Deep Learning Methods for Disease Identification of Cotton Plants

Author

Sajeel Fasihi

Fall 2019-MS(RIME)-00000319310

Supervisor

Dr. Karam Dad Kallu

MS ROBOTICS & INTELLIGENT MACHINE ENGINEERING

DEPARTMENT OF ROBOTICS & AI

SCHOOL OF MECHANICAL & MANUFACTURING ENGINEERING

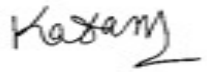
National University of Sciences and Technology (NUST)

Islamabad, Pakistan

(August 2023)

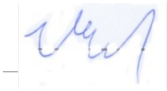
THESIS ACCEPTANCE CERTIFICATE

Certified that final copy of MS/MPhil thesis written by **Regn No. 00000319310 Sajeel Fasihi** of **School of Mechanical & Manufacturing Engineering (SMME) (SMME)** has been vetted by undersigned, found complete in all respects as per NUST Statues/Regulations, is free of plagiarism, errors, and mistakes and is accepted as partial fulfillment for award of MS/MPhil degree. It is further certified that necessary amendments as pointed out by GEC members of the scholar have also been incorporated in the said thesis titled. **Deep Learning Methods for Disease Identification of Cotton Plants**

Signature: 

Name (Supervisor): Karam Dad

Date: 23 - Aug - 2023

Signature (HOD): 

Date: 23 - Aug - 2023

Signature (DEAN): 

Date: 23 - Aug - 2023

Form TH-4

National University of Sciences & Technology (NUST)

MASTER'S THESIS WORK

We hereby recommend that the dissertation prepared under our supervision by: Sajeel Fasihi (00000319310)
Titled: Deep Learning Methods for Disease Identification of Cotton Plants be accepted in partial fulfillment of the requirements for the award of MS in Robotics & Intelligent Machine Engineering degree.

Examination Committee Members

1. Name: Hasan Ali Khattak Signature:

2. Name: Zuhair Zafar Signature:

3. Name: Hasan Sajid Signature:

Supervisor: Karam Dad Signature:

Date: 23 - Aug - 2023

Head of Department 23 - Aug - 2023

Date

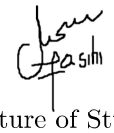
COUNTERSINGED

23 - Aug - 2023

Date Dean/Principal

Declaration

I certify that this research work titled "Deep Learning Methods for Disease Identification of Cotton Plants" is my own work. The work has not been presented elsewhere for assessment. The material that has been used from other sources it has been properly acknowledged / referred.



Signature of Student

Sajeel Fasihi

319310

Proposed Certificate for Plagiarism

It is certified that PhD/M.Phil/MS Thesis Titled Deep Learning Methods for Disease Identification of Cotton Plants by Sajeel Fasihi has been examined by us. We undertake the follows:

a. Thesis has significant new work/knowledge as compared already published or are under consideration to be published elsewhere. No sentence, equation, diagram, table, paragraph or section has been copied verbatim from previous work unless it is placed under quotation marks and duly referenced.
b. The work presented is original and own work of the author (i.e. there is no plagiarism). No ideas, processes, results or words of others have been presented as Author own work.
c. There is no fabrication of data or results which have been compiled/analyzed.
d. There is no falsification by manipulating research materials, equipment or processes, or changing or omitting data or results such that the research is not accurately represented in the research record.
e. The thesis has been checked using TURNITIN (copy of originality report attached) and found within limits as per HEC plagiarism Policy and instructions issued from time to time.

Name & Signature of Supervisor

DR KARAM DAD KALLU

Signature :

Dr. Karam Dad Kallu
Assistant Professor
Robotics and Intelligent Machine Engineering
School of Mechanical and
Manufacturing Engineering
(SMME) NUST H-12 Islamabad

Dr. Karam Dad Kallu
Assistant Professor
Robotics and Intelligent Machine Engineering
School of Mechanical and
Manufacturing Engineering
(SMME) NUST H-12 Islamabad

Thesis

ORIGINALITY REPORT

10%	7%	5%	4%
SIMILARITY INDEX	INTERNET SOURCES	PUBLICATIONS	STUDENT PAPERS

PRIMARY SOURCES

1	Submitted to Cranfield University Student Paper	1%
2	Submitted to Liverpool John Moores University Student Paper	1%
3	www.mdpi.com Internet Source	1%
4	www.arxiv-vanity.com Internet Source	<1%
5	Submitted to University of Leeds Student Paper	<1%
6	www.ijraset.com Internet Source	<1%
7	Submitted to University of Auckland Student Paper	<1%
8	www.hindawi.com Internet Source	<1%
9	deepai.org Internet Source	<1%

Copyright Statement

Acknowledgments

In the name of Allah, the most merciful and benevolent. All praises to Allah Almighty for giving me the fortitude to finish my thesis. I owe my supervisor Dr. Karam Dad Kallu and my co-supervisor Dr. Hassan Ali Khattak the utmost gratitude.Due to their persistent concern for the project, as well as wise counsel, encouragement, and support, would have left this effort futile. I also want to express my gratitude to my committee members Dr. Zuhair Zafar and Dr Hassan Sajid for their insightful suggestions throughout this investigation. I am so appreciative of my parents' love and unwavering support during this journey.

Dedication

I would love to dedicate my thesis to my beloved parents, teachers, my family and my supervisor Dr. Karam dad Kallu who belived in my skills and abilities and guided me throughout my thesis. I would also like to thank my co-supervisor Dr Hasan Ali Khattak for supporting me and guiding. Lastly i would thank Salman Hassan and Arslan who helped and motivated me during entire MS degree.

Abstract

Cotton is a vital cash crop, contributing significantly to the global textile industry and the livelihoods of millions of farmers worldwide. However, diseases such as bacterial blight, leaf curl virus, and whitefly infestations pose a severe threat to cotton production and quality. Timely detection and accurate identification of these diseases are crucial for implementing effective control measures and ensuring crop health by exploring multiple state-of-the-art deep learning models, including CNNs and transformers. The research utilizes a diverse dataset of cotton plant images, encompassing healthy and diseased leaves, to train and fine-tune the deep learning models and Vision transformers. Additionally, we will focus on evaluating the models' capability to detect varying intensities of whitefly infestations, which is critical for assessing disease severity and implementing appropriate control strategies. The models were cross-validated and regularized to improve the models working. This study has the potential to contribute significantly to the field of computer vision, particularly for cotton disease detection.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

Agriculture plays a key role in meeting the escalating food demands of a growing global population, while also serving as a substantial source of revenue. However, this sector confronts significant challenges, chiefly due to the rising demand for food and the prevalence of crop diseases. Such diseases can substantially undermine agricultural productivity, resulting in notable crop losses, diminished food quality, and consequently, a reduced food supply.

1.1 Background

In particular, cotton, a leading cash crop worldwide, has substantial economic and social implications, influencing the global textile industry and providing livelihoods to countless farmers. However, cotton cultivation is not without its challenges, as diseases pose a primary threat to crop yield and quality. The detection and management of these diseases, therefore, is of utmost importance to ensure the sustainability of cotton production.

The prompt and accurate detection of crop diseases, coupled with well-planned management strategies, are vital to curbing the spread of these diseases and mitigating their deleterious effects on yields. In this context, the integration of cutting-edge technologies and data-driven methodologies can offer substantial benefits. Through these approaches, the agricultural sector can enhance its re-

silience against crop diseases, thereby ensuring sustainable production to cater to the needs of a growing population. The application of these novel strategies is particularly pertinent to cotton disease detection and management, given its economic and social importance worldwide.

1.2 Importance of Cotton Disease Detection

The detection of diseases in cotton is of paramount importance, as it directly impacts the health of the crop and, consequently, the quality and quantity of the yield. Cotton, being a significant cash crop in many regions worldwide, plays a vital role in economies, contributing to income generation, employment, and trade. Therefore, disease-inflicted losses can have severe repercussions not just at the individual farmer level, but also at regional and national scales[6]. Efficient and early detection of diseases allows for timely intervention, minimizing crop damage and preventing widespread outbreaks. This is particularly critical given the evolving disease landscape due to factors such as climate change and the increased movement of people and goods. Hence, enhancing our capabilities in cotton disease detection is not merely a matter of improving agricultural practices, but also of ensuring food security, promoting economic stability, and safeguarding the livelihoods of millions who rely on cotton cultivation.

1.3 Problem Statement

The need for effective and efficient detection of cotton diseases is crucial due to its significant impact on agricultural productivity. Traditional detection methods, while valuable, have their limitations in terms of labor intensity, timeliness, and precision. This research addresses this issue by exploring the application of deep learning models, specifically Convolutional Neural Networks (CNNs) and Vision Transformers (ViT small, Swin Transformer, and CCT), for automated cotton disease detection. Furthermore, it aims to evaluate these models' ability to classify the severity of damage caused by Whitefly on cotton leaves. The ultimate goal is

to contribute to an automated, precise, and early detection mechanism for cotton diseases, thereby aiding in their timely management and potentially reducing crop losses.

1.4 Proposed Solution

The proposed solution involves the implementation of different deep learning models—specifically, Convolutional Neural Networks (CNNs) and three types of Vision Transformers (ViT small, Swin Transformer, and CCT)—trained and tested on a curated dataset of cotton disease images.

The solution should also include the application of transfer learning and tuning strategies to optimize these models for the task. Importantly, it extends beyond disease detection, with an additional focus on classifying the intensity of cotton leaf damage caused by Whitefly.

By employing these advanced machine learning methods, the proposed solution seeks to provide a robust, automated, and high-performance tool for early and accurate cotton disease detection and severity classification. This could significantly aid timely disease management, potentially reducing crop losses and improving overall agricultural productivity.

1.5 Expected Outcome

The target of this research is to propose a comprehensive and efficient deep learning-based system capable of accurately detecting and classifying cotton diseases, as well as quantifying the severity of leaf damage caused by Whitefly. This system is anticipated to surpass traditional methods in terms of speed, accuracy, and scalability.

The rigorous evaluation of the proposed models is expected to identify the most effective architecture—whether it be a Convolutional Neural Network or one of the Vision Transformers—for cotton disease detection. In particular, we anticipate

that the application of transfer learning and tuning strategies will significantly enhance the performance of these models.Additionally, the research hopes to demonstrate the practicality of using such automated tools in a real-world agricultural context, potentially paving the way for their wider adoption in the industry. By enabling earlier and more precise detection of cotton diseases, the proposed system could substantially aid in timely disease management, thereby reducing crop losses and enhancing agricultural productivity. Overall, this research aims to make a significant contribution to the agricultural sector by revolutionizing the way cotton diseases are detected and managed.

1.6 Thesis Overview

This thesis presents a comprehensive study on the application of deep learning techniques for the detection and severity classification of cotton diseases. The research is situated at the intersection of agricultural health management and artificial intelligence, leveraging advanced machine learning methods to revolutionize traditional practices of disease detection in cotton cultivation.

The work begins with an exploration of Convolutional Neural Networks (CNNs) applied to a curated dataset of cotton disease images. The CNN models are fine-tuned using transfer learning and tuning techniques, thereby adapting the pre-trained models to the specific task of cotton disease detection.

The thesis then transitions to an examination of three distinct Vision Transformers – ViT small, Swin Transformer, and CCT. These transformer-based models are trained and tested on the same dataset, providing a comparative analysis of their performances against the earlier deployed CNN models.

In addition to disease detection, the research expands to investigate the classification of disease intensity, specifically focusing on the damage caused by Whitefly on cotton leaves. This additional layer of investigation provides a more nuanced understanding of disease impact, which could be crucial for effective disease management strategies.The final segment of the thesis involves a comparative analysis

of the performance of all the tested models. F1 score and accuracy metrics are employed to evaluate and compare model performances, resulting in a comprehensive understanding of the most effective deep learning architecture for this task.

Overall, the thesis provides a substantial contribution to the domain of cotton disease detection by proposing a robust, automated, and high-performing deep learning-based system. It not only showcases the potential of deep learning in revolutionizing traditional agricultural practices but also provides valuable insights for future research in this direction.

Chapter 2

Literature Review

2.1 Overview

Deep learning technologies are making significant inroads into various sectors, including agriculture, healthcare, transportation, and manufacturing. However, this thesis study will primarily concentrate on the application of deep learning techniques in identifying cotton diseases within the agricultural industry[7]. Cotton, being one of the most vital cash crops, plays a crucial role in the economies of many countries. The detection and management of diseases affecting cotton plants are paramount in ensuring a healthy yield. Traditional methods may not be fast or accurate enough, leading to significant losses. Utilizing deep learning algorithms can revolutionize the early detection of diseases, thereby minimizing damage and maximizing production. The application of deep learning in cotton disease identification is a promising frontier that can bring substantial advancements in agriculture, helping farmers and industries alike. This research aims to explore and develop deep learning models specifically tailored to the identification and management of various cotton diseases, ensuring a robust, efficient, and timely response[8].

2.2 Historical background

Research in the identification of cotton diseases has advanced significantly with the advent of deep learning techniques. Early efforts relied on traditional image processing, followed by machine learning methods, which required extensive feature engineering. The shift to deep learning, employing architectures like Convolutional Neural Networks (CNN), brought a more robust and automated approach, enhancing efficiency and accuracy[9].

However, initial deep learning models faced challenges such as the need for extensive labeled datasets and issues with overfitting. Subsequent research addressed these limitations through data augmentation, transfer learning, and optimized training algorithms. Recent advancements have explored hybrid models, combining various deep learning structures for even more precise disease identification in cotton[10].

The evolution from traditional methods to deep learning in cotton disease identification highlights an essential progression in agricultural technology, continually improving accuracy and efficiency in this critical field[11]

2.3 Work done in the literature

The implementation of a plant disease diagnosis system is vital for the agricultural industry to enhance crop production capacity by timely identifying and controlling diseases in their early stages. The proposed approach utilizes image processing techniques to analyze leaf samples and detect disease symptoms in their early stages. The research employs thresholding techniques to fragment the affected regions in the leaf images, precisely identifying the areas impacted by diseases.Furthermore, the study incorporates Gray-Level Co-occurrence Matrix (GLCM) features extracted from the diseased portions of the leaves. These GLCM features are used to categorize the type of disease affecting the cotton plants. By accurately identifying the specific diseases, farmers can make informed decisions to mitigate the spread and impact of these diseases, thereby protecting

their crops and enhancing productivity[12].

The research emphasizes the significance of using image processing techniques and GLCM-based analysis for cotton disease detection, as it provides a non-intrusive and rapid method for identifying the presence of diseases in cotton plants. Early detection enables timely interventions, such as targeted treatments or preventive measures, leading to better crop protection and increased yields[13]

The paper **"Detection and Classification of Cotton Leaf Diseases Using Faster R-CNN on Field Condition Images"** [14] presents a study on the use of convolutional neural networks to detect and classify diseases in cotton crops. The paper discusses the significance of cotton cultivation in Pakistan and other developing countries, and how the use of Faster R-CNN improves the accuracy of disease detection in cotton crops. The study was conducted on field condition images collected in Pakistan, and the results showed promising potential for improving crop management and increasing yields.

The article by**M. Zekiwos** [15] explains the gathering of images from the field using digital cameras and cellphones is an essential initial step in the process of building this model.Initially, images of cotton leaves with potential diseases and pests are acquired from the field using digital cameras and smartphones. Preprocessing techniques are then applied to prepare the acquired images for further analysis.Next, the preprocessed images are fed into a Convolutional Neural Network (CNN) for feature extraction. CNN extracts the most relevant features from the images to represent them effectively. These extracted features are then subjected to image analysis techniques to identify the best-suited representations for each image.Based on the extracted features, the study creates training and testing datasets for disease and pest identification. A trained knowledge base is developed, which classifies new images into their respective classes of diseases and pests. This classification enables efficient and accurate diagnosis, aiding farmers in taking timely and appropriate actions to manage and control the threats to their crops. The automated system provides a non-intrusive and rapid method for identifying diseases and pests, allowing farmers to make informed decisions for effective crop protection and management.

"Cotton Crop Disease Detection Using SWIN Transformers and Attention-based CNN" [16] explains a proposed system which was trained on large and small datasets of images of healthy and diseased cotton crops, covering three different diseases. The model utilizes SWIN Transformers and Attention-based CNN to achieve high accuracy in detecting the diseases. The paper also discusses the benefits of using deep learning and computer vision for crop disease detection and compares the proposed model with other existing models. Overall, the proposed model shows promising results and can be used as a tool for early detection and prevention of crop diseases .

In another research article named **An Image is Worth 16x16 Words** [17] proposed the use of Transformers for image recognition at scale, shows the efficacy of using a pristine transformer model directly on sequences of picture patches, yielding exceptional performance in the realm of image classification problems. The findings demonstrate that the Vision Transformer (ViT) exhibits superior performance in comparison to state-of-the-art convolutional networks when pre-trained on extensive datasets and applied to various mid-sized or small image recognition benchmarks. Notably, the ViT achieves these excellent results while necessitating significantly fewer computational resources for training. The paper also discusses potential applications of this research and future directions for exploring self-supervised pre-training methods and further scaling of ViT.

The paper titled Vision Transformers For Weeds and Crops Classification Of High Resolution UAV Images [18] provides a comprehensive analysis of the application of Vision Transformers for the classification of high-resolution UAV images in the context of weeds and crops. This study investigates the use of self-attention processes using Vision Transformer (ViT) models for the purpose of classifying weeds and crops in high-resolution UAV pictures within the domain of plant classification.This study provides an overview of the process involved in obtaining, processing, and manually annotating the information obtained from a high-resolution camera installed on an unmanned aerial vehicle (UAV). The Vision Transformer (ViT) model demonstrates superior performance compared to the current state-of-the-art convolutional neural network (CNN)-based models, namely ResNet and

EfficientNet, even when trained on a limited amount of labelled data. The report finishes by presenting encouraging findings and discussing potential avenues for future research.

In a relevant study by **Jajja)** [19] The researchers put out a proposal to train a Compact Convolutional Transformer (CCT) model for the purpose of detecting instances of whitefly bites in cotton crops.The main objective of this work is to devise a method that is both more effective and precise in detecting the presence of whiteflies, a common pest that can cause significant damage to cotton plantsThe CCT-based methodology integrates the respective advantages of Convolutional Neural Networks (CNNs) and Transformers. Convolutional Neural Networks (CNNs) demonstrate proficiency in extracting features from images, whereas Transformers have exceptional capabilities in gathering comprehensive contextual information and effectively modelling relationships that span across large distances. By integrating these two architectures, the CCT model aims to enhance its capability to detect whitefly attacks in cotton crops effectively.Compact version of the Transformer architecture tailored for image processing tasks, ensuring computational efficiency while maintaining competitive performance. The CCT model processes the input images using convolutional operations, followed by self-attention mechanisms from Transformers to capture relevant spatial relationships and contextual information.CCT-based approach in accurately detecting whitefly attacks in cotton crops. The model's compact design allows for faster inference and efficient deployment, making it well-suited for real-world agricultural applications.

A similar study with the name **"Vision transformers for remote sensing image classification"** [20] The text initiates The current consensus in the field acknowledges vision transformers as the leading remote-sensing scene-classification approaches in natural language processing. Unlike normal convolutional neural networks (CNNs), these models do not depend on convolution layers. Instead, multihead attention processes are employed as the primary component to establish extensive contextual relationships among pixels in visuals. The initial stage involves partitioning the analysed pictures into smaller sections, which are subse-

quently transformed into a sequential format through the processes of flattening and embedding. In order to retain information pertaining to the location, the inclusion of an embedding position is included inside these patches. Next, the generated sequence is inputted into many multihead attention layers in order to produce the ultimate representation. During the classification step, the initial sequence of tokens is inputted into a softmax classification layer.

The paper titled **"Understanding robustness of transformers for image classification"** [21] suggested that Deep Convolutional Neural Networks (CNNs) have historically been the preferred architectural choice for computer vision applications. In contemporary times, Transformer-based designs, such as the Vision Transformer (ViT), have demonstrated comparable or even superior performance to Residual Networks (ResNets) in the domain of picture categorization..The robustness of Vision Transformer (ViT) models versus ResNet baselines under input and model perturbations has been extensively studied. Despite initial concerns due to unique architecture aspects like non-overlapping patches, these models showed comparable robustness when pre-trained on substantial data. The ViT models also demonstrated resilience to changes in their structure, maintaining functionality even with the removal of almost any single layer. Despite high correlation between activations from later layers, their role in classification remained crucial, underscoring the model's robustness and versatility.

A novel research paper named **" Vision Transformer (ViT)-based Applications in Image Classification"** [22] provided information for ViT. The Vision Transformer (ViT) is a paradigm introduced by the Google team in 2020, which utilises a transformer architecture for the purpose of picture categorization.ViT model, which is a transformer-based model that applies self-attention mechanisms to image classification. The Vision Transformer (ViT) model, focusing on its capacity to depict long-range dependencies and its dynamic response characteristics.A comparison is made between the performance of Vision Transformer (ViT) and standard Convolutional Neural Networks (CNNs) in the context of picture categorization tasks. The findings indicate that the performance of ViT is comparable to that of CNNs, and in certain instances, it significantly surpasses them.

Additionally, the research presents an enhanced ViR model derived from ViT, hence augmenting the picture classification capabilities of ViT.

Another research using YOLOX model[23] proposed to address the challenges in automatic detection of plant diseases, specifically in cotton plants. Modifications include a Spatial Pyramid Pooling (SPP) layer for feature extraction and IoU-based regression loss function. Testing on a self-collected dataset from Pakistan, the model achieved a 73.13% mAP and outperformed the original YOLOX model by 3.27% in accuracy, demonstrating its efficacy in detecting diseases with varying symptoms and severities

The computerized method described in this work utilizes a pre-trained VGG-16 model[24] in conjunction with 11 fully convolutional layers for early detection of cotton leaf diseases. Data augmentation is employed to get a more balanced distribution of input data, hence enhancing the training process of the model. Additionally, the model is trained using carefully selected hyperparameters.

The rise of Transformers in language processing and computer vision has led to a belief that they are unsuitable for small data sets, a notion associated with concerns about limited data availability and exclusion of researchers with scarce resources. This study challenges the misconception that Transformers exhibit high data dependency, demonstrating their ability to rival contemporary Convolutional Neural Networks (CNNs) on limited datasets, frequently achieving superior accuracy while using fewer parameters. The authors provide a novel model that effectively removes the requirement for class tokens and positional embeddings by employing a unique sequence pooling method and convolutional operations. With flexibility in size, the model can function with as few as 0.28M parameters, performance of contemporary Convolutional Neural Networks (CNNs) such as ResNet and more current Neural Architecture Search (NAS)-based methods like Proxyless-NAS is surpassed. The compact and easily accessible architecture of Transformers expands the potential usage of this technology, even for individuals or organisations with limited resources or tiny datasets[25].

The CvT: Introducing Convolutions to Vision Transformers [26] paper introduces

a new architecture called Convolutional vision Transformer (CvT) that combines the benefits of convolutional neural networks (CNNs) and Transformers for image classification tasks. The CvT model incorporates several key components from Convolutional Neural Networks (CNNs), including local receptive fields, shared weights, and spatial subsampling. Additionally, it leverages dynamic attention, global context fusion, and improved generalisation techniques from Transformers. The authors demonstrate that CvT achieves state-of-the-art performance on ImageNet-1k and ImageNet-22k datasets while being lightweight and efficient. They also show that CvT can accommodate variable resolutions of input images and can be fine-tuned for downstream tasks.

Chapter 3

Methodology

This section would primarily focus on the approach and methodology that is followed in this research study. The main focus of the work includes identification of different cotton leaves diseases and to classify the intensity of the leaves affected by Whitefly insects.. A comparison is drawn amongst different deep learning models, improvements made in those models and to test the use of vision transformers for this problem to bring out the best results and taking a different approach while working with cotton disease detection. It also helps in bringing up the limitations and improvements that would be a focal point in the later research

3.1 Research Questions

The basic questions that this research aims to answer are as mentioned below

- How effectively Deep learning models are able to detect cotton disease and classify the intensity of affect in case of Whitefly?
- Which deep learning model successfully detects the cotton diseases with a high accuracy?
- Performance of Vision Transformer for the classification of cotton leaf diseases.
- Comparison of results for deep learning models and vision transformers.

- Comparison of Vision transformers for performance on the dataset.

The series of experiments conducted in the subsequent sections of the paper aimed to provide comprehensive answers to the research questions posed above. Through careful design and execution of the experiments, we would seek to obtain clear and reliable results that would shed light on the topic under investigation. The results obtained from the experiments were analyzed in detail to draw meaningful conclusions and address the objectives. By presenting these experiments and their corresponding results, the report aimed to contribute valuable insights and provide a solid foundation for further understanding of the topic.

3.2 Data Collection

Two publicly available datasets were used for the research. Both the datasets were merged to produce a large dataset having multiple classes of diseases keeping in mind the class imbalance factor. Dataset 1 had 6 classes of diseases with 600 images each in a single class. The classes were Aphids,Armyworm, Bacterial blight, Healthy, Powdery Mildew, Target spot. Dataset 2 had images for Whitefly diseases with classes of intensity of affected leaves due to Whitefly. THe classes were Mild,Severe,Nutritional deficiency. 200 images from each class of dataset 2 were used to build a new class whitefly and the final dataset for the model 1 was prepared.

The Dataset 2 was also used to train the model 2 to detect the intensity of the effect of whitefly on cotton leaves..[19]

DataSet for model 1: 600 images for each 7 classes

DataSet for model 2: 400 images for each 3 classes.

3.2.1 DataSet Distribution

The Distribution of the Dataset is as follows

Table 3.1: Data Classes Model-1

CLASSES	IMAGES
Aphids	600
Army Worm	600
Bacterial Blight	600
Healthy	600
Powdery Mildew	600
Target Spot	601
White Fly	600

Table 3.2: Data Classes Model-2

CLASSES	IMAGES
Severe	200
Mild	200
Nutritional deficiency	200

3.2.2 Data Preprocessing

The Data Preprocessing has five main stages of processing.

- Resizing:The photos should be resized to a uniform size that corresponds to the input dimensions required by the model. In this case, the `image_size` hyperparameter is set to 254, so you should resize the images to that resolution.
- Normalization: Normalize the pixel values of the images to bring them within a specific range. In general, the process entails rescaling the pixel values to a range of 0 to 1 or standardising them by removing the mean and dividing by the normal deviation.

- Data Augmentation: The implementation of data augmentation techniques is employed to enhance the variety of the training data and thus enhance the generalisation capabilities of the model. Common data augmentation techniques include random flips, rotations, zooming, and brightness adjustments.
- Converting to Tensor: Convert the images from NumPy arrays or other image formats to TensorFlow tensors. The model expects TensorFlow tensors as input.
- Batching: Organize the images into batches before feeding them to the model. This is usually done to improve computational efficiency during training.

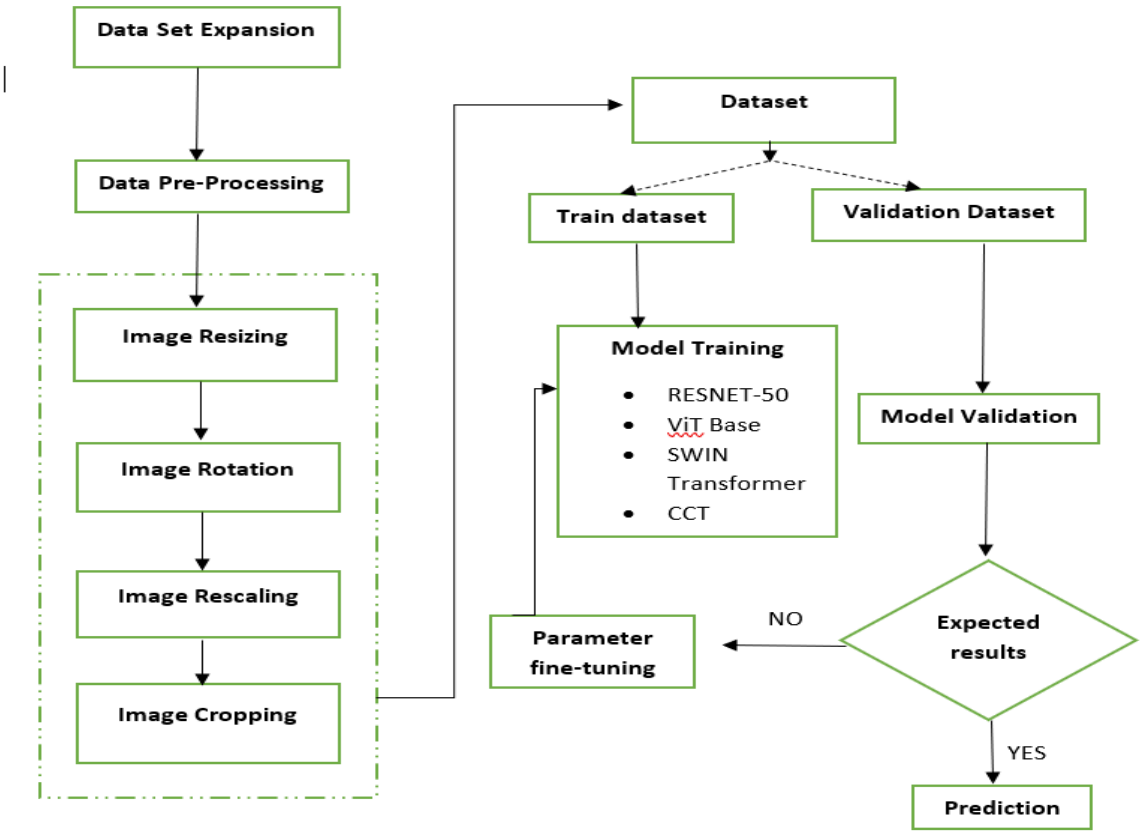


Figure 3.1: FLOWCHART

3.3 Architecture of CNN

Convolutional Neural Network (CNN) is a deep learning architecture particularly tailored for the processing of visual data. Fundamentally, a Convolutional Neural Network (CNN) has three primary categories of layers: convolutional layers, pooling layers, and fully connected layers. The convolutional layers apply filters to the input, extracting essential features such as edges or textures through convolution operations. Pooling layers then downsample the extracted features to reduce their dimensionality, preserving the most important information. These layers are often stacked, with multiple convolutional and pooling layers in sequence, to detect more complex patterns. Finally, the fully connected layers interpret these high-level patterns and perform the final classification or regression task. Regularisation approaches, such as the use of dropout, are commonly employed in academic research and practical applications to mitigate overfitting and improve the generalisation performance of machine learning models. CNNs are widely used for image classification, object detection, and various other computer vision tasks, effectively learning hierarchical representations of visual data

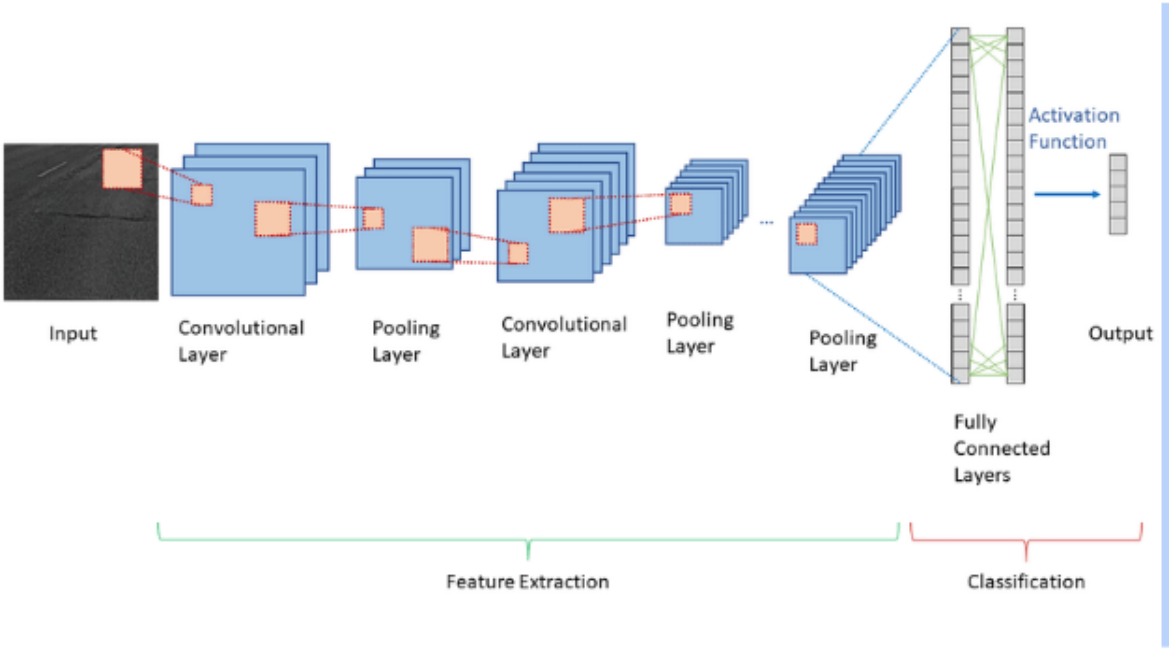


Figure 3.2: Basic CNN architecture[1]

3.3.1 ResNet Architecture

The ResNet-50 architecture is a specific configuration of the Residual Network (ResNet) that utilizes skip connections, or shortcuts, to jump over certain layers. Comprising 50 layers, ResNet-50 contains a series of stacked residual blocks where each block consists of a shortcut connection parallel to the main path of convolutional layers. The main path includes convolutional layers with small kernel sizes, such as 1x1 and 3x3, along with Batch Normalization and ReLU activation functions. The utilisation of shortcut connections facilitates the circumvention of the primary pathway for certain inputs, hence enabling the network to acquire identity functions with more ease.. This design helps mitigate the vanishing gradient problem and allows the training of much deeper networks. The architecture starts with an initial convolutional and max-pooling layer, architecture consists of a sequence of four residual blocks, followed by a global average pooling layer and a fully linked layer that is utilised for classification purposes. ResNet-50 is widely

utilized for various computer vision tasks due to its efficiency and ability to learn complex patterns.

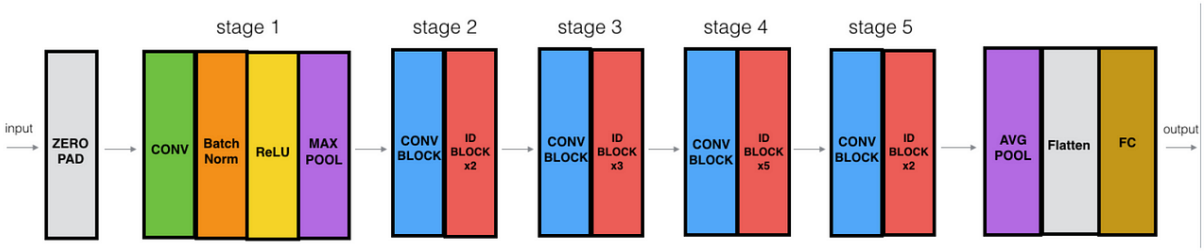


Figure 3.3: Resnet architecture[2]

The pre-trained ResNet-50 model is loaded without the top (classification) layer, additional unique dense layers, namely a Global Average Pooling layer and two fully linked layers using Rectified Linear Unit (ReLU) and softmax activations, respectively. The model's final layer's output size is determined by the number of classes in the dataset. The model is subsequently constructed using the Adam optimizer and sparse categorical cross-entropy loss, with accuracy and top-5-accuracy serving as evaluation measures. Additionally, the last ten layers of the model are set as trainable for fine-tuning. The model's summary is displayed, followed by training the model on the training dataset for 30 epochs. A batch size of 32 is utilised, and the model's performance is evaluated against a separate validation dataset. The historical record of training is saved within the variable denoted as "history".

```python
import tensorflow as tf
from tensorflow.keras.applications import ResNet50
from tensorflow.keras.models import Model
from tensorflow.keras.layers import Dense, GlobalAveragePooling2D
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow import keras

# Assuming you have loaded and preprocessed your image data into train_generator and validation_generator

# Define the input shape for your images (adjust based on your dataset)
device_name = tf.test.gpu_device_name()
if len(device_name) > 0:
    print("Found GPU at: {}".format(device_name))
else:
    device_name = "/device:CPU:0"
    print("No GPU, using {}.".format(device_name))
# Load the pre-trained ResNet50 model (weights will be downloaded automatically)
with tf.device(device_name):
  base_model = ResNet50(weights='imagenet', include_top=False, input_shape=(img_width, img_height, 3))

  # Add your custom layers for the classification task
  x = base_model.output
  x = GlobalAveragePooling2D()(x)
  x = Dense(512, activation='relu')(x)
  predictions = Dense(len(classes), activation='softmax')(x)  # Use the number of classes in your dataset

  # Create the final model
  model = Model(inputs=base_model.input, outputs=predictions)

  # Compile the model (you can choose the optimizer and loss based on your task)
  model.compile(optimizer='adam', loss='sparse_categorical_crossentropy', metrics=[
            keras.metrics.SparseCategoricalAccuracy(name="accuracy"),
            keras.metrics.SparseTopKCategoricalAccuracy(5, name="top-5-accuracy"),
        ])
  for layer in model.layers[-10:]:
      layer.trainable = True
  # Print the summary of the model
  model.summary()

  # Train the model on your dataset with fine-tuning
  epochs = 30
  batch_size = 32

  history=model.fit(
      X_train, y_train,
      validation_data=(X_test, y_test),
      batch_size=batch_size,
      epochs=epochs
  )
```

Figure 3.4: Resnet Model Code

3.4 Vision Transformer

The Vision Transformer (ViT) is an innovative architectural approach that employs transformers, initially developed for natural language processing, for the purpose of addressing computer vision challenges. In contrast to conventional convolutional neural networks (CNNs) which employ local operations for image

processing, the Vision Transformer (ViT) approach involves partitioning a picture into a predetermined number of non-overlapping patches and subsequently embedding them as a sequential arrangement of vectors. Subsequently, the vectors undergo processing through a series of transformer layers, enabling the model to concurrently analyse the interconnections among all components of the image. To convey spatial arrangement information of the patches, a learnable positional encoding is incorporated into the patch embeddings. Finally, the processed sequence is passed through a classifier head to generate predictions.

3.4.1 Architecture of Vision Transformer

The Vision Transformer (ViT) architecture applies the transformer model to visual data by treating images as sequences of patches. An image is divided into fixed-size non-overlapping patches, and each patch is linearly embedded into a vector. Positional encodings are added to these vectors to maintain spatial information, forming the input sequence for the transformer. The transformer architecture is composed of several levels, with each layer comprising multi-head self-attention mechanisms and feed-forward neural networks. The utilisation of the attention mechanism enables the model to assess the significance of various patches in respect to one another, therefore capturing comprehensive interconnections within the image. The final output of the transformer is taken from the embedding corresponding to a special [CLS] token or aggregated across patch embeddings and passed through a classification head, usually a simple linear layer, to produce predictions for tasks like image classification. The ViT architecture has proven effective in various computer vision tasks and highlights the flexibility and capability of the transformer model outside the realm of text.

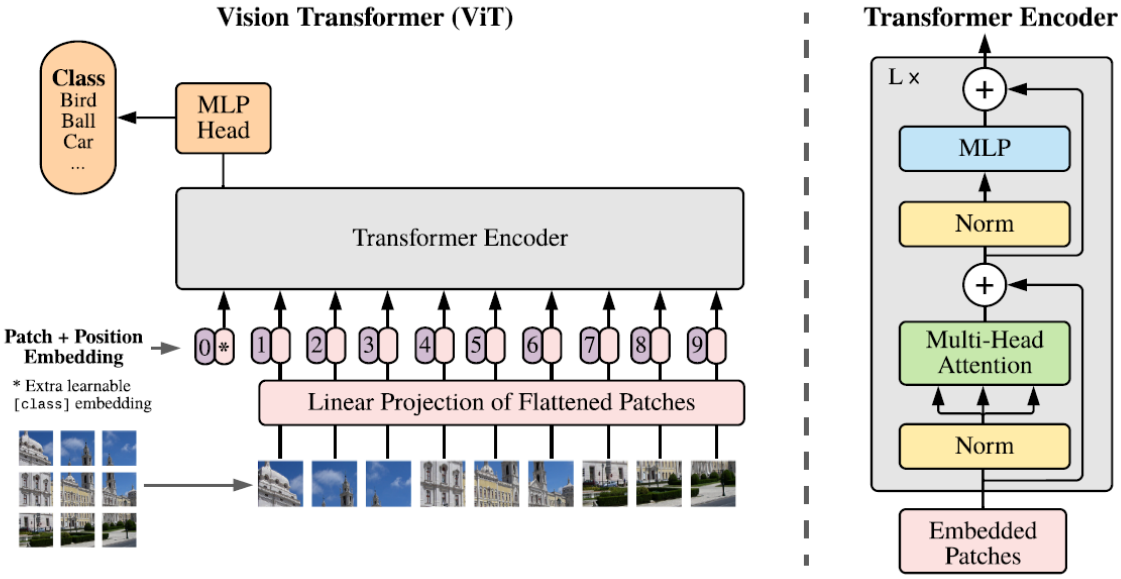


Figure 3.5: Vision Transformer Architecture[3]

3.4.2 ViT Base

ViT Base is a specific configuration of the Vision Transformer (ViT) architecture, the objective of this design is to provide a harmonious equilibrium between computing efficiency and performance in the realm of computer vision activities. In the ViT Base model, an image is divided into 16x16 patches, and these patches are linearly embedded into 768-dimensional vectors. The architectural component referred to as the transformer has a total of 12 layers. Each of these layers is equipped with a multi-head self-attention mechanism, which consists of 12 attention heads. The feed-forward neural networks within the transformer layers also follow specific dimensions. A special [CLS] token is used to gather classification information, and its corresponding output embedding is sent through a final linear layer to generate predictions. ViT Base is often used as a standard configuration for various vision tasks, offering a more compact alternative to larger ViT models while still delivering strong performance.

3.4.2.1 Data Split and Hyperparameters

```
[ ] x_train, x_test, y_train, y_test = train_test_split(X, labels, test_size=0.2, random_state=42, stratify=labels)

[ ] print(f"x_train shape: {x_train.shape} - y_train shape: {y_train.shape}")
    print(f"x_test shape: {x_test.shape} - y_test shape: {y_test.shape}")

    x_train shape: (2974, 254, 254, 3) - y_train shape: (2974,)
    x_test shape: (744, 254, 254, 3) - y_test shape: (744,)

[ ] num_classes = 7
    input_shape = (254, 254, 3)

    learning_rate = 0.001
    weight_decay = 0.0001
    batch_size = 256
    num_epochs = 40
    image_size = 72
    patch_size = 5
    num_patches = (image_size // patch_size) ** 2
    projection_dim = 64
    num_heads = 4
    transformer_units = [projection_dim * 2,projection_dim,]
    transformer_layers = 8
    mlp_head_units = [2048, 1024]
```

Figure 3.6: Vision Transformer Hyperparameter

3.4.2.2 Data Augmentation

```
import numpy as np
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers
import tensorflow_addons as tfa
data_augmentation = keras.Sequential(
    [
        layers.Normalization(),
        layers.Resizing(72, 72),
        layers.RandomFlip("horizontal"),
        layers.RandomRotation(factor=0.02),
        layers.RandomZoom(
            height_factor=0.2, width_factor=0.2
        ),
    ],
    name="data_augmentation",
)
data_augmentation.layers[0].adapt(x_train)
```

Figure 3.7: ViT Data Augmentation

3.4.2.3 MLP Unit and Image Patches

Mlp function creates a Multi-Layer Perceptron (MLP), a type of feedforward neural network. It builds dense layers with the specified number of hidden units using the GELU activation function. A dropout operation, set by `dropout_rate`, follows each dense layer, acting as a regularization technique to mitigate overfitting.

Patches function creates a custom TensorFlow layer, extracts non-overlapping patches from the input images. Upon initialization, it sets a `patch_size` parameter. The call method uses `tf.image.extract_patches` to segment input images into patches of the given size.

```
[ ]  def mlp(x, hidden_units, dropout_rate):
         for units in hidden_units:
             x = layers.Dense(units, activation=tf.nn.gelu)(x)
             x = layers.Dropout(dropout_rate)(x)
         return x
```

```
[ ]  class Patches(layers.Layer):
         def __init__(self, patch_size):
             super().__init__()
             self.patch_size = patch_size

         def call(self, images):
             batch_size = tf.shape(images)[0]
             patches = tf.image.extract_patches(
                 images=images,
                 sizes=[1, self.patch_size, self.patch_size, 1],
                 strides=[1, self.patch_size, self.patch_size, 1],
                 rates=[1, 1, 1, 1],
                 padding="VALID",
             )
             patch_dims = patches.shape[-1]
             patches = tf.reshape(patches, [batch_size, -1, patch_dims])
             return patches
```

Figure 3.8: MLP Unit

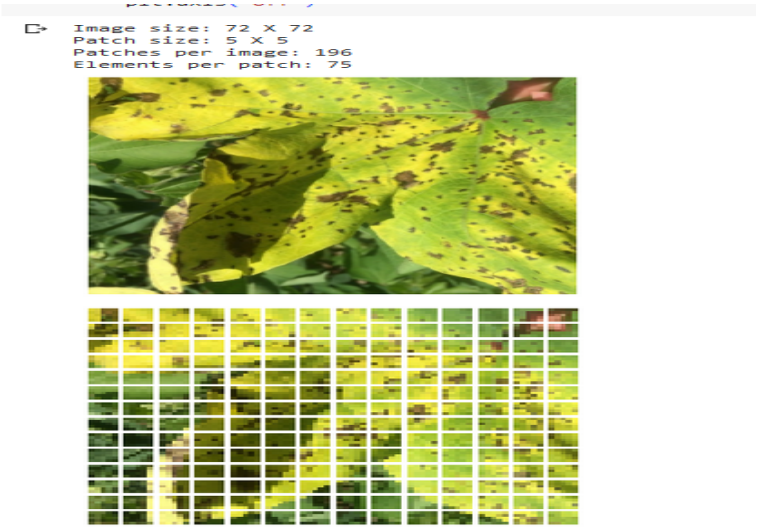


Figure 3.9: Image after patch

3.4.2.4 ViT Classifier

The function constructs a Vision Transformer (ViT) classifier model. It starts by accepting an input of a defined shape and applying data augmentation to it. This is followed by breaking down the augmented input into patches and encoding these patches using the previously defined Patches and PatchEncoder classes, respectively. Multiple layers of Transformer blocks are then created, each layer comprises a multi-head self-attention mechanism and a feedforward neural network (MLP). For improved learning stability, residual connections and layer normalization are integrated around both components. The output from the final Transformer block undergoes layer normalization, is then flattened, and processed via a dropout operation, followed by an MLP. Ultimately, the final class probabilities are obtained through a dense layer with units equal to the number of classes.

```
[ ]  def create_vit_classifier():
         inputs = layers.Input(shape=input_shape)
         # Augment data.
         augmented = data_augmentation(inputs)
         # Create patches.
         patches = Patches(patch_size)(augmented)
         # Encode patches.
         encoded_patches = PatchEncoder(num_patches, projection_dim)(patches)

         # Create multiple layers of the Transformer block.
         for _ in range(transformer_layers):
             # Layer normalization 1.
             x1 = layers.LayerNormalization(epsilon=1e-6)(encoded_patches)
             # Create a multi-head attention layer.
             attention_output = layers.MultiHeadAttention(
                 num_heads=num_heads, key_dim=projection_dim, dropout=0.1
             )(x1, x1)
             # Skip connection 1.
             x2 = layers.Add()([attention_output, encoded_patches])
             # Layer normalization 2.
             x3 = layers.LayerNormalization(epsilon=1e-6)(x2)
             # MLP.
             x3 = mlp(x3, hidden_units=transformer_units, dropout_rate=0.1)
             # Skip connection 2.
             encoded_patches = layers.Add()([x3, x2])

         # Create a [batch_size, projection_dim] tensor.
         representation = layers.LayerNormalization(epsilon=1e-6)(encoded_patches)
         representation = layers.Flatten()(representation)
         representation = layers.Dropout(0.5)(representation)
         # Add MLP.
         features = mlp(representation, hidden_units=mlp_head_units, dropout_rate=0.5)
         # Classify outputs.
         logits = layers.Dense(num_classes)(features)
         # Create the Keras model.
         model = keras.Model(inputs=inputs, outputs=logits)
         return model
```

Figure 3.10: ViT Classifier

The AdamW optimizer is employed with a predetermined learning rate and weight decay. Additionally, the model is constructed using the Sparse Categorical Crossentropy loss function, along with accuracy and top-5 accuracy measures. A checkpoint callback is created to save the best weights of the model during training to a temporary file path. The model is then trained for 75 epochs on the training data with a validation split of 10%, using the specified batch size, and the checkpoint callback to save the best model.Following the completion of training, the optimal weights are loaded, and subsequently, the model is assessed using the test data.

```
with tf.device(device_name):
  def run_experiment(model):
      optimizer = tfa.optimizers.AdamW(
          learning_rate=learning_rate, weight_decay=weight_decay
      )

      model.compile(
          optimizer=optimizer,
          loss=keras.losses.SparseCategoricalCrossentropy(from_logits=True),
          metrics=[
              keras.metrics.SparseCategoricalAccuracy(name="accuracy"),
              keras.metrics.SparseTopKCategoricalAccuracy(5, name="top-5-accuracy"),
          ],
      )

      checkpoint_filepath = "/tmp/checkpoint"
      checkpoint_callback = keras.callbacks.ModelCheckpoint(
          checkpoint_filepath,
          monitor="val_accuracy",
          save_best_only=True,
          save_weights_only=True,
      )

      history = model.fit(
          x=x_train,
          y=y_train,
          batch_size=batch_size,
          epochs=75,
          validation_split=0.1,
          callbacks=[checkpoint_callback],
      )

      model.load_weights(checkpoint_filepath)
      _, accuracy, top_5_accuracy = model.evaluate(x_test, y_test)
      print(f"Test accuracy: {round(accuracy * 100, 2)}%")
      print(f"Test top 5 accuracy: {round(top_5_accuracy * 100, 2)}%")

      return history


 vit_classifier = create_vit_classifier()
 history = run_experiment(vit_classifier)
```

Figure 3.11: ViT model training

3.4.3 SWIN Transformer

The SWIN Transformer, also known as the Shifted Window Transformer, is a vision paradigm that use Transformers to process both local windows and global pictures, therefore establishing a hierarchical framework. Unlike traditional Transformers that operate on fixed-size patches,the SWIN model employs a technique wherein the picture is partitioned into distinct windows that do not overlap, and subsequently applies self-attention mechanisms within each of these windows. In order to encompass a wider scope, these windows are systematically moved across many layers, so establishing a method of shifted windowing.. This allows for more efficient computation without sacrificing the ability to model long-range dependencies. By combining local and global attentions in this hierarchical manner, SWIN Transformers provide strong performance for various vision tasks. They are designed to be computationally efficient, and the architecture is easily scalable, making it suitable for different sizes and complexities of vision tasks.

3.4.3.1 Architecture of SWIN transformer

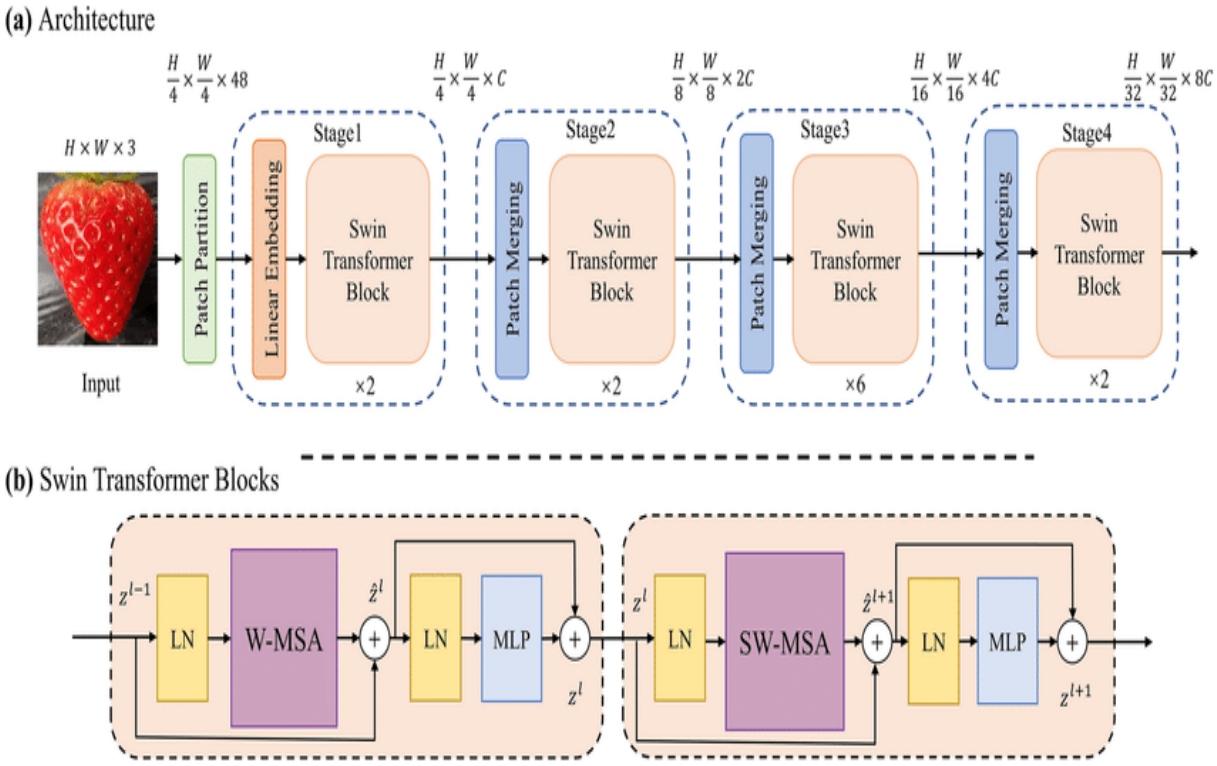


Figure 3.12: SWIM transformer architecture[4]

3.4.3.2 Hyperparameters

Hyperparameters for a SWIN Transformer model and its training. It uses 2x2 image patches, 8-head attention, 64-dimension embeddings, and MLP layers of size 256. Attention windows have size 2 with shift size 1. Training parameters include a learning rate of 0.001, batch size of 128, and 40 epochs. The model applies 10% validation split, weight decay of 0.0001 for regularization, and label smoothing with a factor of 0.1

```
[ ] patch_size = (2, 2)  # 2-by-2 sized patches
    dropout_rate = 0.03  # Dropout rate
    num_heads = 8  # Attention heads
    embed_dim = 64  # Embedding dimension
    num_mlp = 256  # MLP layer size
    qkv_bias = True  # Convert embedded patches to query, key, and values with a learnable additive value
    window_size = 2  # Size of attention window
    shift_size = 1  # Size of shifting window
    image_dimension = 164  # Initial image size

    num_patch_x = input_shape[0] // patch_size[0]
    num_patch_y = input_shape[1] // patch_size[1]

    learning_rate = 1e-3
    batch_size = 128
    num_epochs = 40
    validation_split = 0.1
    weight_decay = 0.0001
    label_smoothing = 0.1
```

Figure 3.13: SWIM transformer hyperparameters

3.4.3.3 Window Partition

`window_partition(x, window_size)`: Splits the input tensor x into non-overlapping windows of `window_size` size. This is done by reshaping and transposing the input.

`window_reverse(windows, window_size, height, width, channels)`: It reverts the window partition operation, transforming the windowed input back to its original form. The transformation is achieved by reshaping and transposing the windowed input.

`DropPath`: A custom Keras layer that implements DropPath regularization, a variant of dropout. Given a probability `drop_prob`, it stochastically drops entire paths (as opposed to individual nodes in standard dropout) in the computational graph of the model. This is done by generating a random tensor, creating a binary path mask, and applying it to the input. If a path is dropped, the remaining paths are scaled to maintain the expectation of the output.

```python
def window_partition(x, window_size):
    _, height, width, channels = x.shape
    patch_num_y = height // window_size
    patch_num_x = width // window_size
    x = tf.reshape(
        x, shape=(-1, patch_num_y, window_size, patch_num_x, window_size, channels)
    )
    x = tf.transpose(x, (0, 1, 3, 2, 4, 5))
    windows = tf.reshape(x, shape=(-1, window_size, window_size, channels))
    return windows


def window_reverse(windows, window_size, height, width, channels):
    patch_num_y = height // window_size
    patch_num_x = width // window_size
    x = tf.reshape(
        windows,
        shape=(-1, patch_num_y, patch_num_x, window_size, window_size, channels),
    )
    x = tf.transpose(x, perm=(0, 1, 3, 2, 4, 5))
    x = tf.reshape(x, shape=(-1, height, width, channels))
    return x


class DropPath(layers.Layer):
    def __init__(self, drop_prob=None, **kwargs):
        super().__init__(**kwargs)
        self.drop_prob = drop_prob

    def call(self, x):
        input_shape = tf.shape(x)
        batch_size = input_shape[0]
        rank = x.shape.rank
        shape = (batch_size,) + (1,) * (rank - 1)
        random_tensor = (1 - self.drop_prob) + tf.random.uniform(shape, dtype=x.dtype)
        path_mask = tf.floor(random_tensor)
        output = tf.math.divide(x, 1 - self.drop_prob) * path_mask
        return output
```

Figure 3.14: SWIM transformer Window function

3.4.3.4 Window Attention

The WindowAttention class implements a window-based multi-head self-attention layer. It initializes with parameters like dimensions, window size, number of heads, and QKV bias. The build method sets up a table to track relative positional biases. During the forward pass in call method, the input undergoes QKV transformation, scaled dot-product attention calculation with positional biases, optional masking, softmax normalization, and dropout. The layer output results from a projection of the attention-weighted values

```python
class WindowAttention(layers.Layer):
    def __init__(
        self, dim, window_size, num_heads, qkv_bias=True, dropout_rate=0.0, **kwargs
    ):
        super().__init__(**kwargs)
        self.dim = dim
        self.window_size = window_size
        self.num_heads = num_heads
        self.scale = (dim // num_heads) ** -0.5
        self.qkv = layers.Dense(dim * 3, use_bias=qkv_bias)
        self.dropout = layers.Dropout(dropout_rate)
        self.proj = layers.Dense(dim)

    def build(self, input_shape):
        num_window_elements = (2 * self.window_size[0] - 1) * (
            2 * self.window_size[1] - 1
        )
        self.relative_position_bias_table = self.add_weight(
            shape=(num_window_elements, self.num_heads),
            initializer=tf.initializers.Zeros(),
            trainable=True,
        )
        coords_h = np.arange(self.window_size[0])
        coords_w = np.arange(self.window_size[1])
        coords_matrix = np.meshgrid(coords_h, coords_w, indexing="ij")
        coords = np.stack(coords_matrix)
        coords_flatten = coords.reshape(2, -1)
        relative_coords = coords_flatten[:, :, None] - coords_flatten[:, None, :]
        relative_coords = relative_coords.transpose([1, 2, 0])
        relative_coords[:, :, 0] += self.window_size[0] - 1
        relative_coords[:, :, 1] += self.window_size[1] - 1
        relative_coords[:, :, 0] *= 2 * self.window_size[1] - 1
        relative_position_index = relative_coords.sum(-1)

        self.relative_position_index = tf.Variable(
            initial_value=tf.convert_to_tensor(relative_position_index), trainable=False
        )
```

Figure 3.15: SWIM transformer Window Attention

3.4.3.5 SWIN Transformer

The SwinTransformer class represents a Transformer block with a Swin Transformer architecture. Upon initialization, it sets up key layers like LayerNormalization, WindowAttention, DropPath, and an MLP sequence. The build method configures an attention mask if the shift size isn't zero. During the forward pass in call method, the input goes through layer normalization, partitioning into windows, window attention application (with optional shift), path dropout, and finally an MLP. An important feature here is the residual connections used after drop path and MLP, adding the original input (or 'skip' input) back into the layer outputs, promoting information flow throughout the network.

```python
class SwinTransformer(layers.Layer):
    def __init__(
        self,
        dim,
        num_patch,
        num_heads,
        window_size=7,
        shift_size=0,
        num_mlp=1024,
        qkv_bias=True,
        dropout_rate=0.0,
        **kwargs,
    ):
        super().__init__(**kwargs)

        self.dim = dim  # number of input dimensions
        self.num_patch = num_patch  # number of embedded patches
        self.num_heads = num_heads  # number of attention heads
        self.window_size = window_size  # size of window
        self.shift_size = shift_size  # size of window shift
        self.num_mlp = num_mlp  # number of MLP nodes

        self.norm1 = layers.LayerNormalization(epsilon=1e-5)
        self.attn = WindowAttention(
            dim,
            window_size=(self.window_size, self.window_size),
            num_heads=num_heads,
            qkv_bias=qkv_bias,
            dropout_rate=dropout_rate,
        )
        self.drop_path = DropPath(dropout_rate)
        self.norm2 = layers.LayerNormalization(epsilon=1e-5)

        self.mlp = keras.Sequential(
            [
                layers.Dense(num_mlp),
                layers.Activation(keras.activations.gelu),
                layers.Dropout(dropout_rate),
                layers.Dense(dim),
                layers.Dropout(dropout_rate),
            ]
        )

        if min(self.num_patch) < self.window_size:
            self.shift_size = 0
            self.window_size = min(self.num_patch)

    def build(self, input_shape):
        if self.shift_size == 0:
            self.attn_mask = None
        else:
            height, width = self.num_patch
            h_slices = (
                slice(0, -self.window_size),
                slice(-self.window_size, -self.shift_size),
                slice(-self.shift_size, None),
            )
            w_slices = (
                slice(0, -self.window_size),
                slice(-self.window_size, -self.shift_size),
                slice(-self.shift_size, None),
```

Figure 3.16: SWIM Transformer

First, the input image is processed with random cropping and horizontal flipping for data augmentation. Then, patches are extracted from the image using a custom PatchExtract layer, and these patches are embedded into vectors using a PatchEmbedding layer. Two instances of the SwinTransformer layer are applied sequentially, responsible for modeling both local and global relationships between the patches. Each SWIN Transformer has specific hyperparameters such as embedding dimensions, the number of attention heads, window size, shift size, MLP size, QKV bias, and dropout rate. After the second transformer block, the patches are merged using a PatchMerging layer. The resulting output is pooled globally, and a dense output layer with softmax activation is used to produce the final class predictions. The overall architecture represents a deep neural network that leverages the SWIN Transformer to handle visual tasks and is designed for

classification over a predefined number of classes.

```
input = layers.Input(input_shape)
x = layers.RandomCrop(image_dimension, image_dimension)(input)
x = layers.RandomFlip("horizontal")(x)
x = PatchExtract(patch_size)(x)
x = PatchEmbedding(num_patch_x * num_patch_y, embed_dim)(x)
x = SwinTransformer(
    dim=embed_dim,
    num_patch=(num_patch_x, num_patch_y),
    num_heads=num_heads,
    window_size=window_size,
    shift_size=0,
    num_mlp=num_mlp,
    qkv_bias=qkv_bias,
    dropout_rate=dropout_rate,
)(x)
x = SwinTransformer(
    dim=embed_dim,
    num_patch=(num_patch_x, num_patch_y),
    num_heads=num_heads,
    window_size=window_size,
    shift_size=shift_size,
    num_mlp=num_mlp,
    qkv_bias=qkv_bias,
    dropout_rate=dropout_rate,
)(x)
x = PatchMerging((num_patch_x, num_patch_y), embed_dim=embed_dim)(x)
x = layers.GlobalAveragePooling1D()(x)
output = layers.Dense(num_classes, activation="softmax")(x)
```

Figure 3.17: SWIM transformer Model training

3.4.4 Compact Convolution Transformer

Compact Convolutional Transformers (CCT) present a fusion of convolutional neural networks (CNNs) with transformers to efficiently process visual data. Unlike conventional transformers that operate on fixed-sized patches, CCT starts with a convolutional stem, applying smaller convolutions to the input image, effectively integrating local spatial information. This retains a more fine-grained understanding of the spatial structures in the image while reducing the sequence length that the transformer has to handle. After the convolutional stem, the transformed data is passed through a series of transformer blocks, inheriting the ability of transformers to model long-range dependencies. CCT aims to combine the

spatial effectiveness of CNNs with the expressive power of transformers, thereby enabling a more compact and computationally efficient model. It is found to be effective in various visual tasks, including image classification, without needing extensive computational resources like many of its transformer-based counterparts.

3.4.4.1 CCT Architecture

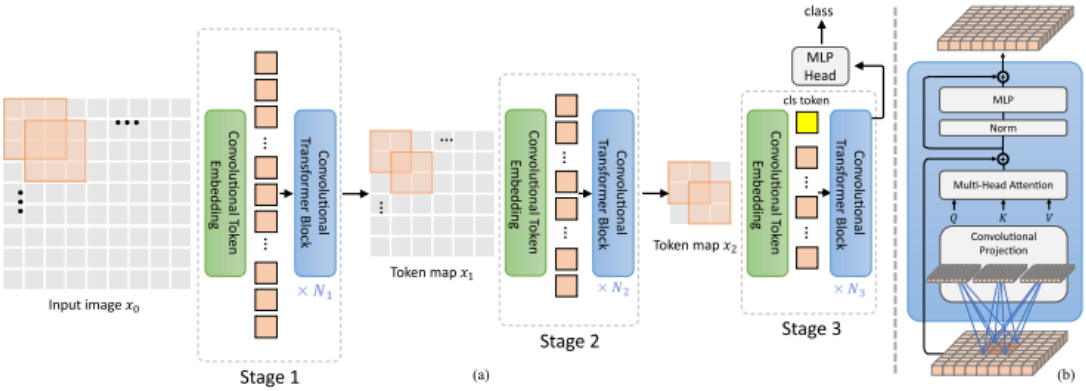


Figure 3.18: CCT Architecture[5]

3.4.4.2 CCT Model

The function `create_cct_model` builds a Convolutional Classifier Transformer (CCT) for image classification. It augments inputs, encodes them into patches with `CCTTokenizer`, and optionally adds positional embeddings. It then builds a series of transformer layers, each including layer normalization, multi-head self-attention, skip connections, and an MLP. Stochastic depth is used within these layers for regularization. Finally, it applies layer normalization and sequence pooling to the outputs of the transformer blocks, and passes them through a dense layer to get the class probabilities. The function returns this configured CCT model.

```
[ ]  def create_cct_model(
         image_size=image_size,
         input_shape=input_shape,
         num_heads=num_heads,
         projection_dim=projection_dim,
         transformer_units=transformer_units,
     ):

         inputs = layers.Input(input_shape)

         # Augment data.
         augmented = data_augmentation(inputs)

         # Encode patches.
         cct_tokenizer = CCTTokenizer()
         encoded_patches = cct_tokenizer(augmented)

         # Apply positional embedding.
         if positional_emb:
             pos_embed, seq_length = cct_tokenizer.positional_embedding(image_size)
             positions = tf.range(start=0, limit=seq_length, delta=1)
             position_embeddings = pos_embed(positions)
             encoded_patches += position_embeddings

         # Calculate Stochastic Depth probabilities.
         dpr = [x for x in np.linspace(0, stochastic_depth_rate, transformer_layers)]

         # Create multiple layers of the Transformer block.
         for i in range(transformer_layers):
             # Layer normalization 1.
             x1 = layers.LayerNormalization(epsilon=1e-5)(encoded_patches)

             # Create a multi-head attention layer.
             attention_output = layers.MultiHeadAttention(
                 num_heads=num_heads, key_dim=projection_dim, dropout=0.1
             )(x1, x1)

             # Skip connection 1.
             attention_output = StochasticDepth(dpr[i])(attention_output)
             x2 = layers.Add()([attention_output, encoded_patches])

             # Layer normalization 2.
             x3 = layers.LayerNormalization(epsilon=1e-5)(x2)

             # MLP.
             x3 = mlp(x3, hidden_units=transformer_units, dropout_rate=0.1)

             # Skip connection 2.
             x3 = StochasticDepth(dpr[i])(x3)
             encoded_patches = layers.Add()([x3, x2])

         # Apply sequence pooling.
         representation = layers.LayerNormalization(epsilon=1e-5)(encoded_patches)
         attention_weights = tf.nn.softmax(layers.Dense(1)(representation), axis=1)
         weighted_representation = tf.matmul(
             attention_weights, representation, transpose_a=True
         )
         weighted_representation = tf.squeeze(weighted_representation, -2)
         print(num_classes)
```

Figure 3.19: CCT Model

Chapter 4

Results and Challenges

The chapter presents the results of a study on Cotton disease detection using deep learning techniques. The study evaluates the effectiveness of different evaluation metrics, such as Accuracy, Mean absolute error and Classification Report in predicting intrusion attacks.

In the preceding section, an extensive exposition was provided on the comprehensive methodology, approach, and steps undertaken to train the three machine learning models. This detailed account encompassed the challenges encountered during the research process, elucidating their impact and the deliberate measures implemented to overcome them. Building upon this foundation, the subsequent section focuses on the outcomes attained as a direct consequence of the methodological framework delineated above. Herein, we delve into a comprehensive analysis of the results, presenting a comprehensive evaluation of the model performance and their efficacy in addressing the research objectives.

The research endeavor encompassed a meticulous application of various data transformation techniques and methodological approaches to Deep Learning models: *Resnet 50*, *ViT Base*,*SWIN transformer*, and *Compact Convolution Transformer*. Through a rigorous evaluation process involving multiple deep learning algorithms, these three models emerged as the focal point of the research investigation. Consequently, the dataset underwent extensive training and optimization procedures, aimed at maximizing the performance and efficacy of the models. This deliberate

focus on the three selected models underscores their significance and prominence within the research, setting the stage for comprehensive analysis and evaluation of their outcomes.

4.1 Evaluation Metrics

As discussed in the previous sections, the research had two main portions of the study. The first one was the detection of the records as either Normal or Attacked and the second one were for the detection of the stage at which the attack was carried out. For both these categories, bucket approach was followed, as discussed with details in the previous section, for the three machine learning models used. Four main evaluation metrics used for the models are

- *Accuracy Score*
- *Mean Absolute Error (MAE)*
- *Classification Report*

4.1.1 Accuracy

The accuracy score is a commonly utilised statistic for assessing the efficacy of machine learning models. The metric assesses the predictive accuracy of the model in assigning proper class labels to instances within a certain dataset. The accuracy score is given as a percentage and is calculated as the proportion of correctly categorised cases to all instances. It is frequently utilised in many machine learning applications and provides an accurate indicator of the model's overall predicted accuracy.

To determine the accuracy score for training and test datasets, the following steps are typically followed:

1. The training of the model is conducted by utilising the training dataset, which comprises instances that have been labelled with known class labels.

2. After training,the model is subsequently employed to generate predictions on the test dataset, which comprises examples that have been labelled and has known class labels. model's predicted class labels are compared to the actual class labels in the test dataset.

3. The accuracy score is obtained by multiplying the ratio of properly detected cases to the total number of occurrences in the test dataset by 100.

When the accuracy score is higher, it signifies that the model is making a greater number of correct predictions. Conversely, a lower accuracy score indicates that the model's predictions are less accurate. The accuracy score is a dependable metric used to evaluate the model's performance in terms of its ability to make accurate predictions.

While accuracy is an important metric for evaluating model performance, it may not provide a comprehensive understanding, particularly in situations with class imbalances or other complexities, as observed in our research. Therefore, it was necessary to incorporate additional evaluation metrics, including precision, recall, and F1 score. These metrics offer a more nuanced assessment of the model's performance from various perspectives and cater to specific requirements of the problem domain. By considering multiple evaluation metrics, we were able to gain a more comprehensive and reliable assessment of the model's performance.

4.1.2 Classification Report

In the deep learning **Classification Report** function offers a thorough analysis of a classification model's performance. It computes a number of metrics for each class in the classification task, including precision, recall, F1 score, and support.

- **PRECISION:** By comparing the proportion of accurate positive predictions to all positive predictions, precision, a performance metric for machine learning, quantifies the accuracy of positive predictions. It offers insightful data on the percentage of accurately identified positive examples, assisting in evaluating the model's accuracy in identifying good outcomes.

- **RECALL:** Recall, in the context of classification models, refers to the ratio of correctly predicted positive occurrences to the total number of positive instances. The evaluation measures the model's ability to effectively detect positive events and calculates the proportion of positive cases that are correctly classified.

- **F1 SCORE:** The F1 score, a widely used metric in machine learning, combines precision and recall by calculating their harmonic mean. This balanced measure takes into account both precision and recall, making it particularly useful in scenarios where achieving a balance between the two is important. The F1 score proves valuable in situations with imbalanced class distributions, allowing for a comprehensive evaluation of model performance.

Through an analysis of the accuracy, recall, and F1 score derived from the classification report, valuable insights were obtained on the efficacy of the models:

- High precision indicates a low false positive rate, meaning that the model has a low tendency to incorrectly classify negative instances as positive

- High recall indicates a low false negative rate, meaning that the model has a low tendency to incorrectly classify positive instances as negative

- The F1 score is a metric that offers a fair evaluation by taking into account both precision and recall.It is useful when you want to evaluate the model's overall performance, especially when there is an imbalance between classes

By analyzing these metrics for each class, you can understand how well the model is performing for different categories and identify any imbalances or specific issues. This information helped us make informed decisions about the model's performance and potential areas for improvement to get the optimized results for each model.

4.2 Deep Learning Model Result

This portion would discuss the results that we were able to achieve during the classification of cotton disease dataset.

4.2.1 Resnet 50

4.2.1.1 Training and Validation Accuracy

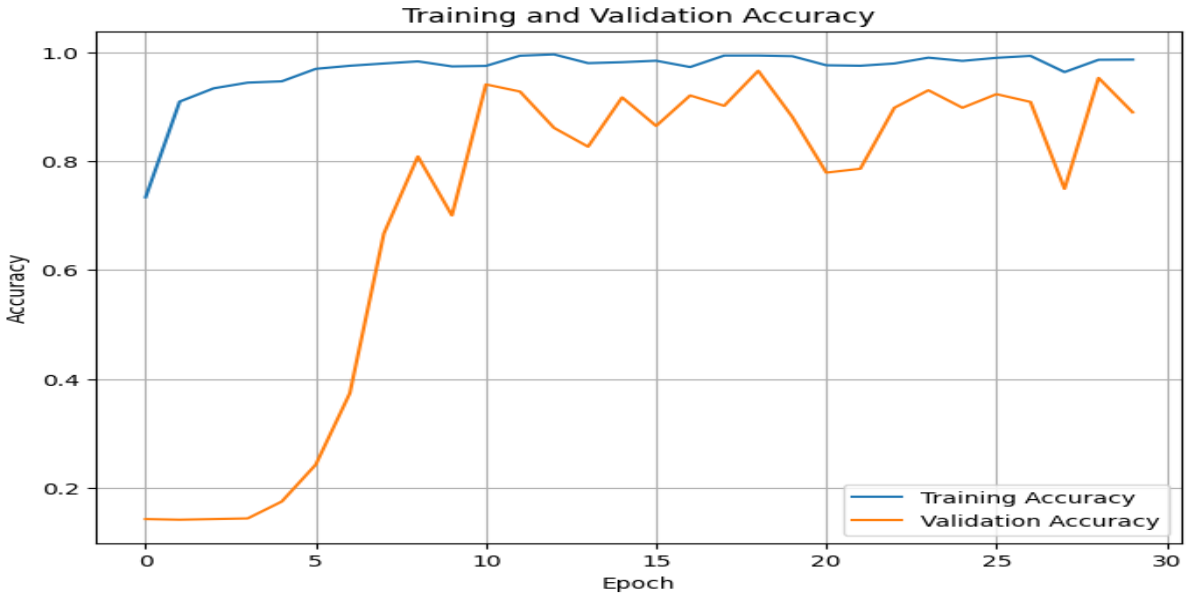


Figure 4.1: Resnet Training and Validation Accuracy

4.2.1.2 Classification Report

Classification report for the Resnet 50 Model using transfer learning

```
27/27 [==============================] - 1s 9ms/step
Accuracy: 0.8892857142857142
F1 Score: 0.8880631975434081
Classification Report:
                    precision    recall  f1-score   support

     Powdery Mildew       0.98      0.88      0.93       120
  Bacterial Blight        0.83      0.88      0.86       120
       Target spot        0.99      0.64      0.78       120
             Aphids       0.91      0.88      0.89       120
            Healthy       0.90      0.97      0.94       120
         Army worm        0.72      0.97      0.83       120
          Whitefly        0.99      1.00      1.00       120

          accuracy                           0.89       840
         macro avg        0.90      0.89      0.89       840
      weighted avg        0.90      0.89      0.89       840
```

Figure 4.2: Resnet Classification Report

4.2.2 ViT Base

4.2.2.1 Validation and Training Accuracy

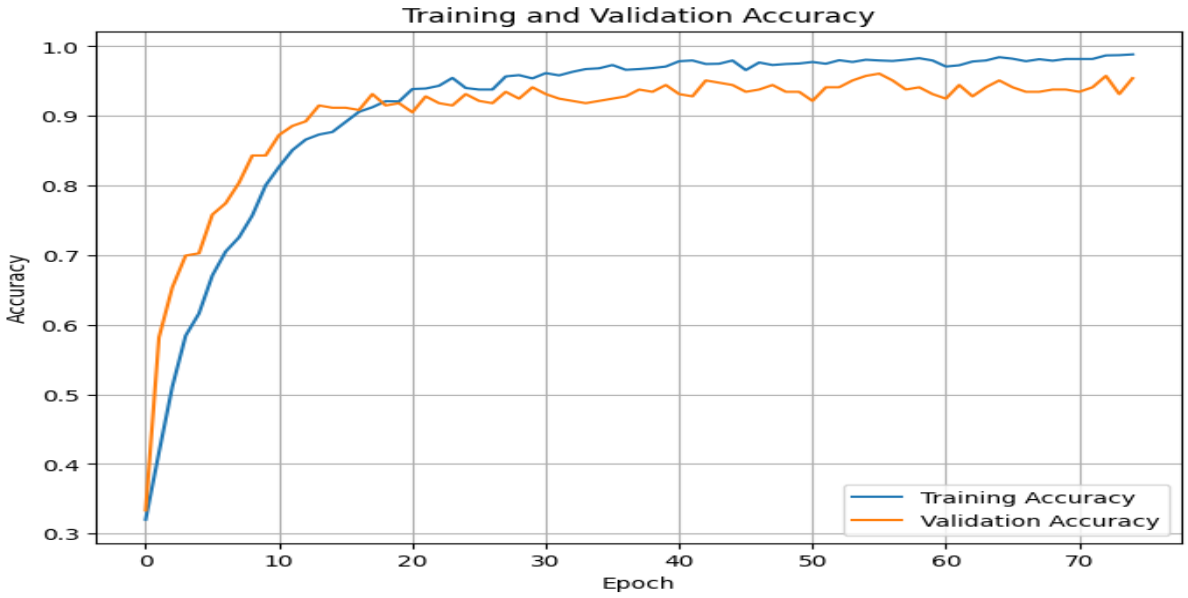


Figure 4.3: ViT Base Training and Validation Accuracy

4.2.2.2 Validation and Training Loss

Figure 4.4: ViT Base Training and Validation Loss

4.2.2.3 Top 5 Validation and Training Accuracy

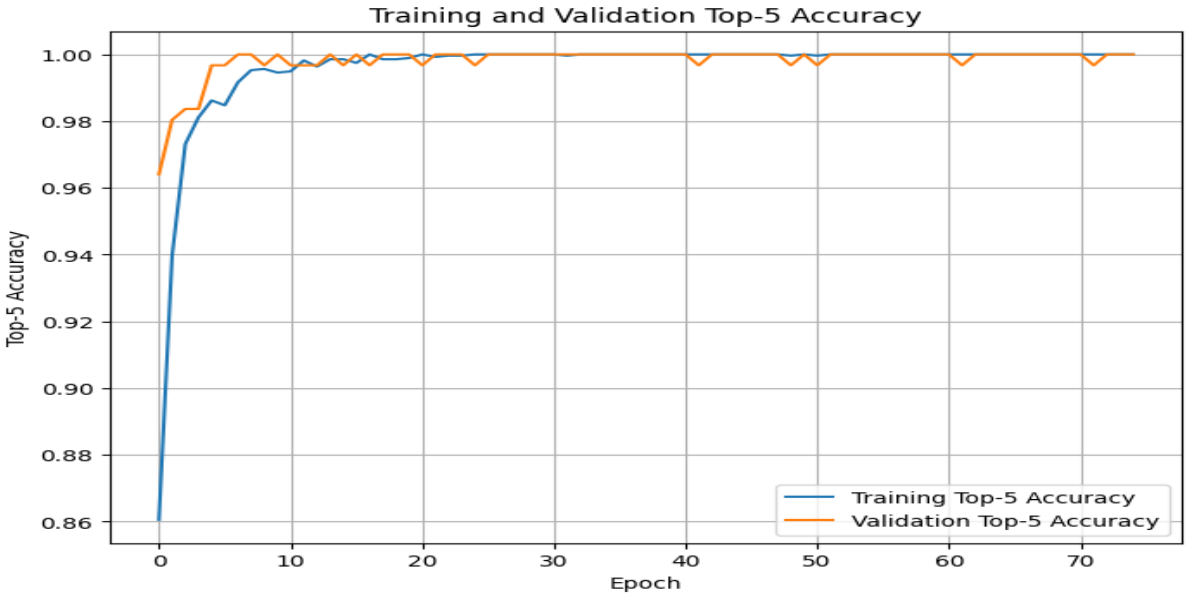


Figure 4.5: ViT Base Top5 Training and Validation Accuracy

4.2.2.4 Classification Report

```
24/24 [==============================] - 2s 25ms/step
Accuracy: 0.9476
Precision: 0.9471
Recall: 0.9466
F1 Score: 0.9468
Confusion Matrix:
[[ 95   1   1   4   0   3   0]
 [  1  99   0   2   0   2   0]
 [  2   1 100   1   0   0   0]
 [  4   0   0  97   0   3   0]
 [  0   2   2   1  99   0   0]
 [  2   3   0   2   0  96   1]
 [  0   0   0   0   1   0 119]]
Classification Report:
              precision    recall  f1-score   support

           0       0.91      0.91      0.91       104
           1       0.93      0.95      0.94       104
           2       0.97      0.96      0.97       104
           3       0.91      0.93      0.92       104
           4       0.99      0.95      0.97       104
           5       0.92      0.92      0.92       104
           6       0.99      0.99      0.99       120

    accuracy                           0.95       744
   macro avg       0.95      0.95      0.95       744
weighted avg       0.95      0.95      0.95       744
```

Figure 4.6: ViT Base Classification Report

4.2.2.5 Precision Recall Graph

```
24/24 [==============================] - 2s 25ms/step
Accuracy: 0.9476
Precision: 0.9471
Recall: 0.9466
F1 Score: 0.9468
Confusion Matrix:
[[ 95   1   1   4   0   3   0]
 [  1  99   0   2   0   2   0]
 [  2   1 100   1   0   0   0]
 [  4   0   0  97   0   3   0]
 [  0   2   2   1  99   0   0]
 [  2   3   0   2   0  96   1]
 [  0   0   0   0   1   0 119]]
Classification Report:
              precision    recall  f1-score   support

           0       0.91      0.91      0.91       104
           1       0.93      0.95      0.94       104
           2       0.97      0.96      0.97       104
           3       0.91      0.93      0.92       104
           4       0.99      0.95      0.97       104
           5       0.92      0.92      0.92       104
           6       0.99      0.99      0.99       120

    accuracy                           0.95       744
   macro avg       0.95      0.95      0.95       744
weighted avg       0.95      0.95      0.95       744
```

Figure 4.7: ViT Base Precision Recall Graph

4.2.3 Swin Transformer

4.2.3.1 Validation and Training Accuracy

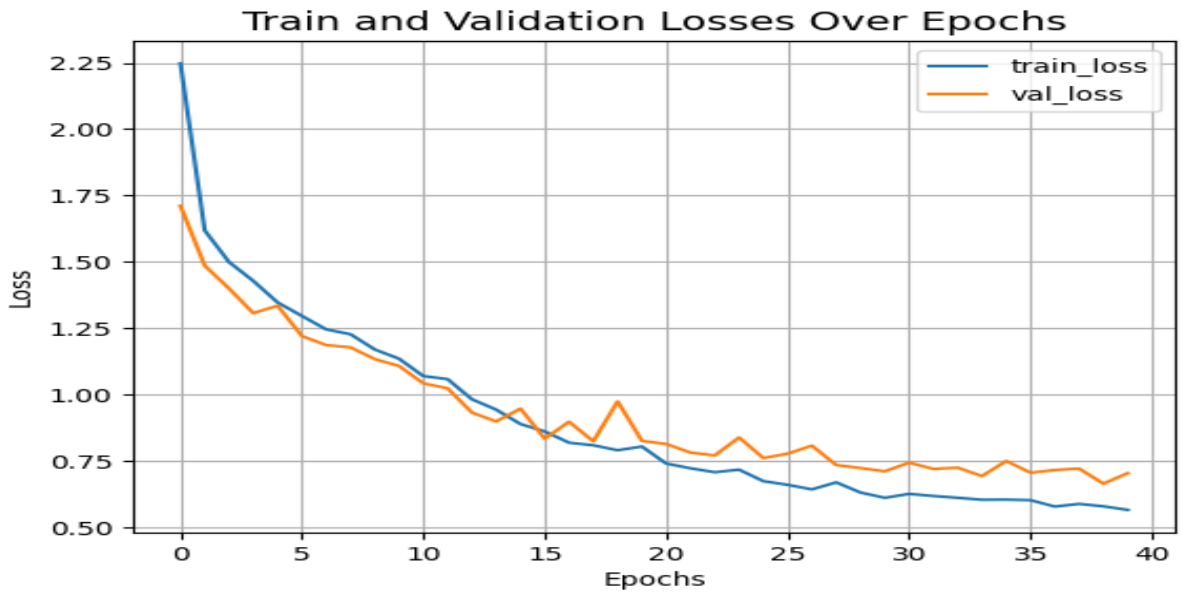


Figure 4.8: Swin Transformer Training and Validation Accuracy

4.2.3.2 Validation and Training Loss

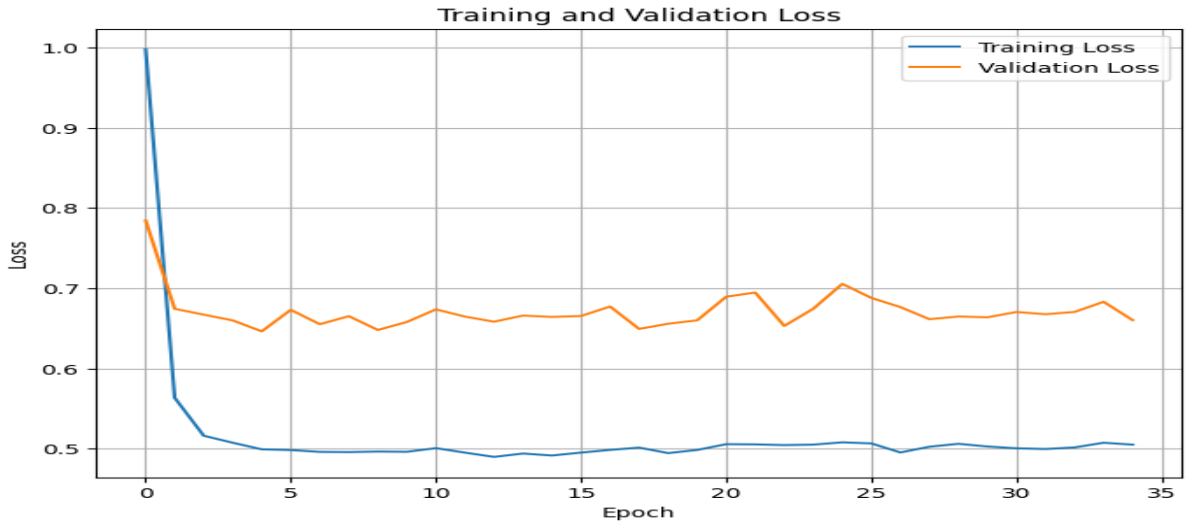


Figure 4.9: Swin Transformer Training and Validation Loss

4.2.3.3 Classification Report

```
27/27 [==============================] - 2s 30ms/step
Accuracy: 0.9024
Precision: 0.9047
Recall: 0.9024
F1 Score: 0.9020
Confusion Matrix:
[[114   1   1   1   1   1   1]
 [  1 104   8   1   4   1   1]
 [  3   8  97   2   7   2   1]
 [  1   8   2 104   2   3   0]
 [  1   0   2   0 116   1   0]
 [  4   3   1   0   9 103   0]
 [  0   0   0   0   0   0 120]]
Classification Report:
              precision    recall  f1-score   support

           0       0.92      0.95      0.93       120
           1       0.84      0.87      0.85       120
           2       0.87      0.81      0.84       120
           3       0.96      0.87      0.91       120
           4       0.83      0.97      0.90       120
           5       0.93      0.86      0.89       120
           6       0.98      1.00      0.99       120

    accuracy                           0.90       840
   macro avg       0.90      0.90      0.90       840
weighted avg       0.90      0.90      0.90       840
```

Figure 4.10: Swin Transformer Classification Report

4.2.4 Compact Convolution Transformer

4.2.4.1 Training and Validation Accuracy

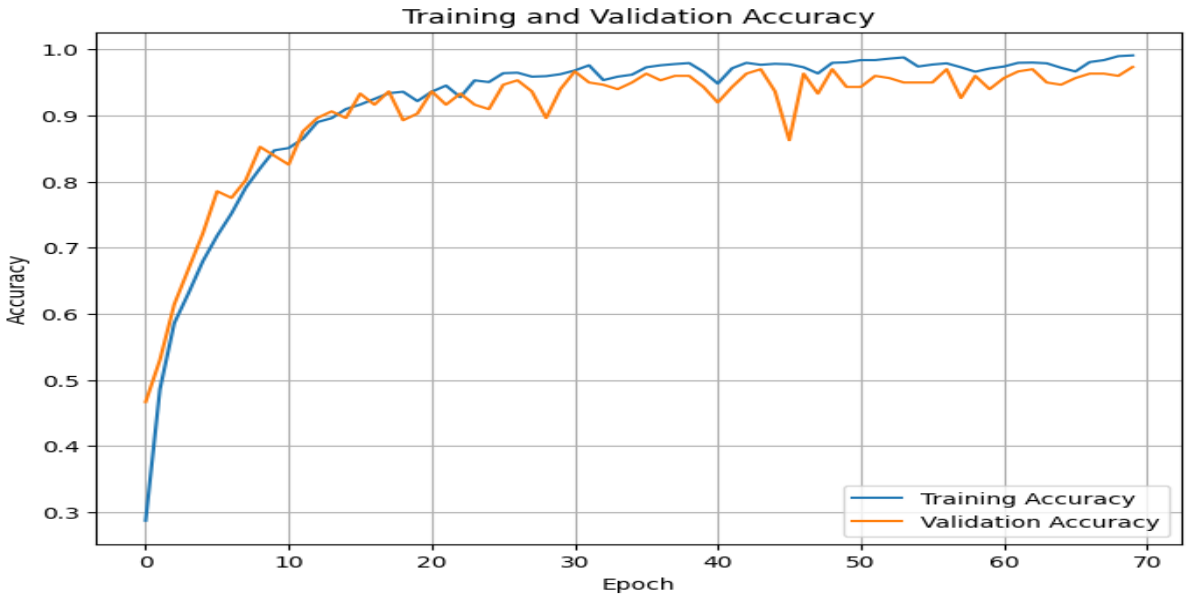


Figure 4.11: CCT Training and Validation Accuracy

4.2.4.2 Validation and Training Loss

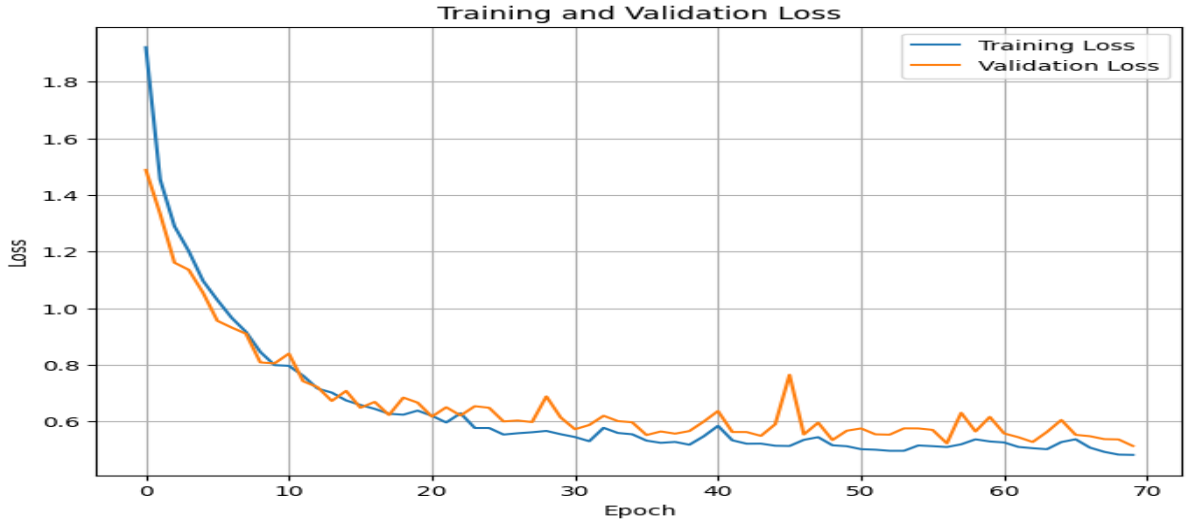


Figure 4.12: CCT Training and Validation Loss

4.2.4.3 Top 5 Validation and Training Accuracy

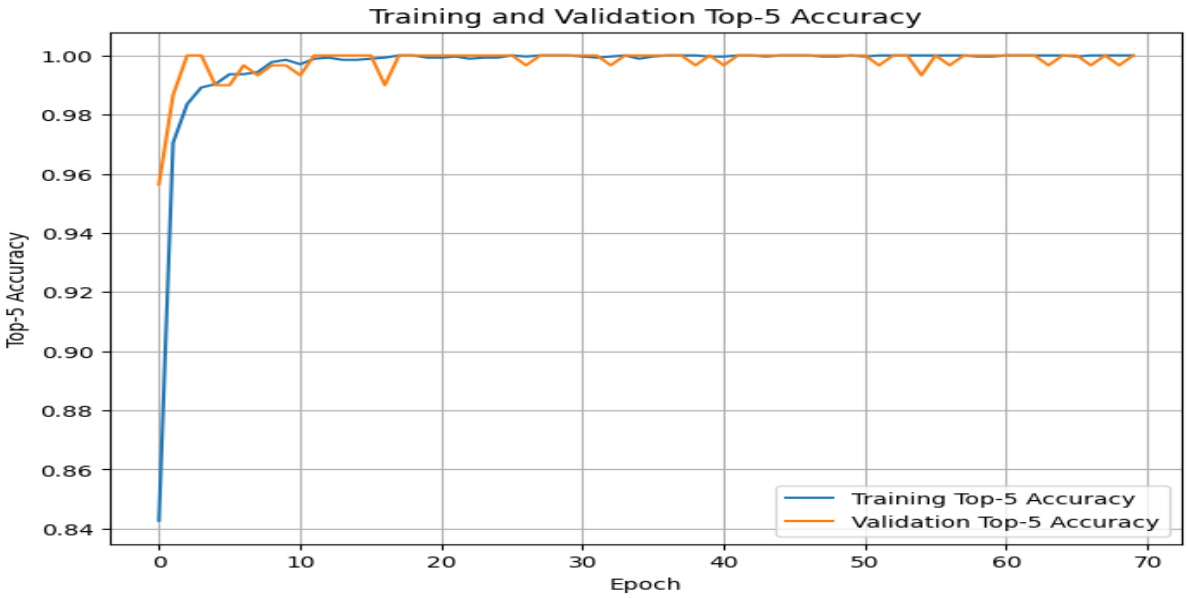


Figure 4.13: CCT Top5 Training and Validation Accuracy

4.2.4.4 Classification Report

```
27/27 [==============================] - 2s 36ms/step
Accuracy: 0.9560
Precision: 0.9559
Recall: 0.9560
F1 Score: 0.9559
Confusion Matrix:
[[117   1   0   0   1   1   0]
 [  0 113   5   0   0   2   0]
 [  1   3 110   2   1   3   0]
 [  0   1   0 115   4   0   0]
 [  1   0   0   1 116   2   0]
 [  1   2   2   2   1 112   0]
 [  0   0   0   0   0   0 120]]
Classification Report:
              precision    recall  f1-score   support

           0       0.97      0.97      0.97       120
           1       0.94      0.94      0.94       120
           2       0.94      0.92      0.93       120
           3       0.96      0.96      0.96       120
           4       0.94      0.97      0.95       120
           5       0.93      0.93      0.93       120
           6       1.00      1.00      1.00       120

    accuracy                           0.96       840
   macro avg       0.96      0.96      0.96       840
weighted avg       0.96      0.96      0.96       840
```

Figure 4.14: CCT Classification Report

4.2.4.5 Precision Recall Graph

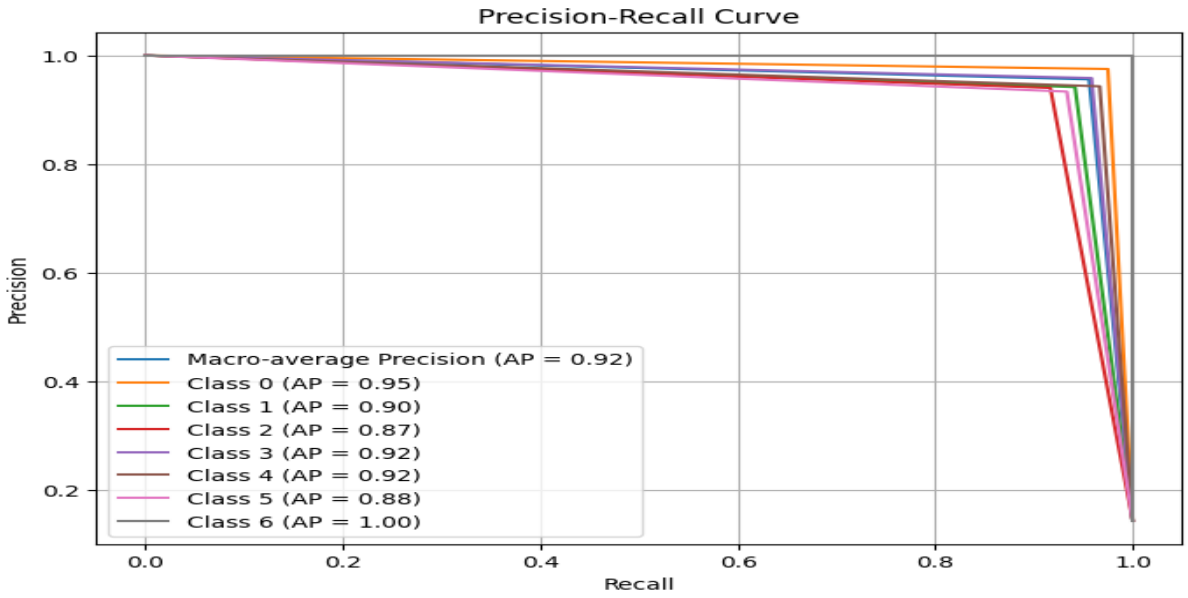


Figure 4.15: CCT Precision Recall Graph

4.2.5 Error Analysis

Error analysis in the context of deep learning refers to the process of examining mistakes made by your model. It involves diving deep into the cases where the model predicted incorrectly, understanding the possible reasons for these mistakes, and considering strategies to rectify them

4.2.5.1 Importance of Error Analysis

- Understanding Model Limitations: No model is perfect. By performing error analysis, you can understand where your model falls short
- Guiding Future Work: By understanding where errors are coming from, you can prioritize which aspects of your model or data to improve next
- Avoiding Overfitting: It can help in identifying if the model is overfitting to certain kinds of data or certain types of errors.

- Enhancing Generalization: Understanding error patterns can provide insights into model biases, which when rectified, can improve generalization.
- Building Trust: For those who use your model, seeing a detailed error analysis can build trust.

4.2.5.2 Error Analysis Models 1

- Class 0 and Class 3 Confusion: Class 0 instances are most commonly misclassified as class 3 (4 times). Similarly, class 3 instances are most commonly misclassified as class 0 (4 times). This indicates a potential similarity or overlap between these two classes which the model is finding challenging.
- High Precision Classes: Classes 2 and 6 have the least misclassifications, with class 6 having the highest true positive count of 119. This suggests the model can distinguish these classes quite well compared to others.
- Misclassification Patterns: Class 5 is misclassified across four other classes, indicating potential overlapping features or less clear boundaries with other classes.
- Potential Improvements: To improve the model's performance, one could investigate the data points that are commonly misclassified. For instance, examining the overlap between class 0 and class 3 could provide insights into the source of confusion.
- Overall Performance: The majority of predictions are on the diagonal, which indicates that the model performs well. However, the off-diagonal elements, which represent errors, provide valuable feedback for model improvement

In summary, the confusion matrix suggests that the model performs quite well on this 7-class problem, with some room for improvement, especially concerning the confusion between certain class pairs like class 0 and class 3

4.2.5.3 Confusion Matrix ViT Small

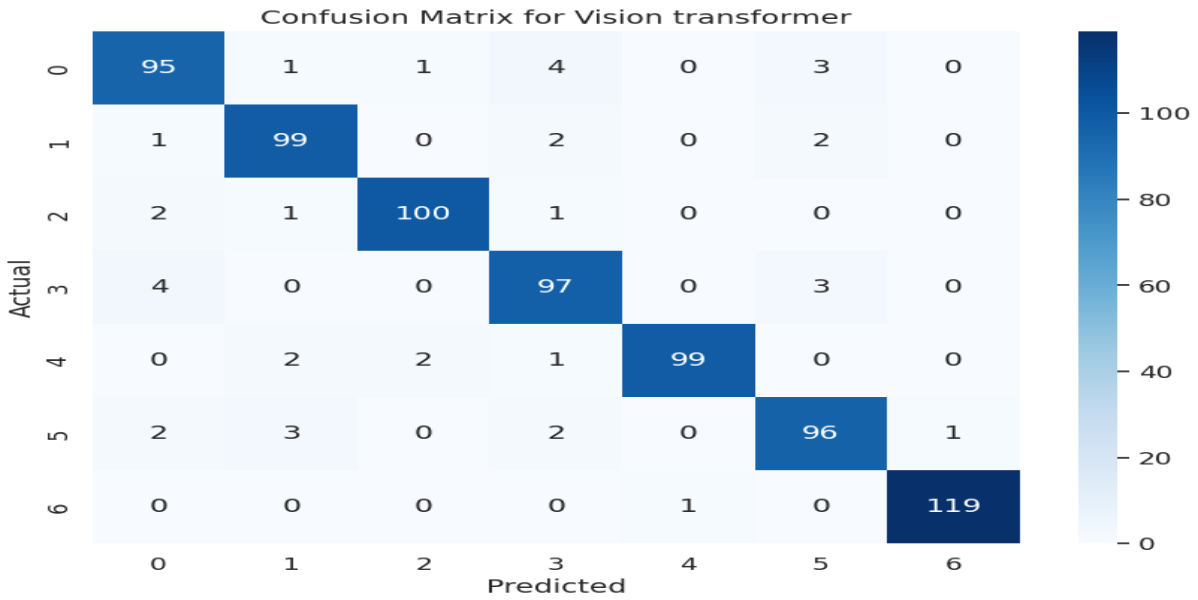


Figure 4.16: Confusion Matrix ViT Small Model 1

4.2.5.4 Confusion Matrix Compact Convolutional Transformer

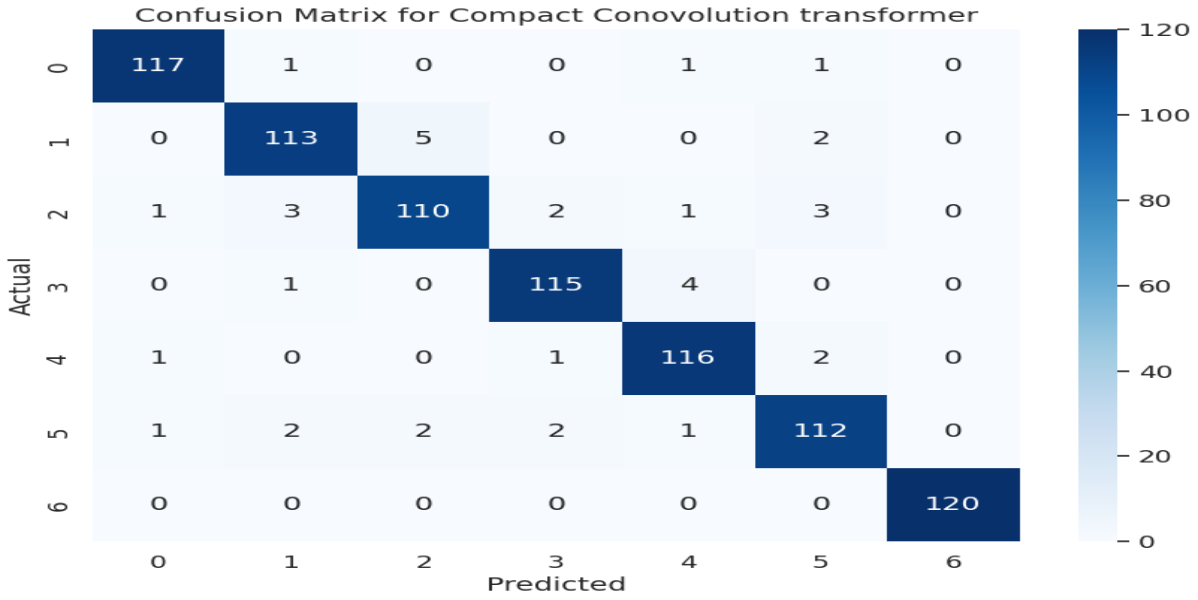


Figure 4.17: Confusion Matrix CCT Model 1

4.2.5.5 Model 2

- Mutual Confusion: Class 0 instances sometimes confuse with Class 1 (15 times) and Class 2 (13 times). Similarly, both Class 1 and Class 2 have instances that are misclassified as Class 0.
- Distinct Classes: Class 1 and Class 2 both stand out with minimal cross-confusion. Class 1 has a distinct 62 true positives with very few instances misclassified as Class 0, and no instances confused with Class 2.
- Overlap Issues: Class 0 has features that possibly overlap with both Class 1 and Class 2, given that it gets misclassified into both these classes more frequently than the other way around.
- Recommendation: Investigate the similarities between Class 0 with both Class 1 and Class 2 to reduce misclassifications.
- General Performance: Most predictions are spot-on, with the primary source of errors stemming from Class 0 misclassifications.

4.2.5.6 Confusion Matrix ViT Model 2

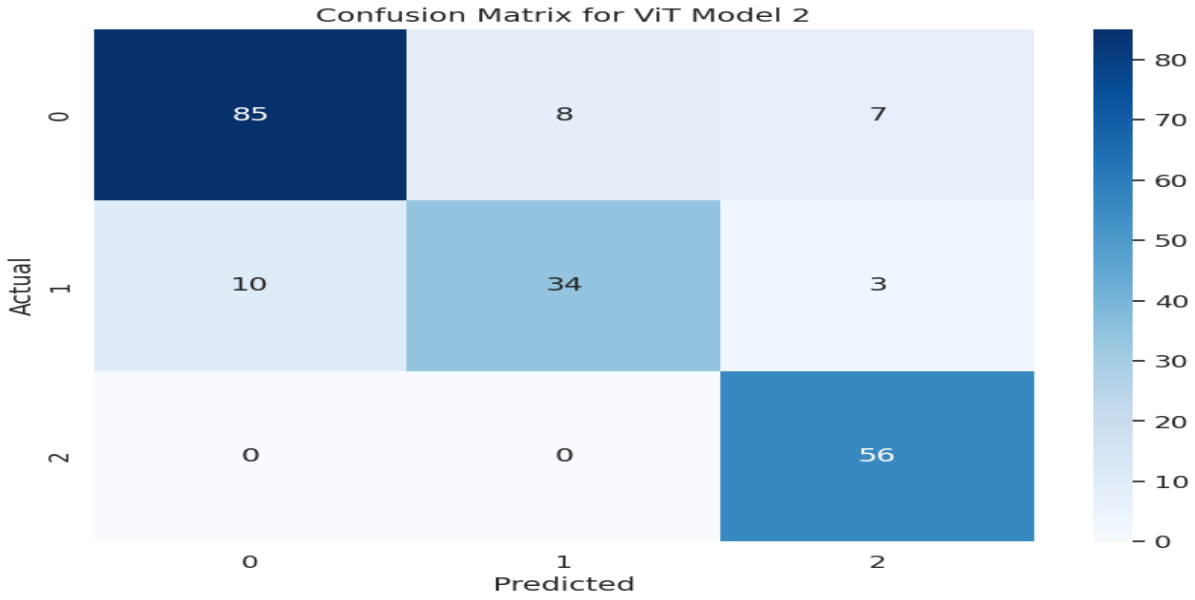


Figure 4.18: Confusion Matrix ViT Model 2

4.2.5.7 Confusion Matrix CCT Model 2

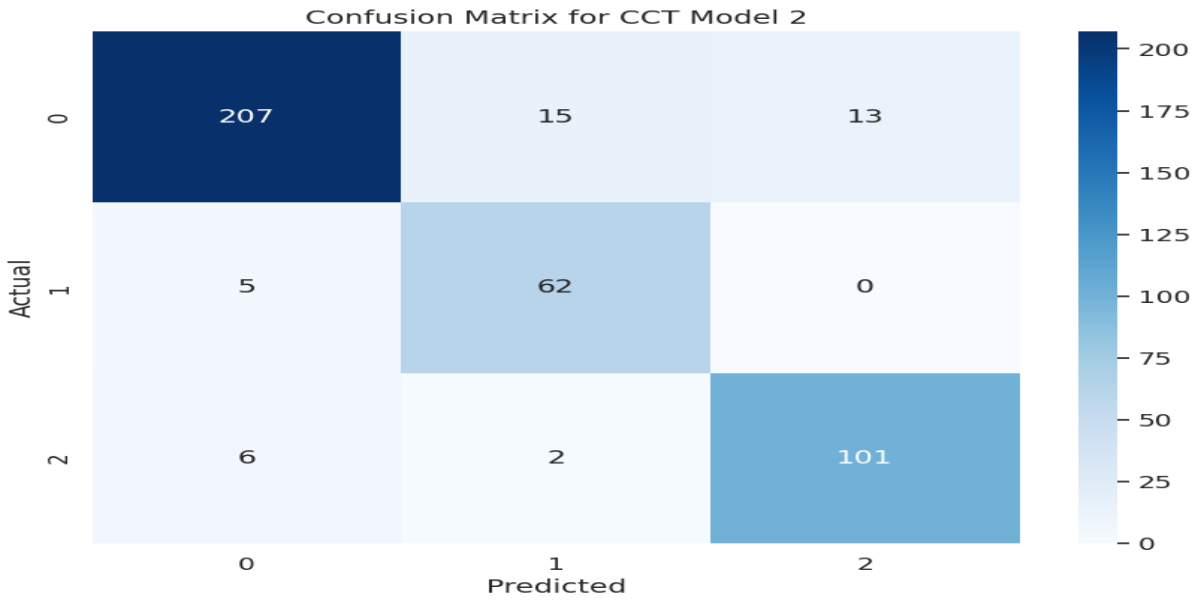


Figure 4.19: Confusion Matrix CCT Model 2

Chapter 5

Conclusions and Future Work

The final chapter of this thesis provides a summary of the research presented and emphasizes its significant contributions. The chapter serves as a conclusion to the study, highlighting the key insights, outcomes and action points.

5.1 Summary

Let us recall the research questions that were put forward at the start. Those questions were

1. How effectively Deep learning models are able to detect cotton disease and classify the intensity of affect in case of Whitefly?
2. Which deep learning model successfully detects the cotton diseases with a high accuracy?
3. Performance of Vision Transformer for the classification of cotton leaf diseases
4. Comparison of results for deep learning models and vision transformers.
5. Comparison of Vision transformers for performance on the dataset

As we conclude our research we will try to answer each one of the question one by one with the help of the results and the findings in the above sections.

5.1.1 Effectiveness of the Deep learning models used

The comprehensive evaluation and fine-tuning of multiple deep learning models have yielded highly compelling results, affirming the reliability and efficacy of these models in detecting cotton diseases. The meticulous selection and implementation of deep learning algorithms in this research have consistently demonstrated remarkable accuracy, with results ranging from a minimum of 86% to an impressive maximum of 96% in certain cases.

5.1.2 Most effective Deep Learning model

The efficacy of these algorithms can be assessed through two distinct perspectives. Firstly, the accuracy in identifying the cotton disease, serves as a primary evaluation criterion. Secondly, the effectiveness of the model in detecting the severity of Whitefly effected leaves. The findings obtained from this study, which focused on various deep learning algorithms, demonstrate that the Compact convolution transformer outperformed others when it comes to accurately detecting cotton diseases, achieving an accuracy rate slightly exceeding 96%. Similarly, in terms of severity detection for Whitefly disease, the CCT yielded the most favorable results, boasting an accuracy rate of nearly 90%. It is important to emphasize that these accuracies were calculated based on the test dataset following model training.

5.1.3 Performance of Vision Transformer

Vision Transformers (ViT) have demonstrated remarkable performance on the dataset in question, outpacing traditional convolution-based models. By leveraging self-attention mechanisms that allow the model to consider the entire image at once, ViT captures long-range dependencies and intricate patterns within the data. This global perspective enables a more comprehensive understanding of the spatial hierarchies present in the images, leading to improved accuracy and robustness. The success of ViT on this dataset underscores its potential as a powerful alternative to conventional architectures, reflecting a significant advancement in

the field of computer vision

5.1.4 Comparison of Results

In the comparative analysis conducted on the specific dataset, various Vision Transformers (ViTs) were pitted against the well-established ResNet architecture. The evaluation focused on key performance metrics, primarily test accuracy, and revealed a noteworthy edge in favor of the ViTs. While ResNet has historically been a strong performer in various computer vision tasks, the self-attention mechanism within ViTs allows them to capture global dependencies and complex patterns more effectively. This nuanced understanding of spatial relations led to higher test accuracy, illustrating the ViTs' ability to generalize better from the training data. The superior performance of Vision Transformers over ResNet in this experiment not only underscores the evolution and adaptability of transformer models but also signifies a potential shift in preferred methodologies for image analysis, encouraging further exploration and adoption of transformer-based architectures in the field of deep learning.

5.1.5 Comparison of Vision Transformer

In the pursuit of solving the intricate problem of cotton disease detection, three distinctive vision transformer architectures were deployed: ViT Base, Swin Transformer, and Compact Convolution Transformer (CCT). Each was methodically tailored and assessed on the specific dataset. While all demonstrated commendable capabilities, the Compact Convolution Transformer emerged as the most efficacious model. The architecture of CCT, which elegantly fuses the benefits of both convolution and transformer layers, enabled a more nuanced understanding of the spatial features within the images. This allowed for a more accurate and robust classification of the diverse disease manifestations present in the cotton samples. The superior performance of the CCT underscores the value of continuous architectural innovation and reaffirms the adaptability of transformer models to specialized domains. This success not only sets a new benchmark for cotton

disease detection but also lays a promising pathway for further exploration and development within the agricultural domain of intelligent machine learning applications

5.2 Conclusion

This research offers valuable insights into the feasibility of employing deep learning methodologies for the identification of cotton diseases, which play a crucial role in the management of agricultural health. It explored the efficacy of Convolutional Neural Networks (CNNs) and a new generation of Vision Transformers for this task. CNN models were honed using transfer learning and tuning techniques, thereby adapting the pre-trained models to the specifics of the cotton disease dataset. Meanwhile, the application of three distinct Vision Transformers—ViT small, Swin Transformer, and CCT—was investigated.

When compared with each other, it was found that Vision Transformers generally outperformed CNN models in terms of F1 score and accuracy. This outcome reinforces the supremacy of transformer-based architectures in the field of deep learning, demonstrating their ability to capture complex patterns in visual data, even in diverse contexts such as plant disease detection. Moreover, it also underscores the potential of transfer learning techniques in increasing model generalization and mitigating overfitting issues.

Within the domain of Vision Transformers, the Cross-Covariance Transformer (CCT) emerged as the most potent model, yielding the highest performance in both F1 score and accuracy metrics. This demonstrates the strength of CCT, with its unique mechanism of leveraging cross-covariance for image classification tasks. The promising performance of ViT small and Swin Transformer should not be understated either; both models showed considerable potential for this application. The results of this research therefore not only provide a concrete direction for the immediate improvement of cotton disease detection but also pave the way for broader investigations into the application of deep learning in agricultural health.

5.3 Future Work

This study opens several exciting avenues. Given the demonstrated superior performance of Vision Transformers, more research is needed to fine-tune these models specifically for the task of cotton disease detection. For instance, we can explore how various architectural modifications or training strategies could potentially enhance their performance. It would also be of interest to study the incorporation of self-supervised learning techniques into Vision Transformers, as they could potentially improve the model's ability to learn meaningful representations from the data.

Beyond this, as the current dataset was developed from public sources and may not fully capture the diversity of cotton diseases, it would be valuable to test and validate these models on larger and more heterogeneous datasets. This could include different types of cotton diseases, different stages of disease progression, and cotton crops grown under a variety of environmental conditions. It would also be insightful to integrate other modalities of data, such as thermal or hyperspectral imaging, to examine if these can contribute to improved performance.

Lastly, since the Cross-Covariance Transformer (CCT) demonstrated the highest performance among the tested models, an in-depth investigation into the workings of CCT is warranted. It would be worthwhile to understand why this particular model performed so well, and how its unique attributes could be further leveraged or enhanced for the task of cotton disease detection. Collectively, these future directions promise to not only advance the specific field of cotton disease detection but also contribute significantly to the broader domain of agricultural health management through deep learning

References

[1] "Typical cnn architecture," Available at ResearchGate. [Online]. Available: https://www.researchgate.net/publication/344012536/figure/fig1/AS:930794704994321@1598930159556/Typical-CNN-Architecture.ppm

[2] "Resnet architecture," Available at Miro. [Online]. Available: https://miro.medium.com/max/1280/1*_XlZtWXLShYsprXVtTyjpQ.png

[3] "Vision transformer," Available online. [Online]. Available: https://www.cloudcraftz.com/wp-content/uploads/2021/04/vit-fig3.png

[4] "Swin transformer architecture," Available at Research Gate. [Online]. Available: https://www.researchgate.net/publication/360004380/figure/fig3/AS:1179981581426689@1658340940131/a-Swin-Transformer-architecture-b-Swin-Transformer-blocks.png

[5] H. Wu, B. Xiao, N. Codella, M. Liu, X. Dai, L. Yuan, and L. Zhang, "Cvt: Introducing convolutions to vision transformers," in *Proceedings of the IEEE/CVF international conference on computer vision*, 2021, pp. 22–31.

[6] M. Khan, A. Wahid, M. Ahmad, M. T. Tahir, M. Ahmed, S. Ahmad, and M. Hasanuzzaman, *World Cotton Production and Consumption: An Overview*, 03 2020, pp. 1–7.

[7] R. Manavalan, "Towards an intelligent approaches for cotton diseases detection: A review," *Computers and Electronics in Agriculture*, vol. 200, p. 107255, 2022. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0168169922005683

[8] S. chohan, R. Perveen, M. Abid, M. Tahir, and D. M. Sajid, *Cotton Diseases and Their Management*, 03 2020, pp. 239–270.

[9] M. Latif, M. Khan, M. Javed, H. Masood, U. Tariq, Y. Nam, and S. Kadry, "Cotton leaf diseases recognition using deep learning and genetic algorithm," *Computers, Materials and Continua*, vol. 69, 08 2021.

[10] C. K. Rai and R. Pahuja, "Classification of diseased cotton leaves and plants using improved deep convolutional neural network," *Multimedia Tools and Applications*, pp. 1–19, 2023.

[11] M. Zekiwos and A. Bruck, "Deep learning-based image processing for cotton leaf disease and pest diagnosis," *Journal of Electrical and Computer Engineering*, vol. 2021, pp. 1–10, 2021.

[12] G. Dhingra, V. Kumar, and H. D. Joshi, "Study of digital image processing techniques for leaf disease detection and classification," *Multimedia Tools and Applications*, vol. 77, pp. 19 951–20 000, 2018.

[13] S. Tripathy, "Detection of cotton leaf disease using image processing techniques," in *Journal of Physics: Conference Series*, vol. 2062, no. 1. IOP Publishing, 2021, p. 012009.

[14] R. Sarwar, A. Muhammad, S. K. Khurshid, T. Ahmed, A. Martinez-Enriquez, and T. Waheed, "Detection and classification of cotton leaf diseases using faster r-cnn on field condition images," *Acta Scientific Agriculture*, vol. 5, pp. 29–37, 09 2021.

[15] M. Zekiwos and A. Bruck, "Deep learning-based image processing for cotton leaf disease and pest diagnosis," *Journal of Electrical and Computer Engineering*, vol. 2021, pp. 1–10, 2021.

[16] G. Yagnesh and M. J. andEbenezer Jangam, "Cotton crop disease detection using swin transformers and attention-based cnn."

[17] A. Dosovitskiy, L. Beyer, A. Kolesnikov, D. Weissenborn, X. Zhai, T. Unterthiner, M. Dehghani, M. Minderer, G. Heigold, S. Gelly *et al.*, "An image

is worth 16x16 words: Transformers for image recognition at scale," *arXiv preprint arXiv:2010.11929*, 2020.

[18] R. Reedha, E. Dericquebourg, R. Canals, and A. Hafiane, "Transformer neural network for weed and crop classification of high resolution uav images," *Remote Sensing*, vol. 14, no. 3, p. 592, 2022.

[19] A. I. Jajja, A. Abbas, H. A. Khattak, G. Niedbała, A. Khalid, H. T. Rauf, and S. Kujawa, "Compact convolutional transformer (cct)-based approach for whitefly attack detection in cotton crops," *Agriculture*, vol. 12, no. 10, p. 1529, 2022.

[20] Y. Bazi, L. Bashmal, M. M. A. Rahhal, R. A. Dayil, and N. A. Ajlan, "Vision transformers for remote sensing image classification," *Remote Sensing*, vol. 13, no. 3, p. 516, 2021.

[21] S. Bhojanapalli, A. Chakrabarti, D. Glasner, D. Li, T. Unterthiner, and A. Veit, "Understanding robustness of transformers for image classification," in *Proceedings of the IEEE/CVF international conference on computer vision*, 2021, pp. 10 231–10 241.

[22] Y. Huo, K. Jin, J. Cai, H. Xiong, and J. Pang, "Vision transformer (vit)-based applications in image classification," in *2023 IEEE 9th Intl Conference on Big Data Security on Cloud (BigDataSecurity), IEEE Intl Conference on High Performance and Smart Computing,(HPSC) and IEEE Intl Conference on Intelligent Data and Security (IDS)*. IEEE, 2023, pp. 135–140.

[23] S. K. Noon, M. Amjad, M. A. Qureshi, and A. Mannan, "Handling severity levels of multiple co-occurring cotton plant diseases using improved yolox model," *IEEE Access*, vol. 10, pp. 134 811–134 825, 2022.

[24] J. Amin, M. Anjum, M. Sharif, S. Kadry, and J. Kim, "Explainable neural network for classification of cotton leaf diseases," *Agriculture*, vol. 12, p. 2029, 11 2022.

[25] A. Hassani, S. Walton, N. Shah, A. Abuduweili, J. Li, and H. Shi, "Escaping the big data paradigm with compact transformers," 2021. [Online]. Available: https://arxiv.org/abs/2104.05704

[26] H. Wu, Y. Zhang, K. Huang, Y. Dong, and B. Lin, "Cvt: Introducing convolutions to vision transformers," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2021, pp. 2203–2213.