S. Bhandarkar, R. Prasad, V. Agarwal, R. Hebbar, D. Uma, D, Y. B. Venkata Reddy, Y. Raghuramulu and K. Ganesha Raj, "Deep learning and statistical models for detection of white stem borer disease in arabica coffee", The International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences, Vol. 42, pp.443-451, 2019.

Bhandarkar et al., (2019), proposed that CNN models may use these images of both healthy and infected plants to identify white stem bore infestations early on. modification of images. It is made up of 2425 images taken from coffee farms that show crops that are healthy, sick, and dead. Without the need for feature engineering, Convolutional Neural Networks can extract picture features from input data. Support Vector Machines (SVM) are another kind of regression technique with limited image datasets. The authors achieved an average accuracy of 85.5% using the Inception v3 transfer learning model. The main advantage of this work is Convolutional Neural networks are Deep Learning (DL) models that possess the ability to extract image features from input data without the need for future engineering. However, the primary limitation of this research is that the Support Vector Regression (SVR) models were either overfitting or did not fully capture the emerging pattern. [3].

[4] **C. Xie and Y. He, "Spectrum and image texture features analysis for early blight disease detection on eggplant leaves", Sensors, Vol. 16, No. 5, p.676, 2016.** Xie and He (2016) focused their

efforts on utilizing hyperspectral image processing for the detection of early blight disease on eggplant leaves. RGB, HSV, and HLS images are created from hyperspectral photographs. Then, the gray level co- occurrence matrix (GLCM) was used to extract the texture features. Early blight disease in eggplants was identified through the application of AdaBoost classificat and KNN models. Both models demonstrated strong performance in the testing sets, achieving classification rates surpassing 88.46%. The study's findings indicated that texture and spectrum characteristics are useful in the early identification of eggplant leaf blight. However, to achieve greater performance, this work can be extended in the future by adding shape- and color-based features in addition to texture features [4].

[5] W. Li, T. Zhu, X. Li, J. Dong and J. Liu, "Recommending advanced deep learning models for efficient insect pest detection", Agriculture, Vol. 12, No. 7, p.1065, 2022.

Li et al., (2022) proposed three Deep Convolutional Neural Network (DCNN) models for insect pest detection: Yolov5, Region-based Convolutional Neural Networks (RCNN), and Mask-RCNN. Yolov5 can detect insect pests in photos with simple backgrounds more accurately than Faster-RCNN and Mask-RCNN, with a speed that is around 2.5 times faster. Its accuracy is above 99%. However, in images that predict the affected part with more complex backgrounds and a higher abundance of insect categories. Insect pest etection is best served by Faster-RCNN and Mask- RCNN, which have accuracy levels exceeding 99% and often operate quickly. However, the drawback of this work is the complexity of implementation [5].

[6] F. Vanegas, D. Bratanov, K. Powell, J. Weiss and F. Gonzalez, " A novel methodology for improving plant pest surveillance in vineyards and crops using UAV-based hyperspectral and spatial data", Sensors, Vol. 18, No. 1, p.260, 2018

Vanegas et al., (2018) proposed a predictive model was created to identify pests in vineyards and crops. They achieved this by evaluating aerial photography captured by RGB, multispectral, and hyperspectral cameras to evaluate vineyard conditions. Expert visual assessments and a digital vigor model of the vineyard were compared in the study. Initially, to assess the extent and size of the infestation as well as its effect on grape yield, particular phylloxera or vegetation indices could be applied to photos taken from vineyards impacted by the plant pest. Nevertheless, a noteworthy constraint of the study is its insufficiency in emphasizing the development of predictive model employing these indices and assessing their precision in forecasting the existence or quantity of the plant pest [6].

A. Hessane, A. El Youssefi, Y. Farhaoui, B. Aghoutane and F. Amounas, "A Machine Learning Based Framework for a Stage-Wise Classification of Date Palm White Scale Disease", Big Data Mining and Analytics, Vol. 6, No. 3, pp.263- 272, 2023.

Hessane et al., (2023) presented a framework of White Scale Disease (WSD) infestation phases in date palm tree leaflet photos that can be analyzed and classified using machine learning approaches. They divided WSD into three infestation levels. Two datasets—one containing texture features based on the Gray-Level Co- occurrence matrix (GLCM) and the other containing color moments characteristics that combine the GLCM and Hue Saturation Value (HSV)—are used to train machine learning models for this purpose. Four machine learning models—Support Vector Machine (SVM), K closest neighbor (KNN), Random Forest (RF), and Light GBM— were used to achieve the goals. This work's primary benefit is demonstrating the potency of machine learning in conjunction with machine vision for the categorization and identification of diseases in the data. However, the availability of high-quality annotated image datasets is limited [7].

[7] L. Nanni, G. Maguolo and F. Pancino, "Insect pest image detection and recognition based on bio-inspired methods", Ecological Informatics, Vol. 57, p.101089, 2020.

Nanni et al. (2020) created an automatic classifier for insect pest picture detection of bio-inspired method. They investigated the feasibility of building an ensemble of classifiers by fusing saliency techniques with convolutional neural networks. The primary benefit of this work is its demonstration of the superiority of the Dense Net architecture, which promotes feature reuse and aids in gradient flow. The primary flaw is the Spectral residual (SPE), which ignores significant foreground areas and highlights very few pixels. Furthermore, this procedure frequently fails if the bug is very little [8].

[8] F. Ali, H. Qayyum, and M. J. Iqbal, "Faster-PestNet: A Lightweight deep learning framework for crop pest detection and classification", IEEE Access, 2023.

Ali et al. (2023) have introduced an automated Deep Learning system for agricultural pest detection and separation that is low-cost. Specifically, the Faster- PestNet model uses the Mobile Net approach as a core network for intensive feature collecting. The authors evaluated the approach with IP102 dataset collection. It is a large and demanding standard collection of photos taken in the field for pest identification. The outcomes showed that, even in the presence of complex backgrounds and variations in the shapes, colors, sizes, locations, and luminance of pests, our system could accurately identify and classify a wide variety of pests. The suggested methods were utilized in other agricultural applications, such as diagnosing pest-caused crop illnesses. However, because of the intricate data models, training is very costly. Furthermore, hundreds of workstations and costly GPUs are needed for deep learning. [9].

[9] Y. Li, Z. Luo, F. Wang, and Y. Wang, "Hyperspectral leaf image-based cucumber disease recognition using the extended collaborative representation model", Sensors, Vol. 20, No. 14, p.4045, 2020.

Li et al., (2020), proposed a cucumber disease identification problem should be used to develop an ECR-based classification model and evaluate its efficacy. Use the HSI technique to first probe the finely tuned spectral information associated with the disease in order to identify cucumber diseases. Next, develop pure and variation spectral libraries to characterise the pure spectral curves and linear interferences caused by lighting, occlusion, or other situations, respectively. Even with a small number of training samples, the ECR-based classification method can achieve good identification accuracy and fast online diagnostic speed, which could partially meet the demands of accurate and timely non-destructive diagnosis in real-world production. But the only downside is that maintaining the data set and its intricacy in order to accomplish the desired outcome [10]

SYSTEM ANALYSIS

3.1 EXISTING SYSTEM

- **Otsu Methodology:** The Otsu method is a widely used image thresholding technique that automatically calculates an optimal threshold to separate the foreground and background in an image. It minimizes intra-class variance, making it effective for applications such as image segmentation and object detection.
- **Gaussian Mixture Models (GMM):** Gaussian Mixture Models involve representing a complex probability distribution as a mixture of several Gaussian distributions. In image processing, GMMs are applied to model pixel intensity distributions, enabling tasks like image segmentation and background subtraction. They are particularly useful in scenarios with varying illumination conditions.
- **Manual Classification:** Manual classification involves human experts visually inspecting and categorizing image content. While accurate, it is labor-intensive and time-consuming. This method is often used as a benchmark for evaluating automated
  classification algorithms, providing a reference for performance comparison.

3.1.1 Disadvantages of Existing System

- Dependence on high-quality image data.
- Variability in lighting and environmental conditions.
- Initial setup and maintenance costs.
- Privacy Concerns
- Inefficient Handling of Data
- Human Error
- Resource-Intensive
- Scalability Issues

3.2 PROPOSED SYSTEM

Convolutional Neural Networks (CNNs) are a class of deep neural networks that have proven highly effective in areas such as image recognition and computer vision. They are designed to automatically and adaptively learn spatial hierarchies of features from input data. CNNs are particularly well-suited for tasks where the input has a grid- like topology, such as images.

- **User-friendly interface:** The proposed system will have a simple and intuitive user interface that allows users to easily upload images for the prediction of pest.
- **Advanced algorithms:** The proposed system will use advanced image processing and machine learning algorithms CNN to accurately capture the essence of the image and create high-quality outputs. Overall, the

proposed system aims to provide a more efficient and effective solution for images, with improved functionality and user experience.

3.2.1 Advantages of Proposed System

Convolutional Neural Networks (CNNs) offer several advantages in the context of pest infestation detection and management

1. **Feature Learning:** CNNs are adept at automatically learning discriminative features from raw image data. In the case of pest infestation detection, CNNs can identify subtle visual cues such as pest morphology, color variations, and patterns in leaves or crops associated with infestation.
2. **Accuracy:** CNNs can achieve high levels of accuracy in pest detection tasks due to their ability to learn complex patterns and features from large datasets. By training on diverse datasets containing images of both healthy and infested crops, CNNs can effectively distinguish between the two classes with minimal error.
3. **Efficiency:** Once trained, CNNs can process large volumes of images rapidly, making them efficient tools for monitoring pest infestations across vast agricultural areas. This efficiency allows for timely detection and intervention, helping farmers mitigate the spread of pests and minimize crop damage.
4. **Adaptability:** CNNs can be adapted to different types of pest infestations and agricultural settings. By fine-tuning the network on specific pest species or crops of interest, CNNs can tailor their detection capabilities to address the unique challenges posed by different pests and environmental conditions. Automation: CNN-based pest detection systems can be integrated into automated monitoring platforms, such as drones or unmanned aerial vehicles (UAVs). These systems can autonomously scan agricultural fields, analyze images in real-time, and alert farmers to potential pest infestations, enabling proactive pest management strategies.
5. **Scalability**: CNN-based pest detection solutions are scalable and can be deployed across diverse agricultural landscapes, from small-scale farms to large commercial plantations. By leveraging cloud-based computing resources, CNN models can be deployed remotely and scaled up or down based on the size and complexity of the agricultural operation.
6. **Cost-Effectiveness:** Compared to traditional methods of pest monitoring, which may involve manual inspection or the use of expensive equipment, CNN-based solutions can offer cost-effective alternatives. Once the initial investment in infrastructure and training data is made, CNN-based systems can operate with minimal ongoing costs, making them accessible to a wide range of farmers and agricultural stakeholders.
7. Overall, CNNs offer significant advantages in pest infestation detection, including accuracy, efficiency, adaptability, automation, scalability, and cost-effectiveness. By harnessing the power of deep learning and computer vision, CNN-based solutions can revolutionize pest management practices and contribute to sustainable agriculture.

3. 3 REQUIREMENTS SPECIFICATION

3. 3. 1 Hardware Requirements

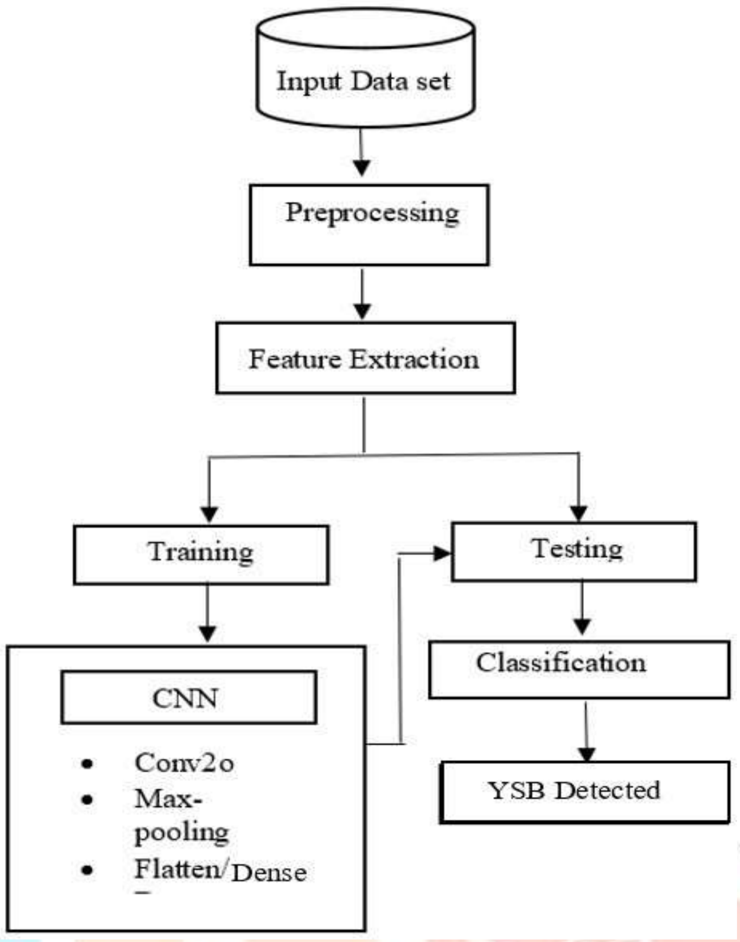
- Computer system with a minimum of 4GB RAM
- Processor with a clock speed of 2.5 GHz or higher
- Graphics card with a minimum of 1GB dedicated memory
- Display monitor with a minimum resolution of 1280x720 pixels
- High-speed internet connectivity

3. 3. 2 Software Requirements

- **Operating System:** Windows 10 or later version, or Mac OS X 10.11 or later version
- **Integrated Development Environment (IDE):** Python IDLE, Jupyter Notebook and Anaconda
- **Python Libraries:** OpenCV
- **Deep Learning Framework:** TensorFlow and Keras

SYSTEM DESIGN

4.1 YSB DETECTION



ig 4.1 Methodology diagram for detection of Yellow Stem Borer(YSB)

Figure 4.1 shows the methodology diagram of the detection of yellow stem borer. Where this diagram outlines the sequential steps involved in the detection of Yellow Stem Borer in pest infestation, starting from data collection and preprocessing to model training, evaluation, and deployment. Each step plays a crucial role in the overall methodology, leading to the development of an effective pest detection system.

4.2 NEURAL NETWORKS

A neural network consists of many Nodes (Neurons) in many layers. Each layer can have any number of nodes and a neural network can have any number of layers.

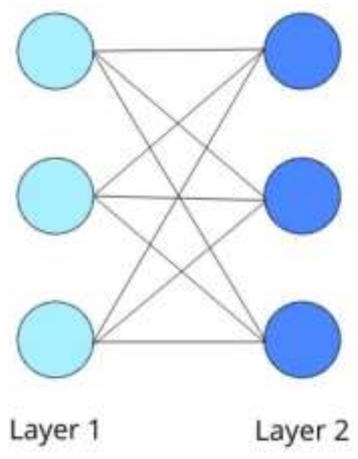


Fig 4.2 Layers in a Neural Network

Fig 4.2 shows there are many interconnections between both layers. These interconnections exist between each node in the first layer with each and every node in the second layer. These are also called the weights between two layers.

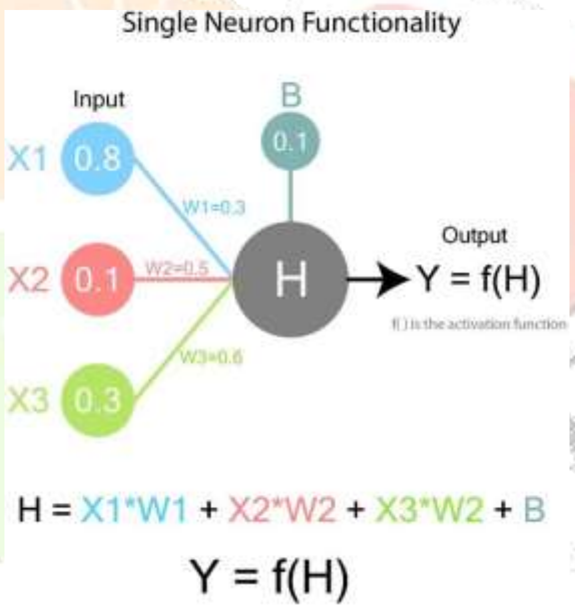

Single Neuron Functionality
Input
B
0.1
X1 0.8
W1=0.3
X2 0.1
W2=0.5
H
Output
Y = f(H)
f( ) is the activation function
W3=0.6
X3 0.3
H = X1*W1 + X2*W2 + X3*W2 + B
Y = f(H)

Fig 4.3 Diagram of single Neuron Function

Fig 4.3 is a single node in the network, Here we are considering all the values from the previous layer connecting to one node in the next layer. Y is the final value of the node.

- W represents the weights between the nodes in the previous layer and the output node.
- X represents the values of the nodes of the previous layer.
- B represents bias, which is an additional value present for each neuron. Bias is essentially a weight without an input term. It's useful for having an extra bit of adjustability which is not dependant on previous layer.
- H is the intermediate node value. This is not the final value of the node.
- f( ) is called an Activation Function and it is something we can choose. We will

go through it's importance later.

- So finally, the output value of this node will be f(0.57)

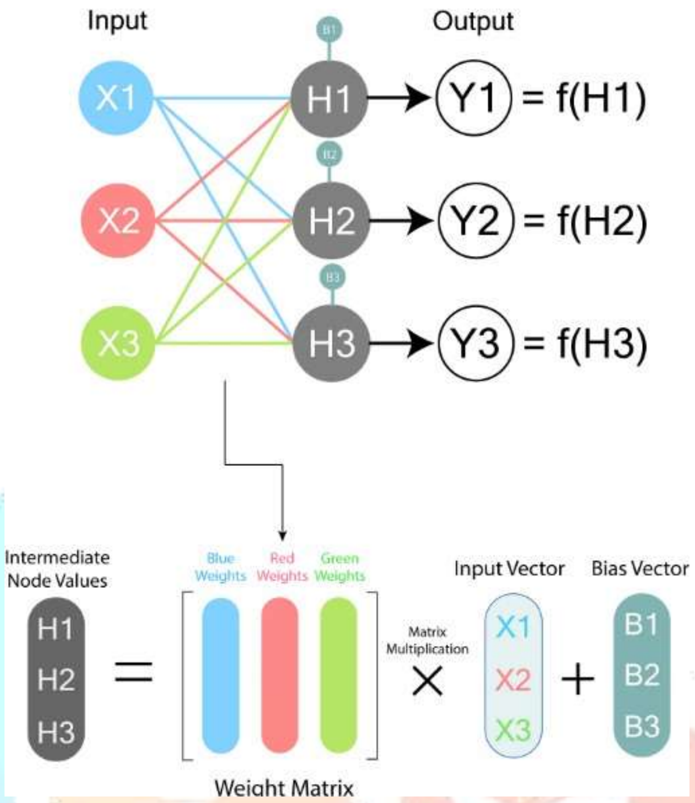


Fig 4.4 Calculation between two complete layers

From Fig 4.4 we can represent the entire calculation as a matrix multiplication. If we represent the weights corresponding to each input node as vectors and arrange them horizontally, we can form a matrix, this is called the weight matrix. Now we can multiply the weight matrix with the input vector and then add the bias vector to get the intermediate node values.

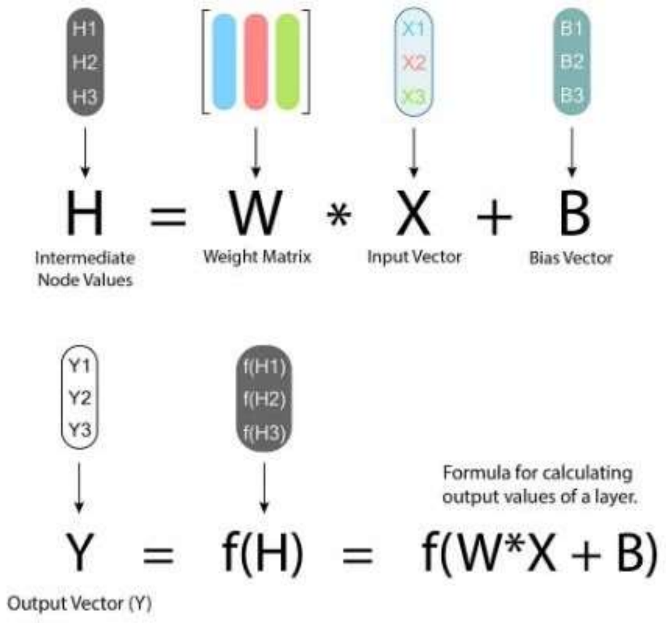


Fig 4.5 Calculation of output values of layer

To summarize the entire calculation as Y = f(W*X + B) from Fig 4.5 Here, Y is the output vector, X is the input vector, W represents the weight matrix between the two layers and B is the bias vector. We can determine the size of the weight matrix by looking at the number of input nodes and output nodes. An M*N weight matrix means that it is between two layers with the first layer having N nodes and the second layer having M nodes.

Fig 4.6 is a small neural network of four layers. The input layer is where we feed our external stimulus, or basically, the data from which our neural network has to learn from. The output layer is where we are supposed to get the target value, this represents what exactly our neural network is trying to predict or learn. All layers in between are called hidden layers. When we feed the inputs into the first layer, the values of the nodes will be calculated layer by layer using the matrix multiplications and activation functions till we get the final values at the output layer. That is how we get an output from a neural network.

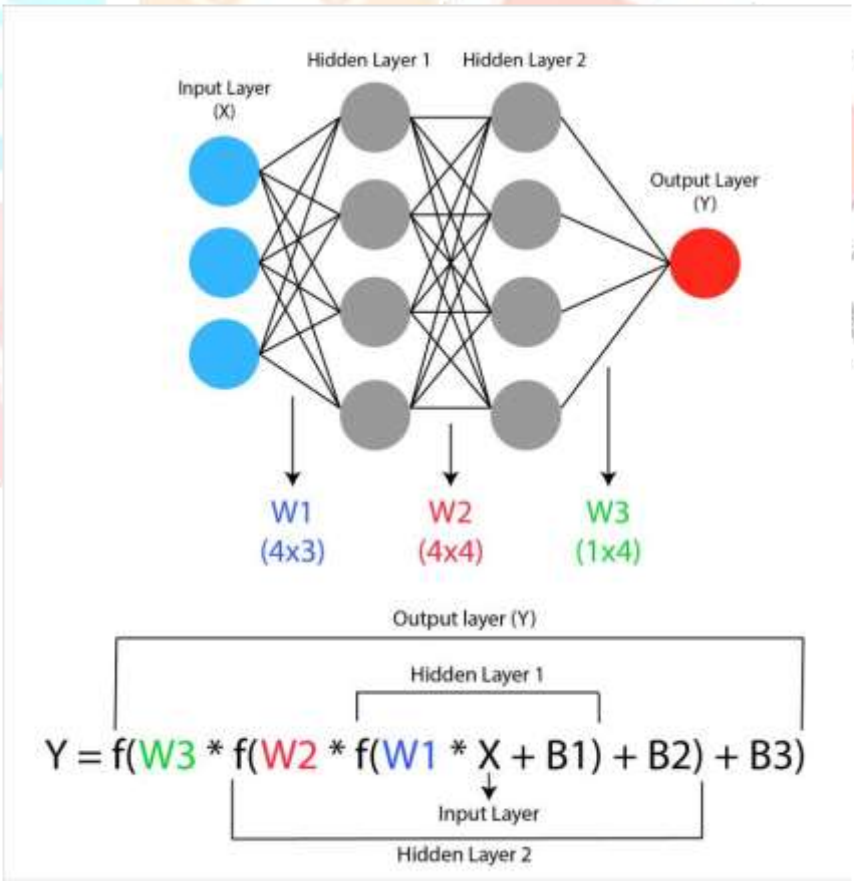


Fig 4.6 Detection of Hidden Layers

So essentially a neural network is, simply put, a series of matrix multiplications and activation functions. When we input a vector containing the input data, the data is multiplied with the sequence of weight matrices and subjected to activation functions till it reaches the output layer, which contains the predictions of the neural network corresponding to that particular input.

4.3 STRIDES IN A CNN

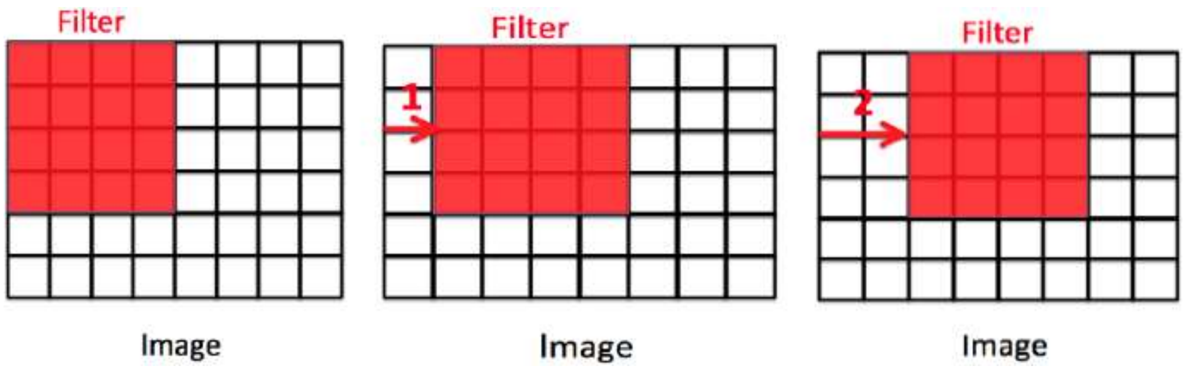


Fig 4.7 Diagram of transition of pixel

From Figure 4.7, we can infer the main operation that a CNN network uses for everything is the convolution operation. This operation is the foundation for the Convolutional Neural Network. As you can see, the image is made up of several pixels. The image pixels in the upper-left corner are our primary focus. We focus on the shaded area in red, which has 3x3 pixels and a center pixel value of 6.
The labeled convolution filter for a 3x3 matrix is the one we are employing on the image. A kernel is another name for this filter. In this instance, the kernel used is the Sobel Gx kernel. You can see the values of the kernel. On the image's upper side, you can also see the convolution process.

Now, by calculating the element-wise dot product, the kernel adds up the values of the red-shaded region and the kernel. Applying the kernel or filter to the source pixel values and performing element-by-element multiplication precedes the sum calculation. The following is the convolutional operation. The first value of the filter and the red-shaded pixels are 3, respectively. As a result, when we add them up, we get -3. The multiplications of subsequent elements, and so on, are then computed. Because it is a parameter that we can define for the model, a stride is a hyperparameter. A stride of one is therefore used to represent moving the kernel or filter just one pixel at a time. The stride parameter specifies how the transition from one pixel to the next should be captured. When the stride is low, we get more information from the data, but when the stride is high, we don't get as much. As a result, strides essentially specify the number of units by which the data ought to be moved to the right of the kernel. Stride refers to how many spaces the filter will be applied by skipping in the input image.

- Stride = 1, apply filter by skipping one space at a time
- If you pad the (n,n) image by p and then stride and convulute it by s with the (f,f) filter,
- The size of the output image is ((n+2p-f)/s+1, (n+2p-f)/s+1).

4.4 POOLING IN CNNs

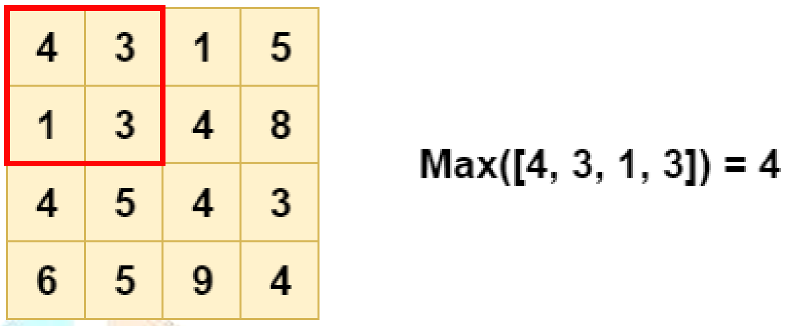
In CNN, there is a pooling layer in addition to the convolution layer. And polling can reduce network complexity and computational cost and There are two types of Pooling

1. Max Pooling
2. Average Pooling

1. **Max Pooling**

The majority of the time, pooling is used to extract the most information from the data while requiring less computational power. A down sampling process, in which the data are reduced in size and only the relevant information is retained, can be compared to the pooling procedure. Pooling helps prevent overfitting. Pooling also provides an output matrix of a fixed size, regardless of the size of the input or filter. This is a very useful feature if we apply kernels of varying sizes or if our input data sizes are inconsistent.

Fig 4.8 shows the Max pooling is the most common pooling technique. In max pooling, the maximum value across a filter is used. The maximum pooling is depicted in the image below. Here, we take a stride of 2. The orange boxes serve as the maximum value across the filter when the filter is applied. The maximum value across the orange filter, which is 20, is used to increase the pooling output in this instance.



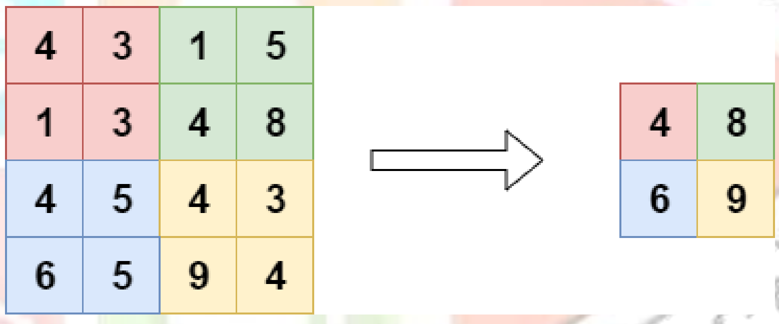


Fig 4.8 Matrix of Max Pooling

Average pooling In average pooling, we take the average of all the values across the applied filter. We calculate by taking the sum of all of the values in the applied filter. Average pooling is depicted in the image below. When the orange filter is used here, all of the values inside are taken into account, averaged, and added to the final pooling output. After dividing the input image into several regions equal to the filter size, this process is done by moving the filter over the image and extracting the maximum value within the filter area. Max Pooling helps capture important information from images. Figure Above Max Pooling with a stride of 2 using the (2,2) filter for the (4,4) image.

2. Average Pooling

Fig 4.9 shows the average pooling which After dividing the input image into several regions equal to the filter size, this process is done by moving the filter over the image and extracting the average value within the filter area. Average Pooling helps balance important and less important information in an image. The below mentioned fig is the Average Pooling with a stride of 2 using the (2,2) filter for the (4,4) image.

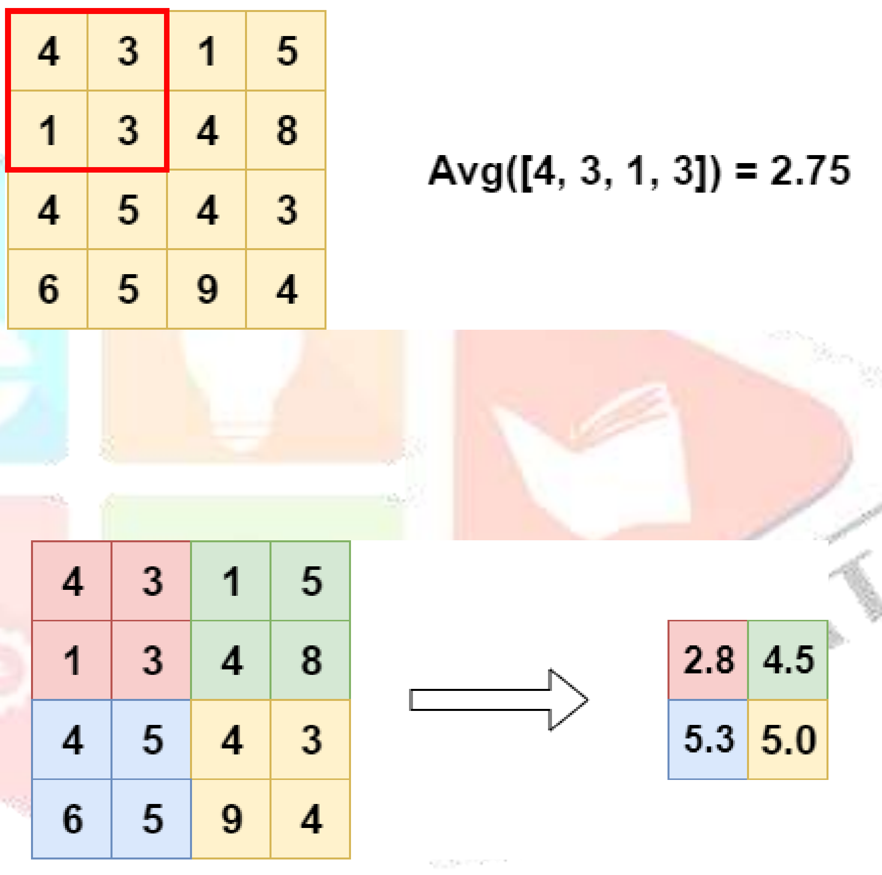


Fig 4.9 Matrix of Average pooling

4.5 IMAGE RECOGNITION

Image recognition is a field of computer vision that focuses on identifying and classifying objects within digital images. With the rise of powerful and affordable computer hardware, deep learning techniques, and large annotated datasets, image recognition has seen significant progress in recent years, and is now widely used in many applications such as security systems, medical diagnosis, and autonomous vehicles.

Fig 4.10 shows the basic process of image recognition involves several steps. First, an image is input into the system, either from a file or from a camera. Next, the image is preprocessed to prepare it for analysis. This may involve resizing the image, converting it to grayscale, or normalizing pixel values. The preprocessed image is then fed into a feature extraction process, which generates a set of features that describe the image's content. These features are used as input to a classifier, which predicts the class or label of the image based on the extracted features.

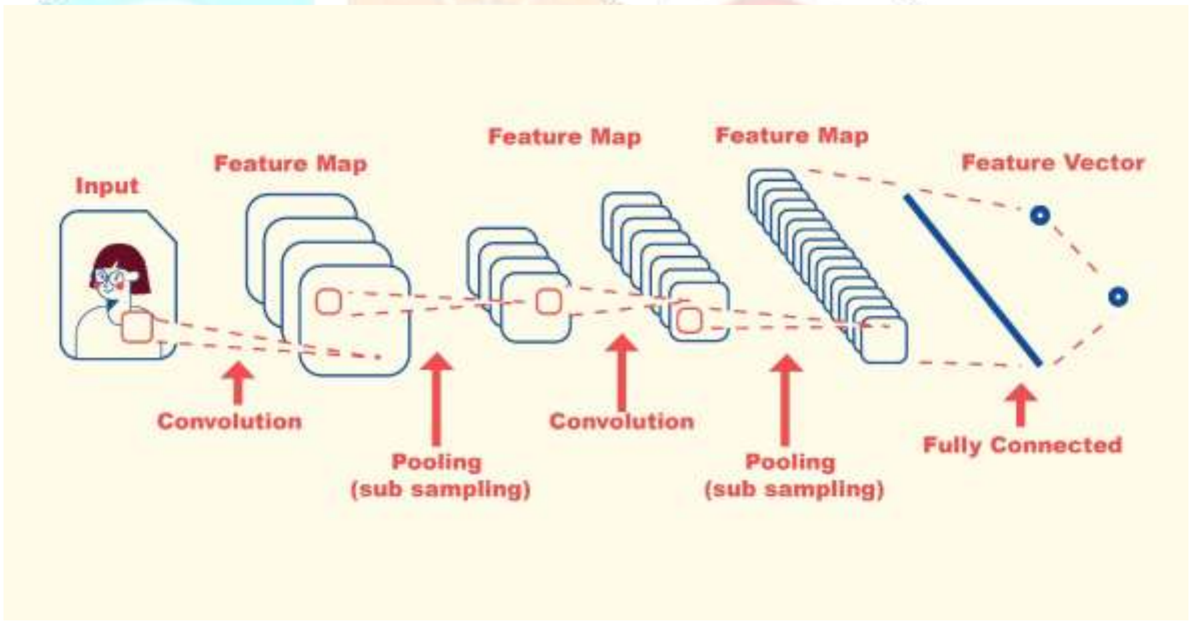


Fig 4.10 Sample test Recognition of Image

One of the most widely used techniques for feature extraction is Convolutional Neural Networks (CNNs). CNNs are deep neural networks specifically designed for image recognition tasks, and are composed of multiple layers of convolutional filters, activation functions, and pooling layers. The convolutional filters extract features from the image, while the activation functions and pooling layers reduce the dimensionality of the feature map and introduce translation invariance.The classifier used in image recognition can be a variety of machine learning models, such as Support Vector Machines (SVMs), Random Forests, or Artificial Neural Networks (ANNs). In recent years, deep learning models, such as Multilayer Perceptrons (MLPs) or Convolutional Neural Networks (CNNs), have become popular for image recognition due to their high accuracy and ability to learn complex relationships between the input and output.

To train an image recognition model, a large annotated dataset is required. The annotated dataset consists of images and their corresponding labels, and is used to train the model to recognize different objects. During training, the model is exposed to many examples of each class, and adjusts its weights and biases to minimize the difference between its predictions and the true labels. The process is repeated multiple times until the model reaches a satisfactory level of accuracy. Once the model has been trained, it can be used for image recognition on new, unseen images. The preprocessing and feature extraction steps are the same as during training, and the output of the feature extraction is used as input to the trained classifier. The classifier then outputs a prediction for the class or label of the image. The performance of an image recognition system can be evaluated using a variety of metrics, such as accuracy, precision, recall, and F1 score. These metrics measure the ability of the system to correctly classify images, and are useful for comparing different models and for detecting any weaknesses in the system. Despite the progress made in recent years, there are still several challenges that must be addressed in the field of image recognition. One of the main challenges is robustness, as current models may not be able to handle variations in lighting conditions, background clutter, or object position by the develop a CNN- based image recognition system for detecting Yellow Stem Borer pest infestation in crops, helping farmers identify and mitigate the impact of YSB on agricultural productivity.

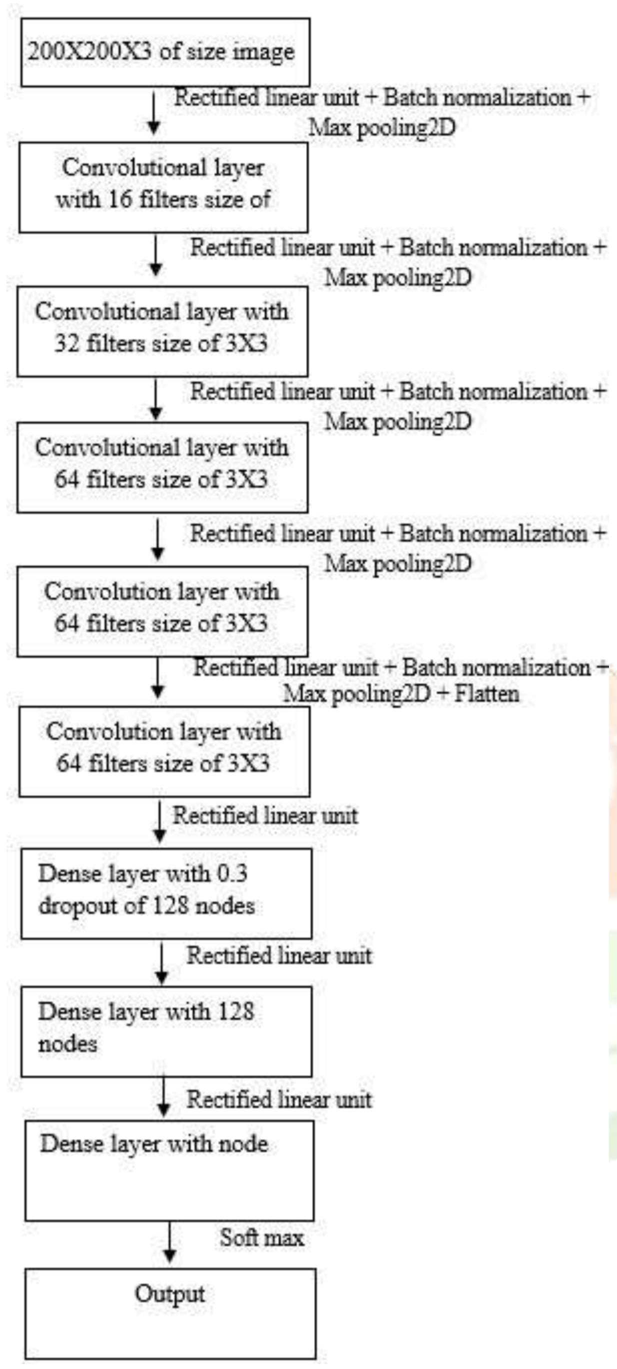

200X200X3 of size image
Rectified linear unit + Batch normalization + Max pooling2D
Convolutional layer with 16 filters size of
Rectified linear unit + Batch normalization + Max pooling2D
Convolutional layer with 32 filters size of 3X3
Rectified linear unit + Batch normalization + Max pooling2D
Convolutional layer with 64 filters size of 3X3
Rectified linear unit + Batch normalization + Max pooling2D
Convolution layer with 64 filters size of 3X3
Rectified linear unit + Batch normalization + Max pooling2D + Flatten
Convolution layer with 64 filters size of 3X3
Rectified linear unit
Dense layer with 0.3 dropout of 128 nodes
Rectified linear unit
Dense layer with 128 nodes
Rectified linear unit
Dense layer with node
Soft max
Output

Fig 4.11 Simple CNN Architecture

In figure 4.11 it is clearly seen that the a fully convolutional network uses a convolutional neural network to transform image pixels to pixel classes by pooling of image in various layers. Unlike the CNNs that we encountered earlier for image classification or object detection, a fully convolutional network transforms the height and width of intermediate feature maps back to those of the input image. As a result, the classification output and the input image have a one-to-one correspondence in pixel level, the channel dimension at any output pixel holds the classification results for the input pixel at the same spatial position.

At the vegetative stage, we employed a Convolutional Neural Network (CNN) algorithm to detect the effects of damaged crops. CNNs are a category of deep learning algorithms known for their exceptional aptitude in image recognition and processing tasks. These networks are composed of multiple layers, including convolutional layers, pooling layers, and fully connected layers. The heart of a CNN lies in its convolutional layers, where filters are applied to the input image, enabling the extraction of critical features such as edges, textures, and shapes. Subsequently, the output from the convolutional layers undergoes further processing in pooling layers, which serve to down-sample the feature maps. This downsizing reduces spatial dimensions while preserving the most pertinent information. Following the pooling layers, the data is then channelled through one or more fully connected layers. These layers play a pivotal role in making predictions or classifying the image, depending on the specific task at hand. To enable the CNN to perform effectively, it is trained using a substantial dataset of labelled images. During training, the network learns to recognize patterns and features associated with specific objects or classes. Once the CNN is trained, it can be utilized to classify new images or extract features for use in a variety of applications, including object detection and image segmentation.

The two-stage training approach was inspired by fine-tuning techniques. In the first stage, the entire image dataset consisting of nine classes was subdivided into 17 classes, with each intra-class variation placed in separate classes. For example, the "others" class was divided into three distinct classes. Consequently, the model was trained using this 17-class dataset, resulting in a final dense layer with 17 nodes utilizing the softmax activation function.

In the second stage, the original nine-class dataset was utilized. While maintaining all layer weights obtained from the first stage, the topmost layer of the Simple CNN architecture underwent modification. The dense layer with 17 nodes was replaced with a dense layer comprising nine nodes and a softmax activation function. This modification was necessary as the second stage training data pertained to the original nine classes. Subsequently, all layers of the Simple CNN architecture were trained using this nine-class dataset, initialized with pre-trained weights obtained from the first stage of training. Experimental results validate the effectiveness of this methodology.

- **UML DIAGRAMS**

SEQUENCE

DIAGRAM

A sequence diagram simply depicts interaction between objects in a sequential order i.e. the order in which these interactions take place.

The Fig 4.12 shows the diagram visually represents the interactions between different components or objects in a system over time. In the context of using a Convolutional Neural Network (CNN) for pest infestation detection, the sequence diagram would illustrate the flow of activities and data between various components involved in the process. Here's a basic outline of a sequence diagram for CNN pest infestation detection.

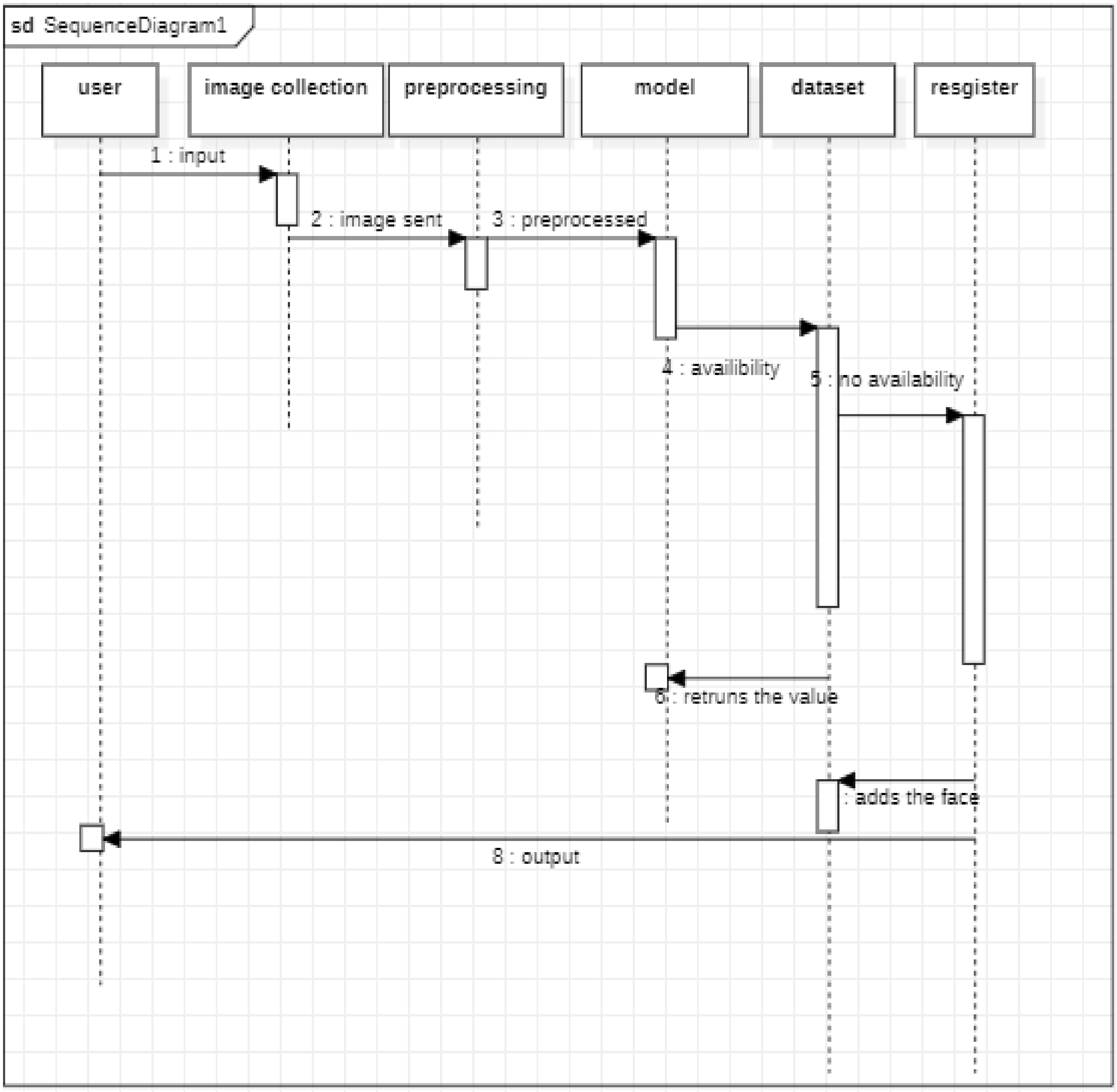


Fig 4.12 Sequence diagram

The User captures or obtains images of crops for pest infestation detection. The raw images are sent to the Preprocessing Module for preprocessing, which may include resizing, normalization, and other transformations. The preprocessed images are then forwarded to the CNN Model for further processing. The Training Module receives the preprocessed images and trains the CNN model on the dataset.

After training, the trained model is evaluated by the Evaluation Module to assess its performance using metrics like accuracy, precision, and recall. Once the model is trained and evaluated, it is saved by the Deployment Module for deployment in real- world scenarios. Finally, the User is informed that the trained model is ready for deployment, and it can be used for pest infestation detection on new images. This sequence diagram outlines the high-level interactions between different modules involved in the process of using a CNN for pest infestation detection.

MODULE DESCRIPTION

5.1 LIST OF MODULES

The system consists of the following Nine modules:

- Data Collection
- Data Preprocessing
- Feature Extraction
- Model Design
- Model Training
- Model Evaluation
- Model Optimization
- Deployment
- Monitoring and Maintenance

5.2 DATA COLLECTION

Data preprocessing is a crucial step undertaken before CNN training, aiming to ready the data for optimal learning. Tasks in this phase encompass standardizing image sizes, normalizing pixel values, and partitioning the data into training, validation, and test sets. This process not only streamlines input images but also expedites subsequent executions. Techniques such as resizing, color conversion, and filtering are applied to enhance efficiency. At first, the Identify Target Pests is done to determine which pests you want to detect using CNNs. This could include pests like aphids, whiteflies, leafhoppers, or specific species such as Yellow Stem Borers. Secondly, Select Crop

Types Identify the crops or plants that are susceptible to infestation by the target pests.

This could include crops like rice, maize, wheat, soybeans, or specific horticultural plants. Then Gather Images Collect a large and diverse dataset of images depicting both healthy plants and plants infested with the target pests. It's essential to capture images under various conditions, including different growth stages, lighting conditions, and angles. Where Capture Infestation Levels Ensure that the dataset includes images with varying degrees of pest infestation, ranging from mild to severe infestation levels. This will help CNN learn to recognize different stages of infestation Which Include Negative Examples Along with images of infested plants, include images of healthy plants as negative examples in the dataset. This will help CNN learn to differentiate between healthy and infested plants accurately.

By which it Ensure Data Quality Verify the quality of the collected images by ensuring they are clear, properly labeled, and free from noise or artifacts that could affect the CNN's performance. Then Augment the Dataset uses data augmentation techniques to increase the variability of the dataset. This can include techniques such as rotation, flipping, scaling, and adding noise to artificially generate additional training examples. To Balance Classes Ensure that the dataset is balanced, meaning there is an equal number of examples for each class (healthy and infested). This will prevent the CNN from being biased towards the majority class and improve its overall performance. Which Split Dataset Divide the dataset into training, validation, and testing sets. The training set is used to train the CNN, the validation set is used to tune hyperparameters and monitor performance during training, and the testing set is used to evaluate the final performance of the trained model.

At last, the document Metadata Maintains detailed documentation of the dataset, including information such as image sources, labels, infestation levels, and any preprocessing or augmentation techniques applied. This documentation will be valuable for reproducibility and future reference. By following these steps, you can ensure that the data collection process for training CNNs for pest infestation detection is comprehensive, diverse, and representative, leading to better-performing models in real-world applications.

5.3 DATA PREPROCESSING

Before the data can be used to train the CNN, it must be preprocessed to prepare it for training. This may involve resizing images to a standard size, normalizing pixel values, and splitting the data into training, validation, and test sets. Resize the images to a uniform size suitable for input into the CNN. This step ensures that all images have the same dimensions, which is necessary for feeding them into the network. Common image sizes for CNNs are often in the range of 200x200 pixels or 256x256 pixels. This leads to Normalizing the pixel values of the images to a common scale to make the training process more stable and efficient. Typically, pixel values are scaled to the range [0, 1] or [-1, 1], and Label Encoding Encodes the class labels of the images into numerical representations suitable for training the CNN model. This could involve assigning unique numerical identifiers to each class or using one-hot encoding for multi-class classification tasks. This progress to Data Augmentation Apply data augmentation techniques to increase the variability of the dataset and improve the generalization ability of the CNN model. Common augmentation techniques include rotation, flipping, scaling, translation, cropping, and adding noise, where Splitting Dataset Splits the preprocessed dataset into training, validation, and testing sets. The training set is used to train the CNN model, the validation set is used to tune hyperparameters and monitor model performance, and the testing set is used to evaluate the final performance of the trained model forms Batching where the

Organize the preprocessed dataset into batches to feed into the CNN model during

training. Batching helps improve training efficiency by processing multiple images simultaneously and leveraging parallelism in hardware. Turns to Data Shuffling: Shuffle the training dataset to randomize the order of samples in each epoch. Shuffling prevents the CNN model from learning the order of the training data and helps avoid overfitting process to the Optional Preprocessing Techniques Depending on the specific characteristics of the dataset and the requirements of the CNN model, additional preprocessing techniques such as histogram equalization, edge detection, or color space conversion may be applied to enhance feature extraction last is the Documentation to Maintain detailed documentation of the preprocessing steps applied to the dataset, including any transformations, augmentations, or adjustments made. This documentation helps ensure reproducibility and facilitates troubleshooting.

By following these preprocessing steps, you can prepare the dataset for training CNN models for pest infestation detection effectively, leading to better model performance and generalization to unseen data.

5.4 FEATURE EXTRACTION

The Gray level co-occurrence matrix (GLCM) serves as a texture analysis technique in the realm of digital image processing. This method delineates the relationship between neighboring pixels in terms of gray intensity, distance, and angle. Typically, GLCM is employed to texture features, including dissimilarity, correlation, homogeneity, contrast, and others. The aim is to predict, identify, and classify the disease and pest impacting rice, the dataset is trained in such a way as to fulfill this aim. A co-occurrence matrix is generated by determining the frequency of occurrence of a pixel with intensity (gray-level) value i adjacent to a pixel with value j within a specified spatial relationship.

In detail the feature extraction process in Convolutional Neural Networks (CNNs) for pest infestation detection involves several key steps. Firstly, the preprocessed images are passed through a series of convolutional layers, where each layer applies learnable filters to extract spatial features such as edges, textures, and patterns. Activation functions like ReLU or sigmoid are then applied to introduce non-linearity into the feature maps, enhancing the model's ability to learn complex relationships. Pooling layers like max pooling or average pooling are utilized to downsample the feature maps, retaining relevant information while reducing spatial dimensions and computational complexity. Subsequently, the output of the last pooling layer is flattened into a 1D vector, making it suitable for input to fully connected layers, which perform high-level feature extraction and capture complex relationships between features. Activation functions are applied to the outputs of these fully connected layers to introduce non-linearity and enable the model to learn complex decision boundaries. Optionally, dropout regularization may be implemented to prevent overfitting by randomly dropping a percentage of neurons during training. The output layer is designed with the appropriate number of nodes corresponding to the classes of interest, and an appropriate activation function, such as softmax for multi-class classification, is used to produce class probabilities. Optional visualization techniques, like feature maps visualization, can provide insights into the learned representations. Finally, maintaining detailed documentation of the feature extraction process, including CNN architecture details and any customization or optimization techniques, aids reproducibility and facilitates model interpretation and refinement. Through these steps, CNNs can effectively extract relevant features from preprocessed images for accurate pest infestation detection in crops.

5.5 MODEL DESIGN

The architecture of the CNN is crafted to suit the proposed system, entailing decisions on the layer count, layer types, and the quality of filters in each layer. Optimizing the design for a balance of accuracy and computational efficiency is crucial. The input shape is the desired size of the image 200x 200 with 3 bytes of color bytes. In this design, we have used layers like Conv2D, Max pooling, Flatten, Dense layers, and ReLU in CNN, which play a vital role in achieving the accuracy and loss of the dataset. Based on the layers classification, the image is trained into the desired shape and size.

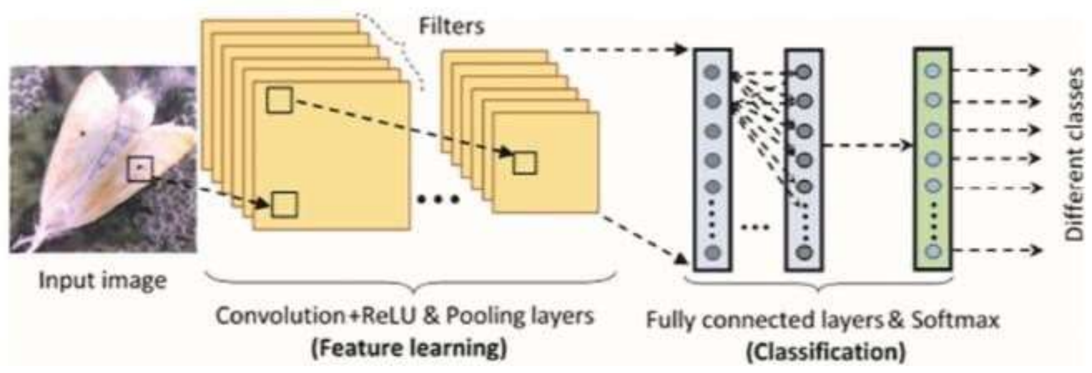


Fig 5.1 Working model of the layers

The Fig 5.1 is the working model layer of process in Convolutional Neural Networks (CNNs) for pest infestation detection is multifaceted, involving several critical steps to create an architecture capable of effectively learning and classifying images of both healthy and infested crops. Initially, the input dimensions of the images are defined, specifying parameters such as height, width, and the number of color channels. Subsequently, a suitable CNN architecture is chosen from options like VGG, ResNet, Inception, MobileNet, or customized architectures tailored to specific needs. The design of convolutional layers follows, determining factors such as the number of layers, filter sizes, and the depth of the network to extract features from input images.

Activation functions are then selected to introduce non-linearity into the model, with ReLU commonly used for hidden layers and softmax for multi-class classification at the output layer. Pooling layers are incorporated to downsample feature maps and reduce spatial dimensions, improving computational efficiency. Optional dropout regularization can be implemented to prevent overfitting by randomly dropping neurons during training. Fully connected layers are designed for high-level feature extraction and relationship capture, with the number of nodes determined based on task complexity. The output layer configuration corresponds to the classes of interest, with appropriate activation functions to produce class probabilities. Model optimization involves selecting suitable algorithms and loss functions for training, experimenting with various hyperparameters, and fine-tuning them for optimal performance on validation data. Hyperparameter tuning, often through techniques like grid search or random search, is crucial for achieving superior model performance. Finally, model evaluation on a separate test dataset using metrics like accuracy, precision, recall, and F1-score is conducted, followed by detailed documentation of the entire model design process to facilitate reproducibility, interpretation, and refinement. Through these steps and iterative experimentation, an effective CNN model can be designed for accurate pest infestation detection in crops.

5.5.1 RELU ACTIVATION IN CNN

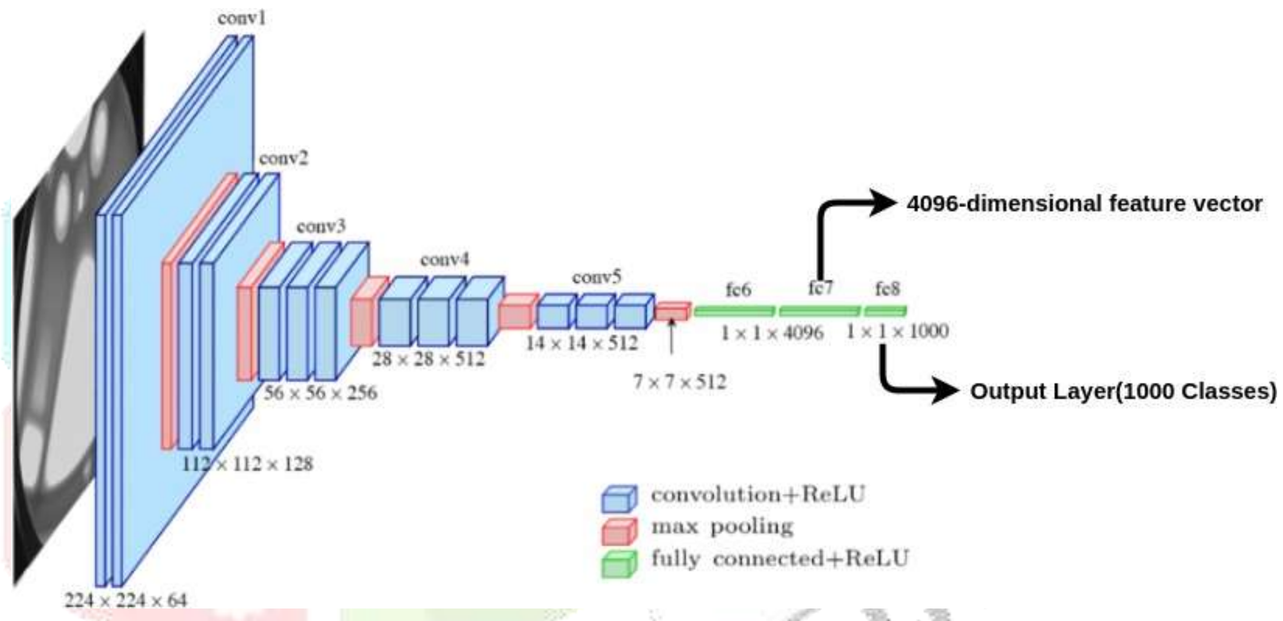


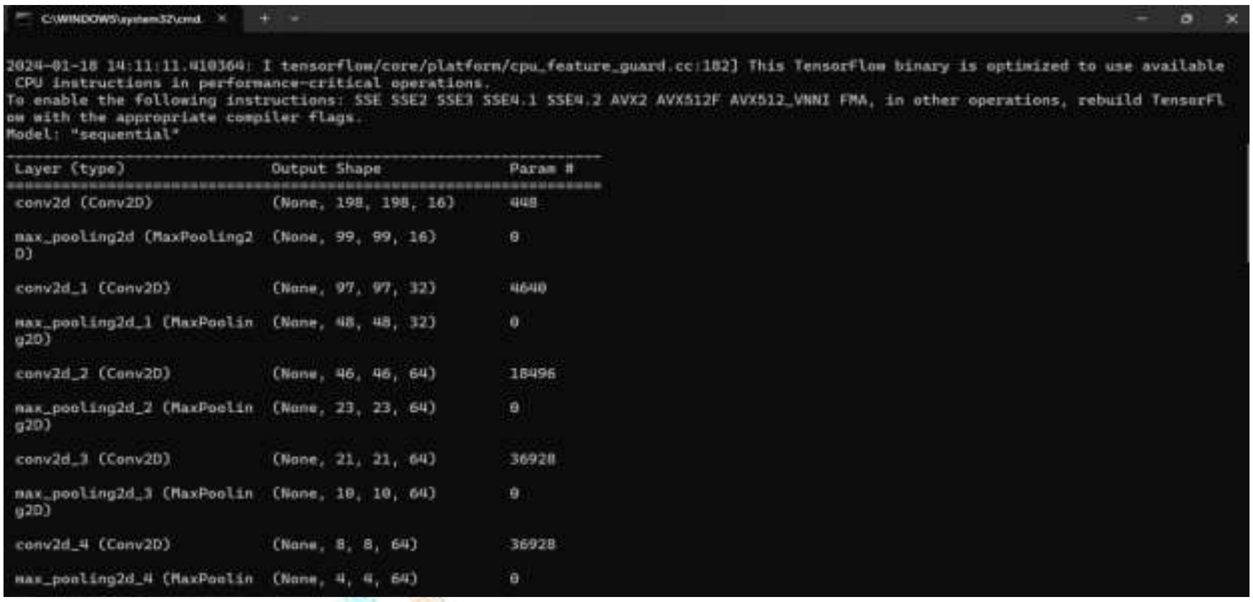
Fig 5.2 Layers with ReLU

Fig 5.2 shows the Rectified Linear Unit (ReLU) serves as a non-linear activation function employed in multi-layer neural networks to introduce non-linearity during the network's operation. The ReLU is not a separate component of the convolutional neural networks' process.

The rectified linear activation function effectively addresses the vanishing gradient problem, enabling models to learn more rapidly and achieve improved performance. The classification of layers with ReLu activation is instrumental in attaining the desired shape and size for the model.

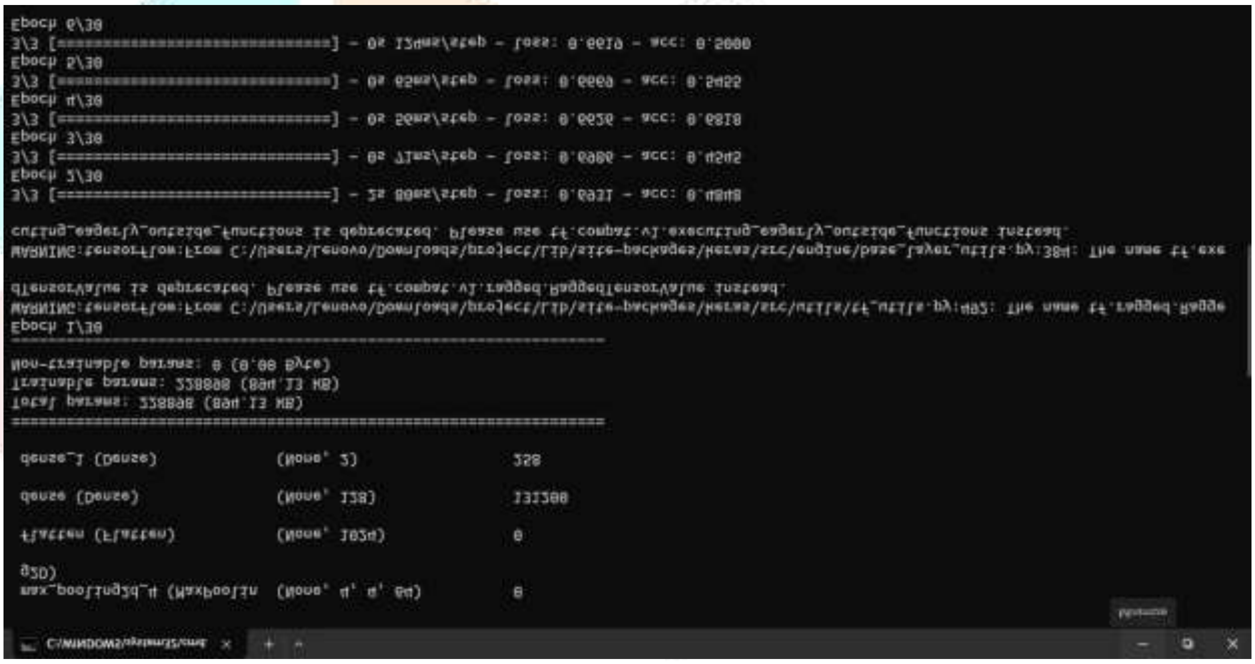
The ReLU (Rectified Linear Unit) activation function plays a crucial role in Convolutional Neural Networks (CNNs) utilized for pest infestation detection. Throughout the CNN architecture, the ReLU function is applied to the output of convolutional and fully connected layers during the forward pass. Mathematically, ReLU is defined as \(f(x) = \max(0, x) \), meaning it outputs the maximum between its input \(x \) and zero. By introducing non-linearity, ReLU enables the CNN to learn complex patterns and relationships in the data, which is essential for accurately distinguishing between healthy and infested crops. Additionally, ReLU promotes sparsity by setting negative values to zero, enhancing training efficiency and reducing the risk of overfitting by preventing the co-adaptation of neurons.

During backpropagation, the derivative of ReLU simplifies gradient computation, leading to more efficient training compared to other activation functions like sigmoid or tanh. Its computational efficiency and resilience to vanishing gradient problems make ReLU a preferred choice for activation functions in CNNs, facilitating faster convergence during training, especially in deep networks. In the context of pest infestation detection, ReLU activation functions are instrumental in enabling the CNN model to effectively learn and extract relevant features from images of healthy and infested crops, contributing to accurate classification performance.



```
2024-01-18 14:11:11.410364: I tensorflow/core/platform/cpu_feature_guard.cc:182] This TensorFlow binary is optimized to use available
 CPU instructions in performance-critical operations.
To enable the following instructions: SSE SSE2 SSE3 SSE4.1 SSE4.2 AVX2 AVX512F AVX512_VNNI FMA, in other operations, rebuild TensorFl
ow with the appropriate compiler flags.
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 198, 198, 16)      448

 max_pooling2d (MaxPooling2  (None, 99, 99, 16)        0
 D)

 conv2d_1 (Conv2D)           (None, 97, 97, 32)        4640

 max_pooling2d_1 (MaxPoolin  (None, 48, 48, 32)        0
 g2D)

 conv2d_2 (Conv2D)           (None, 46, 46, 64)        18496

 max_pooling2d_2 (MaxPoolin  (None, 23, 23, 64)        0
 g2D)

 conv2d_3 (Conv2D)           (None, 21, 21, 64)        36928

 max_pooling2d_3 (MaxPoolin  (None, 10, 10, 64)        0
 g2D)

 conv2d_4 (Conv2D)           (None, 8, 8, 64)          36928

 max_pooling2d_4 (MaxPoolin  (None, 4, 4, 64)          0
```

(a)



```
 max_pooling2d_4 (MaxPoolin  (None, 4, 4, 64)          0
 g2D)

 flatten (Flatten)           (None, 1024)              0

 dense (Dense)               (None, 128)               131200

 dense_1 (Dense)             (None, 2)                 258

=================================================================
Total params: 228898 (894.13 KB)
Trainable params: 228898 (894.13 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
Epoch 1/30
WARNING:tensorflow:From C:\Users\Lenovo\Downloads\project\lib\site-packages\keras\src\utils\tf_utils.py:492: The name tf.ragged.Ragge
dTensorValue is deprecated. Please use tf.compat.v1.ragged.RaggedTensorValue instead.

WARNING:tensorflow:From C:\Users\Lenovo\Downloads\project\lib\site-packages\keras\src\engine\base_layer_utils.py:384: The name tf.exe
cuting_eagerly_outside_functions is deprecated. Please use tf.compat.v1.executing_eagerly_outside_functions instead.

3/3 [==============================] - 3s 808ms/step - loss: 0.6931 - acc: 0.4848
Epoch 2/30
3/3 [==============================] - 0s 71ms/step - loss: 0.6909 - acc: 0.4242
Epoch 3/30
3/3 [==============================] - 0s 50ms/step - loss: 0.6939 - acc: 0.6818
Epoch 4/30
3/3 [==============================] - 0s 65ms/step - loss: 0.6669 - acc: 0.5455
Epoch 5/30
3/3 [==============================] - 0s 13ms/step - loss: 0.6616 - acc: 0.5000
Epoch 6/30
```

(b)

Fig 5.3 (a) and (b) CNN layers of YSB detection

Fig 5.3 is a Convolutional Neural Network (CNN) employs various layers to process input data and extract features crucial for its functioning. These layers include Convolutional, Activation, Pooling, Batch Normalization, Dropout (optional), Flatten, and Fully Connected layers. The Convolutional layer applies learnable filters to the input image, generating feature maps through convolution. Output shape depends on factors like input size, filter/kernel dimensions, stride, and padding, while parameters include filter count, size, stride, and padding values. Activation layers, like ReLU, introduce non-linearity to the network without altering shape, while Pooling layers downsample feature maps, affecting output shape based on parameters like pool size and stride. Batch Normalization normalizes layer activations, maintaining shape. Dropout layers randomly deactivate neurons during training, controlling overfitting. The Flatten layer reshapes the output into a 1D vector, suitable for Fully Connected layers, which perform high-level feature extraction and relationship capture, with output shape contingent on neuron count. Parameters for Fully Connected layers involve neuron count and optional activation functions. Each layer type contributes uniquely to feature extraction, enabling the model to learn hierarchical representations and make accurate predictions. Output shapes and parameters are contingent upon specific CNN architectures and configurations.

5. 6 MODEL TRAINING

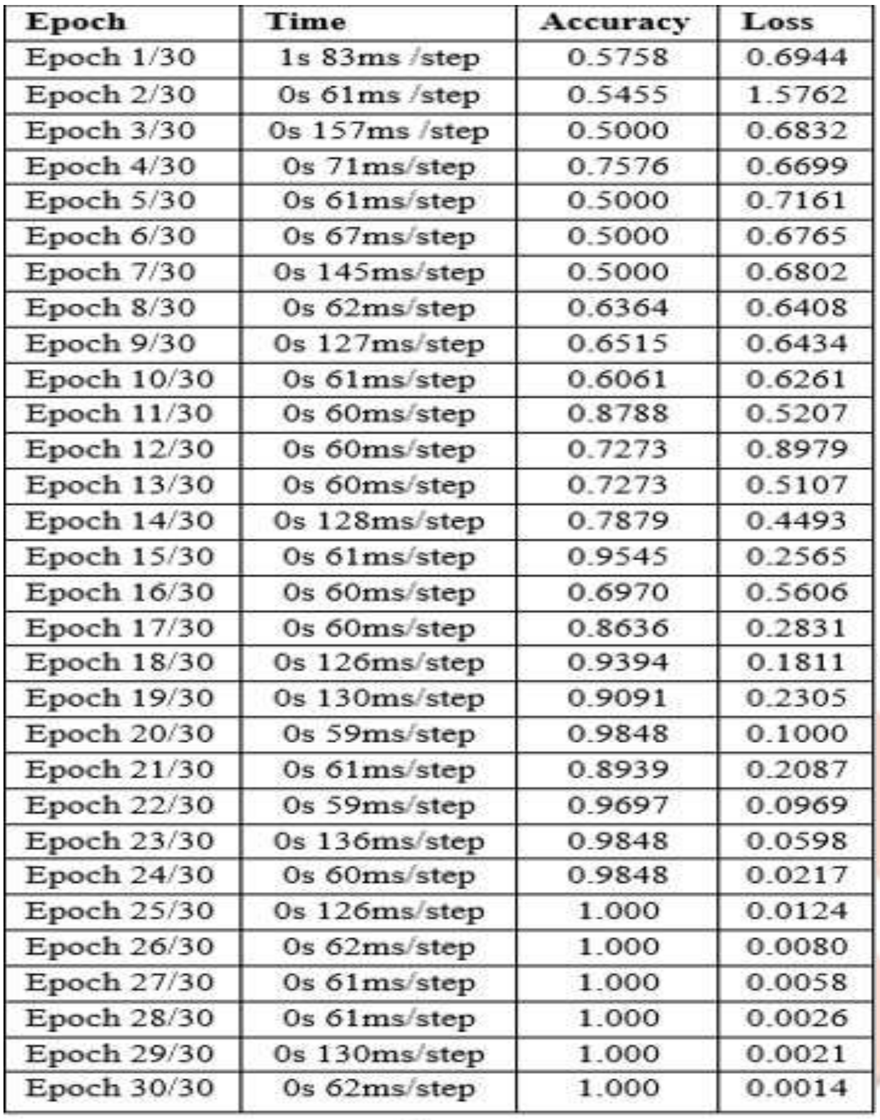
Training the CNN model involves utilizing pre-processed data. Throughout this training process, the model iteratively fine-tunes its weights and biases to minimize the disparity between its predictions and the actual labels. The process is repeated multiple times over the training data until the model reaches a satisfactory level of accuracy. The Epoch in Neural networks plays a major role in the level of accuracy and loss in training.

The training process of a Convolutional Neural Network (CNN) for pest infestation detection encompasses several pivotal stages. Initially, a diverse dataset is compiled, comprising images of healthy and infested crops with accurate labeling. Subsequently, image preprocessing techniques are applied to standardize their attributes, including size, format, and quality. This involves common practices such as resizing, normalization, and augmentation to enhance dataset diversity and improve model generalization. A CNN architecture suitable for the task is then selected and initialized, either with random weights or leveraging pre-trained weights from models like ImageNet through transfer learning. The dataset is split into training, validation, and test sets to facilitate model training, hyperparameter tuning, and performance evaluation, respectively. Selection of an appropriate loss function and optimization algorithm is crucial, alongside hyperparameter tuning to optimize model performance, often via techniques like grid search or random search. The training loop involves iteratively feeding batches of training data through the CNN model, computing loss, and updating weights using backpropagation and the chosen optimization algorithm. Regular validation checks monitor for overfitting, with early stopping implemented based on validation performance to prevent overfitting and conserve computational resources. Upon completion of training, the model's performance is assessed on the test set using various evaluation metrics. Optional fine-tuning may be performed by retraining the model on the entire dataset to further enhance performance. Ultimately, the trained model is deployed for real-world applications, necessitating meticulous monitoring, hyperparameter adjustments, and regularization techniques to ensure robustness. Comprehensive documentation of the training process, including hyperparameters and evaluation results, is imperative for reproducibility and future reference.

5. 6 .1 EPOCH IN NEURAL NETWORKS

An epoch refers to a complete cycle of training the neural network, encompassing the utilization of all training data exactly once during that cycle. In an epoch, we use all of the data exactly once. A forward pass and a backward pass together are counted as one pass.

Table 1 shows the Epoch cycle of training the model. Table I shows the representation of epoch, time versus accuracy, and loss. In this work, we took 30 iterations based on the dataset. Using validation data, we validated accuracy and loss during training.



Epoch	Time	Accuracy	Loss
Epoch 1/30	1s 83ms /step	0.5758	0.6944
Epoch 2/30	0s 61ms /step	0.5455	1.5762
Epoch 3/30	0s 157ms /step	0.5000	0.6832
Epoch 4/30	0s 71ms/step	0.7576	0.6699
Epoch 5/30	0s 61ms/step	0.5000	0.7161
Epoch 6/30	0s 67ms/step	0.5000	0.6765
Epoch 7/30	0s 145ms/step	0.5000	0.6802
Epoch 8/30	0s 62ms/step	0.6364	0.6408
Epoch 9/30	0s 127ms/step	0.6515	0.6434
Epoch 10/30	0s 61ms/step	0.6061	0.6261
Epoch 11/30	0s 60ms/step	0.8788	0.5207
Epoch 12/30	0s 60ms/step	0.7273	0.8979
Epoch 13/30	0s 60ms/step	0.7273	0.5107
Epoch 14/30	0s 128ms/step	0.7879	0.4493
Epoch 15/30	0s 61ms/step	0.9545	0.2565
Epoch 16/30	0s 60ms/step	0.6970	0.5606
Epoch 17/30	0s 60ms/step	0.8636	0.2831
Epoch 18/30	0s 126ms/step	0.9394	0.1811
Epoch 19/30	0s 130ms/step	0.9091	0.2305
Epoch 20/30	0s 59ms/step	0.9848	0.1000
Epoch 21/30	0s 61ms/step	0.8939	0.2087
Epoch 22/30	0s 59ms/step	0.9697	0.0969
Epoch 23/30	0s 136ms/step	0.9848	0.0598
Epoch 24/30	0s 60ms/step	0.9848	0.0217
Epoch 25/30	0s 126ms/step	1.000	0.0124
Epoch 26/30	0s 62ms/step	1.000	0.0080
Epoch 27/30	0s 61ms/step	1.000	0.0058
Epoch 28/30	0s 61ms/step	1.000	0.0026
Epoch 29/30	0s 130ms/step	1.000	0.0021
Epoch 30/30	0s 62ms/step	1.000	0.0014

Table 1 Epoch with Accuracy and Loss

Initially, the model's weights are initialized, either randomly or utilizing pre- trained weights, particularly in scenarios involving transfer learning. Subsequently, the training dataset, comprising labeled images of both healthy and infested crops, is loaded into memory. During each epoch, the CNN traverses the entire training dataset in mini-batches. Within this training loop, the input images undergo a forward pass through the CNN model, yielding predictions. Subsequently, the loss function is computed, quantifying the disparity between predicted outputs and actual labels. Backpropagation is then employed to compute gradients of the loss concerning the model's parameters (weights), facilitating parameter adjustment by the optimizer to minimize the loss. Upon processing all mini-batches in the training dataset, one epoch concludes. Optionally, after each epoch, or at regular intervals, the model's performance is assessed on a distinct validation dataset. This validation step aids in monitoring the model's performance and detecting potential overfitting phenomena.

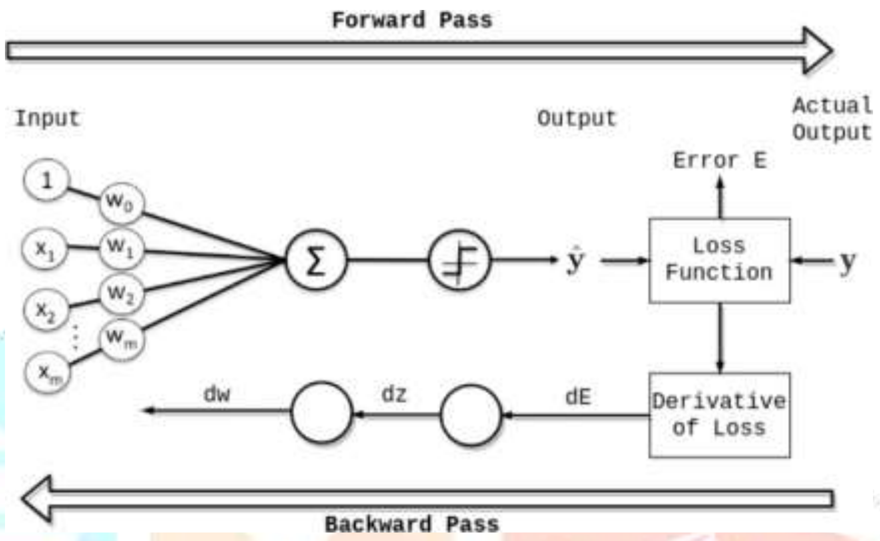


Fig 5.4 Epoch for training in one cycle

Fig 5.4 shows the epoch process iterates for a specified number of epochs, with each iteration updating the model's parameters based on gradients calculated from a distinct subset of the training dataset. Ultimately, training concludes when the predetermined number of epochs is achieved, or when early stopping criteria are met, such as no discernible improvement in validation performance. The number of epochs serves as a critical hyperparameter necessitating tuning during the training process. Insufficient epochs may result in underfitting, where the model fails to capture underlying data patterns, while excessive epochs may lead to overfitting, causing the model to overly memorize the training data and exhibit poor generalization on unseen data.

In essence, the epoch process in CNNs involves cyclically training the model on the entire training dataset for a specified number of passes, thereby optimizing the model's parameters to minimize the loss function and enhance performance in detecting pest infestation. In our project phase, we took 30 iterations based on the dataset. Using Validation data to monitor validate accuracy and validate loss during training.

5. 7 MODEL EVALUATION

After the model has been trained, it must be evaluated on a test set to determine its accuracy and robustness. This may involve measuring metrics such as accuracy, precision, recall, and F1 score, as well as visualizing the model's predictions to identify any patterns or weaknesses.

The evaluation of a Convolutional Neural Network (CNN) model for pest infestation detection is a critical step in assessing its effectiveness. Initially, a distinct test dataset is prepared, comprising labeled images of healthy and infested crops, ensuring unbiased evaluation on unseen data. Subsequently, the test dataset is inputted into the trained CNN model, generating predictions for each image by traversing through the CNN layers. These predictions are then compared with the ground truth labels to interpret the model's accuracy in identifying pest infestation. Various performance metrics are calculated to quantify the model's effectiveness, including accuracy, precision, recall, F1-score, and the confusion matrix. These metrics offer insights into the model's classification performance and aid in understanding its strengths and weaknesses.

Optional visualization techniques, such as activation maps, can provide further insights into the model's decision-making process. Additionally, based on the evaluation results, the model may undergo refinement or optimization through adjustments to hyperparameters, architecture, or retraining with additional data.

Comprehensive documentation of the evaluation process, including metrics, analysis findings, and any model refinements, is crucial for future reference and reproducibility. Through rigorous evaluation, practitioners can confidently assess the CNN model's capability in detecting pest infestation in crops, enabling informed decisions regarding its deployment in real-world scenarios.

5. 8 MODEL OPTIMIZATION

If the model's performance is not satisfactory, various optimization techniques can be used to improve it. This may include fine-tuning the architecture, adding regularization, or using data augmentation to generate additional training data.
Optimizing a Convolutional Neural Network (CNN) for pest infestation detection involves implementing various strategies to enhance the model's performance, efficiency, and ability to generalize. These optimization techniques encompass several key aspects.

Firstly, data augmentation methods such as rotation, flipping, and cropping are employed to enrich the training dataset, promoting better model generalization by exposing it to a wider range of variations in image appearance.Normalization of input pixel values to a standardized range aids in stabilizing and expediting the training process, ensuring uniformity in feature scales across images.Regularization techniques like dropout and L2 regularization help mitigate overfitting by introducing noise or penalizing large weights, thus enhancing the model's ability to generalize to unseen data. Dynamic adjustment of the learning rate through learning rate scheduling techniques prevents the model from getting stuck in local minima, facilitating faster convergence and improved performance.Leveraging transfer learning from pre-trained CNN models, particularly those trained on extensive datasets like ImageNet, accelerates training and boosts performance, particularly beneficial when working with limited training data. Experimenting with different CNN architectures, layer configurations, and hyperparameters allows for the identification of the most effective model architecture for the specific task, potentially reducing computational complexity without sacrificing performance.

Utilizing specialized hardware accelerators like GPUs or TPUs accelerates training, enabling quicker experimentation and iteration of model designs.
Ensemble learning techniques, which involve combining predictions from multiple CNN models, capitalize on diverse model architectures and training strategies to achieve enhanced performance compared to individual models.Cross-validation ensures robustness and generalization ability of the model by assessing its performance across different subsets of the data.Fine-tuning hyperparameters through methods like grid search or random search enables the discovery of optimal settings for the model, improving its overall effectiveness.

By systematically integrating these optimization strategies, practitioners can develop CNN models finely tuned for pest infestation detection tasks, ultimately achieving superior performance, efficiency, and generalization capability.Model optimization in a Convolutional Neural Network (CNN) involves several techniques and strategies aimed at improving the performance, efficiency, and generalization ability of the model. Here's how model optimization occurs in a CNN Adjusting hyperparameters such as learning rate, batch size, dropout rate, optimizer choice, and network architecture can significantly impact the performance of the CNN. Hyperparameter tuning involves experimenting with different combinations of hyperparameters and evaluating their effects on the model's performance using validation data. Regularization techniques such as L1 and L2 regularization, dropout, and batch normalization are used to prevent overfitting and improve the generalization ability of the model. Regularization helps prevent the model from memorizing the training data and encourages it to learn more robust features.

Choosing the right optimization algorithm and its parameters can have a significant impact on training efficiency and model performance. Common optimizers used in CNNs include Adam, RMSprop, and SGD (Stochastic Gradient Descent). Experimentation with different optimizers and learning rates can help find the optimal configuration for the CNN. Adjusting the learning rate during training can help improve convergence and prevent the model from getting stuck in local minima. Learning rate scheduling techniques such as exponential decay, step decay, or adaptive learning rates (e.g., Cyclical Learning Rates) can be used to gradually decrease the learning rate as training progresses.

Increasing the diversity of the training data through techniques such as rotation, flipping, scaling, and translation can help improve the generalization ability of the model. Data augmentation introduces variability into the training data, making the model more robust to variations in the input data. Leveraging pre-trained models trained on large-scale datasets (e.g., ImageNet) as a starting point can help accelerate training and improve performance, especially when the available training data is limited. Transfer learning involves fine-tuning the pre-trained model's parameters on a smaller dataset or using the pre-trained model as a feature extractor and training a new classifier on top. Experimenting with different CNN architectures (e.g., number of layers, filter sizes, number of filters per layer) can help find the optimal architecture for the given task. Techniques such as model ensembling, network pruning, and architecture search algorithms (e.g., Neural Architecture Search) can also be used to optimize the model architecture.

Continuously monitoring the training process, analyzing performance metrics, and debugging any issues that arise during training can help identify potential problems early and make necessary adjustments to improve model performance.By systematically applying these optimization techniques and strategies, practitioners can develop CNN models that achieve better performance, efficiency, and generalization ability for various tasks, including image classification, object detection, and segmentation.

5. 9 DEPLOYMENT

Once the model has been trained and optimized, it can be deployed for real-time use. This may involve integrating the model into a software application or hardware device, such as a camera system. The model should be designed and optimized for efficient real-time performance, as speed is critical for this application.

Deployment of a Convolutional Neural Network (CNN) for pest detection involves integrating the trained model into a real-world application or system where it can be used to perform pest detection tasks. Here's how deployment occurs in a CNN for pest detection. After training and optimizing the CNN model, it needs to be exported in a format suitable for deployment. Common formats include TensorFlow Saved Model, TensorFlow Lite, ONNX (Open Neural Network Exchange), or a custom format depending on the deployment environment and framework requirements.

The exported model is integrated into the application or system where pest detection is required. This may involve writing code to load the model, preprocess input data, perform inference, and post-process the results. Integration can be done using programming languages such as Python, Java, or C++, depending on the application's requirements. The deployed CNN model expects input data in a specific format. For image-based pest detection, input images need to be preprocessed to match the format expected by the model. This may involve resizing, normalization, and converting the images to the appropriate color space (e.g., RGB). During inference, the deployed CNN model processes input data (e.g., images) and generates predictions for pest presence or absence. Inference can be performed using dedicated hardware accelerators (e.g., GPUs, TPUs) or on general-purpose CPUs, depending on the performance and latency requirements of the application.

The model's predictions are typically presented in a human-readable format, such as a graphical interface or textual output. For pest detection, the output may include information about the detected pests, their location, and confidence scores associated with the predictions. In some applications, such as smart agriculture or pest monitoring systems, CNN models may be deployed on edge devices or IoT (Internet of Things) devices. Integration with IoT devices involves considerations such as resource constraints, power consumption, and communication protocols.

Once deployed, the CNN model should be monitored regularly to ensure it continues to perform effectively in real-world scenarios. Monitoring may involve tracking performance metrics, detecting drift in input data distribution, and updating the model periodically with new training data or retraining it if necessary. Considerations related to security and privacy should be addressed during deployment to protect sensitive data and prevent unauthorized access to the deployed model. This may involve implementing encryption, access controls, and compliance with relevant regulations (e.g., GDPR, HIPAA).By following these steps, the CNN model for pest detection can be successfully deployed and integrated into real-world applications and systems, helping to address pest management challenges in agriculture and other domains.

5. 10 MONITORING AND MAINTENANCE

After deployment, the model should be monitored and maintained to ensure it continues to perform well over time. This may involve periodic retraining or fine- tuning, as well as monitoring the performance metrics to detect any degradation or drift in accuracy. The process of Smart agriculture using CNNs involves several distinct steps, each with its own set of modules and components. The success of the overall process depends on careful attention to each step, including data collection, preprocessing, model design, training, evaluation, optimization, deployment, and monitoring and maintenance.

Monitoring and maintenance are essential aspects of deploying a Convolutional Neural Network (CNN) for pest detection to ensure continued performance and reliability over time. Here's how monitoring and maintenance can be carried out. Continuously monitor key performance metrics of the CNN model, such as accuracy, precision, recall, and F1-score, to assess its performance on a regular basis. These metrics provide insights into how well the model is detecting pests and can help identify any degradation in performance. Monitor the distribution of input data to detect any drift or changes in the data distribution over time. Data drift can occur due to changes in environmental conditions, seasonal variations, or other factors, which may impact the model's performance. Implement mechanisms to detect and adapt to these changes to maintain the model's effectiveness.

Periodically reevaluate the CNN model's performance using a separate validation or testing dataset to ensure that it continues to generalize well to new data. This evaluation helps identify any potential degradation in performance and guides decisions regarding model updates or retraining. If performance metrics indicate a decline in model performance or if significant changes in the input data distribution are detected, consider updating or retraining the CNN model. This may involve collecting additional training data, adjusting hyperparameters, or fine-tuning the model architecture to address new challenges or environmental conditions. Ensure that the deployed CNN model complies with relevant security and privacy regulations to protect sensitive data and maintain user trust. Implement security measures such as encryption, access controls, and data anonymization to safeguard against potential threats and unauthorized access. Monitor the CNN model for any bugs, errors, or inefficiencies that may arise during deployment. Address any issues promptly through bug fixes, code optimizations, or updates to ensure the continued reliability and efficiency of the model.

Maintain comprehensive documentation of the deployed CNN model, including its architecture, training process, performance metrics, and any updates or changes made over time. This documentation facilitates knowledge transfer and helps ensure continuity in model maintenance and management, especially in cases where multiple stakeholders are involved. Maintain open communication with stakeholders, including end-users, domain experts, and technical teams, to gather feedback, address concerns, and prioritize maintenance efforts effectively. Regularly solicit feedback from stakeholders to identify areas for improvement and guide future development and maintenance activities. By implementing a systematic approach to monitoring and maintenance, stakeholders can ensure the ongoing effectiveness, reliability, and security of the deployed CNN model for pest detection, thereby contributing to successful pest management efforts in agriculture and other domains.

6.1 CONCLUSION

The proposed work brings together advanced technologies to address a critical challenge in agriculture. Stem borer pests pose a significant threat to crop yields, and early detection is paramount for effective pest management. The utilization of Convolutional Neural Networks serves as the cornerstone of this work, providing a robust and efficient means of identifying subtle patterns indicative of pest infestation in crops. Throughout this work, we emphasized the importance of a diverse and representative dataset, including images of infested and healthy crops, various lighting conditions, different crop varieties, and multiple growth stages. This comprehensive dataset forms the foundation for training a CNN architecture optimized for accurate pest detection.

6.2 FUTURE ENHANCEMENT

Further research will evaluate our model in an environment where these future iterations are likely to enhance the accuracy and efficiency of pest identification, enabling real-time monitoring and early intervention in agricultural settings. The integration of CNNs with emerging technologies such as Internet of Things (IoT) devices, drones, and satellite imagery will further amplify their effectiveness, providing farmers with a comprehensive and automated solution for pest management. The scalability of our approach across various crops and its alignment with precision agriculture practices will make it a pivotal tool in sustaining global food security by minimizing crop losses and optimizing resource utilization in a rapidly evolving agricultural landscape.

APPENDIX-1 SAMPLE CODE

Pest detection

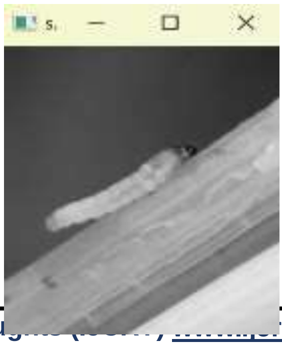
```
batch_size = 32

# All images will be rescaled by 1./255 train_datagen = ImageDataGenerator(rescale=1/255)


# Flow training images in batches using train_datagen generator train_generator =

train_datagen.flow_from_directory(

'dataset',  # This is the source directory for training images target_size=(200, 200),  # All images will be

resized to 200 x 200 batch_size=batch_size,

# Specify the classes explicitly classes=['incertulas', 'yelow'],

# Since we use categorical_crossentropy loss, we need categorical labels class_mode='categorical')


import tensorflow as tf


model = Sequential([# Note the input shape is the desired size of the image 200x 200 with 3 bytes color #
The first convolution
Conv2D(16, (3, 3), activation='relu', input_shape=(200, 200, 3)),

MaxPooling2D(2, 2),

# The second convolution Conv2D(32, (3, 3), activation='relu'),

MaxPooling2D(2, 2),

# The third convolution

Conv2D(64, (3, 3), activation='relu'),

MaxPooling2D(2, 2),

# The fourth convolution Conv2D(64, (3, 3), activation='relu'),

MaxPooling2D(2, 2),  # The fifth convolution

Conv2D(64, (3, 3), activation='relu'),

MaxPooling2D(2, 2),

# Flatten the results to feed into a dense layer Flatten(),

# 128 neurons in the fully-connected layer Dense(128, activation='relu'),

# 4 output neurons for 4 classes with softmax activation
    Dense(2, activation='softmax')]) model.summary()

from tensorflow.keras.optimizers import RMSprop


odel.compile(loss='categorical_crossentropy', optimizer=RMSprop(learning_rate=0.001), metrics=['acc'])

# Define the number of epochs n_epochs = 30


# Use 'validation_data' to monitor validation accuracy and loss during training history = model.fit(

train_generator, steps_per_epoch=len(train_generator), epochs=n_epochs,

verbose=1)


# Plot training accuracy plt.plot(history.history['acc']) plt.title('Model Training Accuracy')

plt.ylabel('Accuracy') plt.xlabel('Epoch')

plt.legend(['Train'], loc='upper left') plt.show()
```

```
    # Plot training loss plt.plot(history.history['loss']) plt.title('Model Training Loss') plt.ylabel('Loss')

    plt.xlabel('Epoch')

    plt.legend(['Train'], loc='upper left') plt.show()


    # Save the model model.save('model.h5')
```

Pest Prediction

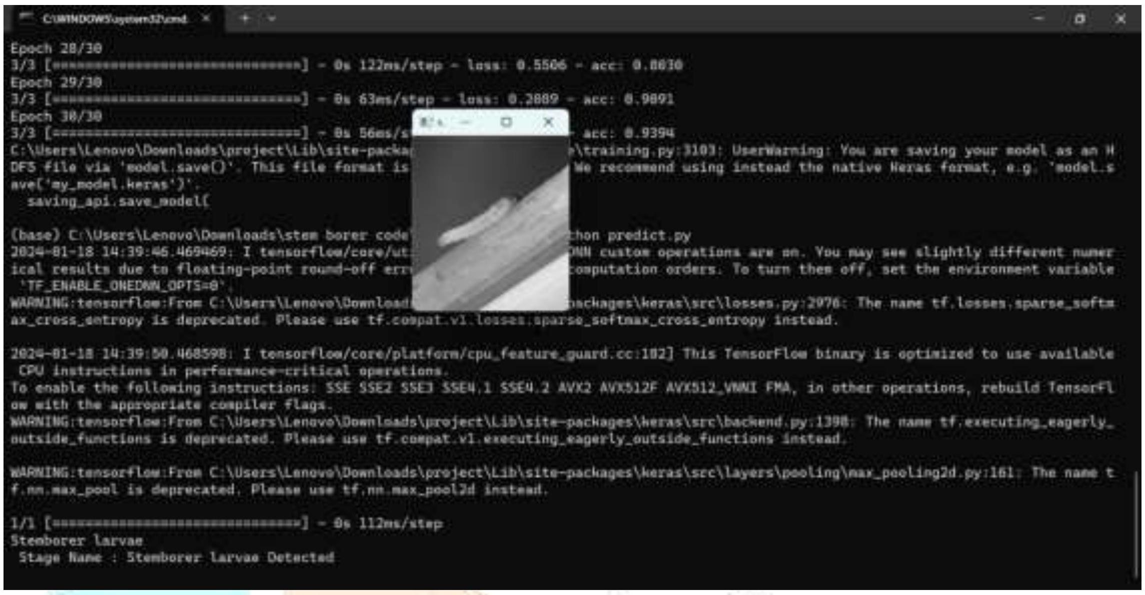
```
    import tensorflow as tf

    classifierLoad = tf.keras.models.load_model('model.h5') # load the model here import pandas as pd

    import numpy as np import cv2

    from PIL import Image, ImageFont, ImageDraw from keras.preprocessing import image


    test_image1 = cv2.imread('3.jpg',0)

    test_image = image.load_img('3.jpg', target_size = (200,200))          # load the sample image

    here

    #test_image = image.img_to_array(test_image) test_image = np.expand_dims(test_image, axis=0) result =

    classifierLoad.predict(test_image)

if result[0][0] == 1: print("Stemborer larvae ")

    print(" Stage Name : Stemborer larvae Detected ") cv2.imshow('sampleimage',test_image1) cv2.waitKey(0)


elif result[0][1] == 1: print(" Yellow stem  ")

    print(" Stage Name : Yellow stem borer detected ") cv2.imshow('sampleimage',test_image1) cv2.waitKey(0)
```

APPENDEX-2 SCREENSHOTS



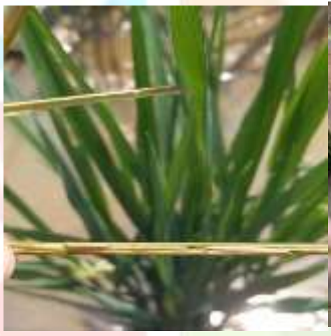
(i) (ii) (iii) Fig A2. 1 (i-ii) and (iv-v)

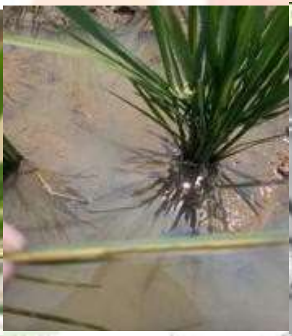
Sample Dataset

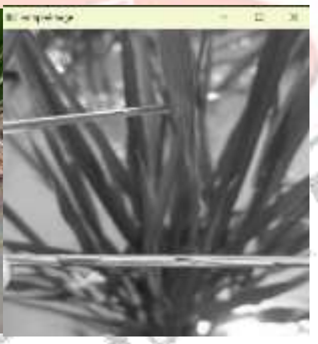


```
Epoch 28/30
3/3 [==============================] - 0s 122ms/step - loss: 0.5506 - acc: 0.8830
Epoch 29/30
3/3 [==============================] - 0s 63ms/step - loss: 0.2889 - acc: 0.9891
Epoch 30/30
3/3 [==============================] - 0s 56ms/s                    - acc: 0.9394
C:\Users\Lenovo\Downloads\project\Lib\site-packa                    e\training.py:3103: UserWarning: You are saving your model as an H
DF5 file via 'model.save()'. This file format is                    We recommend using instead the native Keras format, e.g. 'model.s
ave('my_model.keras')'.
  saving_api.save_model(

(base) C:\Users\Lenovo\Downloads\stem borer code                    thon predict.py
2024-01-18 14:39:46.469469: I tensorflow/core/ut                    DNN custom operations are on. You may see slightly different numer
ical results due to floating-point round-off err                    computation orders. To turn them off, set the environment variable
 `TF_ENABLE_ONEDNN_OPTS=0`.
WARNING:tensorflow:From C:\Users\Lenovo\Download                    packages\keras\src\losses.py:2976: The name tf.losses.sparse_softm
ax_cross_entropy is deprecated. Please use tf.compat.v1.losses.sparse_softmax_cross_entropy instead.

2024-01-18 14:39:50.468598: I tensorflow/core/platform/cpu_feature_guard.cc:182] This TensorFlow binary is optimized to use available
 CPU instructions in performance-critical operations.
To enable the following instructions: SSE SSE2 SSE3 SSE4.1 SSE4.2 AVX2 AVX512F AVX512_VNNI FMA, in other operations, rebuild TensorFl
ow with the appropriate compiler flags.
WARNING:tensorflow:From C:\Users\Lenovo\Downloads\project\Lib\site-packages\keras\src\backend.py:1398: The name tf.executing_eagerly_
outside_functions is deprecated. Please use tf.compat.v1.executing_eagerly_outside_functions instead.

WARNING:tensorflow:From C:\Users\Lenovo\Downloads\project\Lib\site-packages\keras\src\layers\pooling\max_pooling2d.py:161: The name t
f.nn.max_pool is deprecated. Please use tf.nn.max_pool2d instead.

1/1 [==============================] - 0s 112ms/step
Stemborer larvae
 Stage Name : Stemborer larvae Detected
```

Fig A2. 2 Stem Borer Larvae Detected

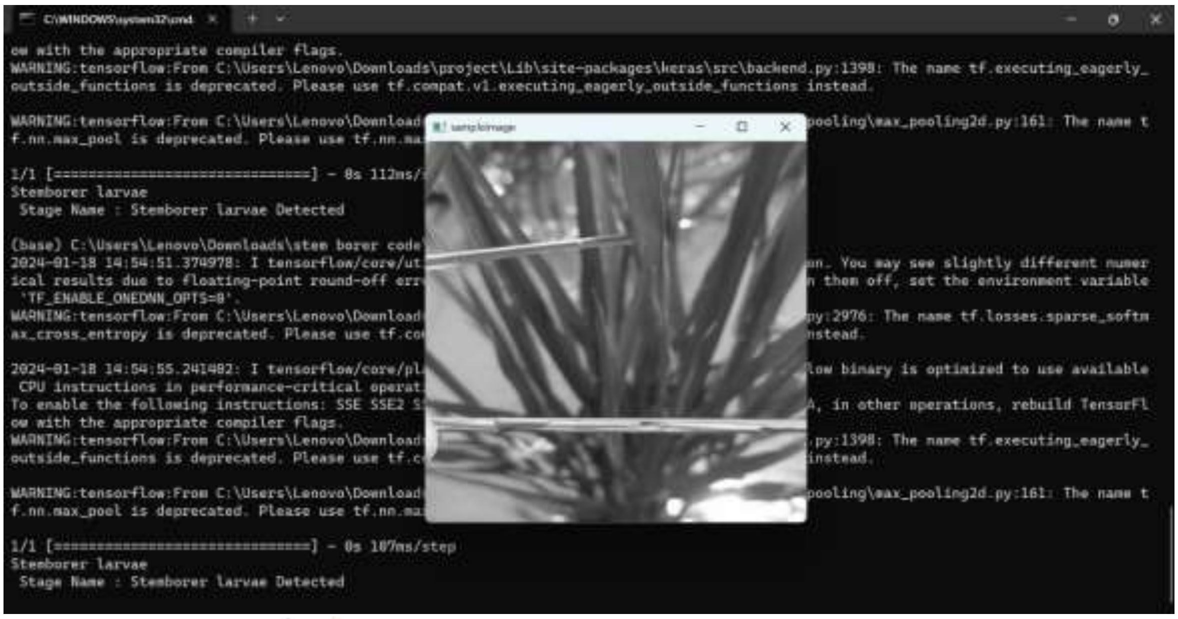






(iv) (v) (vi) Fig A2. 3 (iii &vi) Image

with YSB detected



```
ow with the appropriate compiler flags.
WARNING:tensorflow:From C:\Users\Lenovo\Downloads\project\Lib\site-packages\keras\src\backend.py:1398: The name tf.executing_eagerly_
outside_functions is deprecated. Please use tf.compat.v1.executing_eagerly_outside_functions instead.

WARNING:tensorflow:From C:\Users\Lenovo\Download                                  pooling\max_pooling2d.py:161: The name t
f.nn.max_pool is deprecated. Please use tf.nn.ma

1/1 [==============================] - 0s 112ms/
Stemborer larvae
 Stage Name : Stemborer larvae Detected

(base) C:\Users\Lenovo\Downloads\stem borer code
2024-01-18 14:54:51.374978: I tensorflow/core/ut                                  nn. You may see slightly different numer
ical results due to floating-point round-off err                                  n them off, set the environment variable
 `TF_ENABLE_ONEDNN_OPTS=0`.
WARNING:tensorflow:From C:\Users\Lenovo\Download                                  py:2976: The name tf.losses.sparse_softm
ax_cross_entropy is deprecated. Please use tf.co                                  nstead.

2024-01-18 14:54:55.241482: I tensorflow/core/pl                                  low binary is optimized to use available
 CPU instructions in performance-critical operat
To enable the following instructions: SSE SSE2 S                                  A, in other operations, rebuild TensorFl
ow with the appropriate compiler flags.
WARNING:tensorflow:From C:\Users\Lenovo\Download                                  .py:1398: The name tf.executing_eagerly_
outside_functions is deprecated. Please use tf.c                                  instead.

WARNING:tensorflow:From C:\Users\Lenovo\Download                                  pooling\max_pooling2d.py:161: The name t
f.nn.max_pool is deprecated. Please use tf.nn.ma

1/1 [==============================] - 0s 107ms/step
Stemborer larvae
 Stage Name : Stemborer larvae Detected
```

Fig A2. 4 Yellow Stem Borer detected

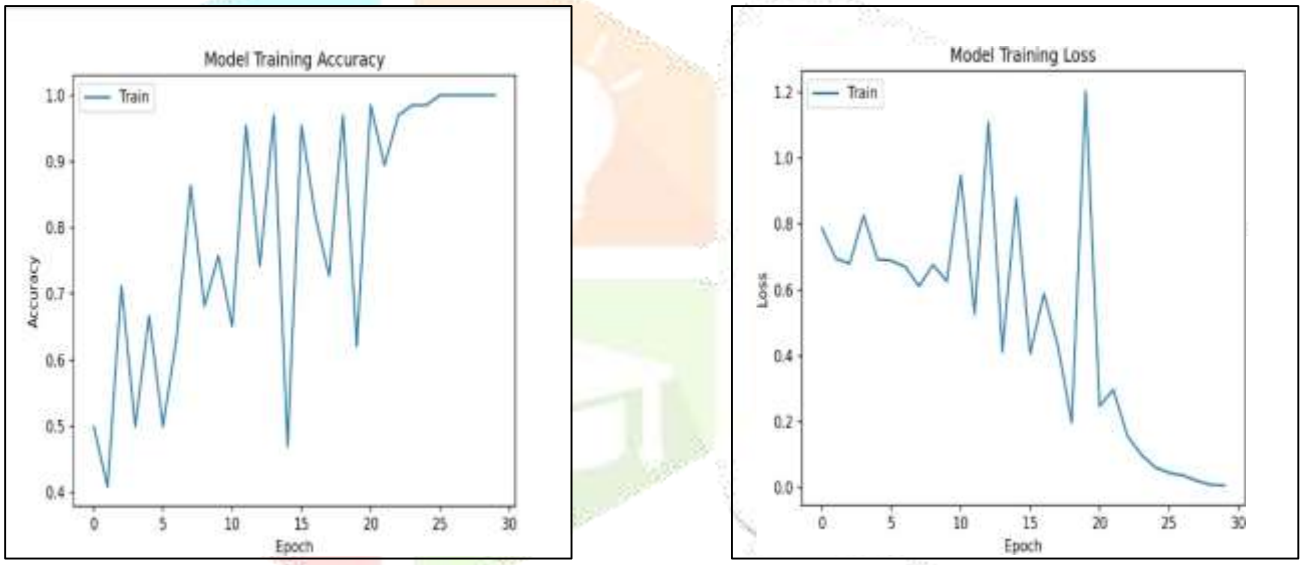


Fig A2. 5(i) Model training Accuracy Vs Epoch (ii) Model training Loss Vs Epoch

REFERENCES:

1. Yangyang Fan, Tao Wang, Zhengjun Qiu, Jiyu Peng, Chu Zhang and Yong He, Fast Detection of Striped Stem-Borer (Chilo suppressalis Walker) Infested Rice Seedling Based on Visible/Near-Infrared Hyperspectral Imaging System, Sensors 2017, 17, 2470 2017

2. Roshan Zameer Ahmed, Rajashekhar C. Biradar. Energy-aware Routing in WSN for pest Detection in Coffee plantations. ICACCI, 2016.

3. SanaBhandarkar,RuthPrasad,Vaibhav AgarwalRHebbar,D.Uma,Y.B.Venkata Reddy Y.Raghuramulu and K.Ganesha Raj.Deep learning and Stastical models for Detection of white stem borer

Disease in Arabica Coffee.ISPRS,2019.

4. Chuanqi Xie and Yong he, Spectrum and Image Texture Features Analysis for Early Blight Disease Detection on Eggplant Leaves, sensors 2016, 16(5) 676

5. Wei Li, Tengfei Zhu, Xiaoyu Li, Jianzhang Dong, and Jun Liu, Recommending Advanced Deep Learning Models for Efficient Insect Pest Detection Agriculture- 2022, 12(7), 1065

6. Fernando Vanegas 1,*, Dmitry Bratanov 1, Kevin Powell 2,3, John Weiss 3,4 and Felipe Gonzalez, A Novel Methodology for Improving Plant Pest Surveillance in Vineyards and Crops Using UAV-Based Hyperspectral and Spatial Data, Sensors 2018, 18(1), 260 - 2018

7. Abdelaaziz Hessane, Ahmed El Youssefi, Yousef Farhaoui, Badraddine Aghoutane, and Fatima Amounas. A Machine Learning Based Framework for a Stage-Wise Classification of Date Palm White Scale Disease.Big Data Mining and Analytics,2023.

8. Loris Nannia, Gianluca Maguoloa, Fabio Pancinoa.Insect pest image detection and recognition based on bio-inspired methods. Science Direct, 2020.

9. Farooq Ali; Huma Qayyum; Muhammad Javed Iqbal, A Lightweight Deep Learning Framework for Crop Pest Detection and Classification – 2023

10. Yuhua Li, Zhihui Luo, Fengjie Wang and Yingxu Wang, Hyperspectral Leaf Image-Based Cucumber DiseaseRecognition Using the Extended CollaborativeRepresentation Model, Sensors 2020, 20(14), 4045- 2020