Real-Time Plant Leaf Disease Detection and Identification using Convolutional Neural Networks on an Embedded Platform

Major Project Report

Submitted in partial fulfillment of the requirements

for the degree of

Master of Technology

in

Electronics & Communication Engineering

(Embedded Systems)

By

Stavan Ruparelia

(18MECE14)

Electronics & Communication Engineering Department

Institute of Technology, Nirma University

Ahmedabad-382 481

May 2020

Real-Time Plant Leaf Disease Detection and Identification using Convolutional Neural Networks on an Embedded Platform

Major Project Report

Submitted in partial fulfillment of the requirements

for the degree of

Master of Technology

in Electronics & Communication Engineering

By

Stavan Ruparelia

(18MECE14)

Under the guidance of

Project Guide:

Dr.Ruchi Gajjar

Assistant Professor

Electronics & Communication Engineering Department

Institute of Technology,Nirma University

Ahmedabad-382 481

May 2020

NIRMA UNIVERSITY

INSTITUTE OF TECHNOLOGY

NAAC ACCREDITED 'A+' GRADE

Certificate

This certifies that **Stavan Ruparelia (18MECE14)** has successfully completed the thesis project entitled "**Real-Time Plant Leaf Disease Detection and Identification using Convolutional Neural Networks on an Embedded Platform**" with an outstanding grade of **A+** and has published a journal paper titled "**Real-time detection and identification of plant leaf diseases using convolutional neural networks on an embedded platform**" under the mentorship of **Dr. Ruchi Gajjar,** Assistant professor at Electronics & Communication Engineering Department, Institute of Technology, Nirma University, Ahmedabad, Gujarat, India.

INSTITUTE OF TECHNOLOGY AHMEDABAD NIRMA UNIVERSITY

Dr. Nagendra Gajjar
Program Coordinator,
Professor,
Electronics & Communication Engineering Department
Institute of Technology,
Nirma University, Ahmedabad.

NIRMA UNIVERSITY
INSTITUTE OF TECHNOLOGY
NAAC ACCREDITED 'A' GRADE

Certificate

This is to certify that the Major Project entitled **"Real-Time Plant Leaf Disease Detection and Identification using Convolutional Neural Networks on an Embedded Platform "** submitted by **Stavan Ruparelia (18MECE14)**, towards the partial fulfillment of the requirements for the degree of Masters of Technology in Embedded Systems, Nirma University, Ahmedabad is the record of work carried out by him under our supervision and guidance. In our opinion, the submitted work has reached a level required for being accepted for examination. The results embodied in this Project, to the best of our knowledge, haven't been submitted to any other university or institution for award of any degree or diploma.

Date:

Place: **Ahmedabad**

Dr. Ruchi Gajjar
Project Guide,
Assistant Professor,
Institute of Technology,
Nirma University, Ahmedabad.

Dr. Nagendra Gajjar
Program Coordinator,
Professor,
Institute of Technology,
Nirma University, Ahmedabad

Dr. Dhaval Pujara
Professor and Head,
EC Department,
Institute of Technology,
Nirma University, Ahmedabad.

Dr. R N Patel
Director,
Institute of Technology,
Nirma University, Ahmedabad

Declaration

This is to certify that

1. The thesis comprises my original work towards the degree of Master of Technology in Embedded Systems at Nirma University and has not been submitted elsewhere for a degree.

2. Due acknowledgment has been made in the text to all other material used.

- Stavan Ruparelia

18MECE14

Certificate

This is to certify that the Major Project entitled **"Real-Time Plant Leaf Disease Detection and Identification using Convolutional Neural Networks on an Embedded Platform "** submitted by **Stavan Ruparelia (18MECE14)**, towards the partial fulfillment of the requirements for the degree of Masters of Technology in Embedded Systems, Nirma University, Ahmedabad is the record of work carried out by him under our supervision and guidance. In our opinion, the submitted work has reached a level required for being accepted for examination. The results embodied in this Project, to the best of our knowledge, haven't been submitted to any other university or institution for award of any degree or diploma.

Date: Place: **Ahmedabad**

Dr. Ruchi Gajjar
Project Guide,
Assistant Professor,
Institute of Technology,
Nirma University, Ahmedabad.

Dr. Nagendra Gajjar
Program Coordinator,
Professor,
Institute of Technology,
Nirma University, Ahmedabad

Dr. Dhaval Pujara
Professor and Head,
EC Department,
Institute of Technology,
Nirma University, Ahmedabad.

Dr. R N Patel
Director,
Institute of Technology,
Nirma University, Ahmedabad

Statement of Originality

I, **Stavan Ruparelia**, Roll.No. **18MECE14**, give undertaking that the Major Project entitled **Real-Time Plant Leaf Disease Detection and Identification using Convolutional Neural Networks on an Embedded Platform** submitted by me, towards the partial fulfillment of the requirements for the degree of **Master of Technology in Electronics and Communication (Embedded Systems)** of Institute of Technology, Nirma University, Ahmedabad, contains no material that has been awarded for any degree or diploma in any university or school in any territory to the best of my knowledge. It is the original work carried out by me and I give assurance that no attempt of plagiarism has been made. It contains no material that is previously published or written, except where reference has been made. I understand that in the event of any similarity found subsequently with any published work or any dissertation work elsewhere; it will result in severe disciplinary action.

Signature of Student

Date:

Place: **Ahmedabad**

Signature of Guide

Endorsed by

Dr.Ruchi Gajjar

Acknowledgements

I am thankful to the Almighty God for giving me strength. I feel very fortunate to receive blessings from my Guru **H.D.H Hariprasad Swami Maharaj** who has always inspired me in every endeavour of my life. I would like to take this opportunity to thank him and extend my utmost gratitude unto his feet.

I would also like extent my gratitude to those who have helped me throughout my project work and presentations. First of all, I would like to thank my project Guide **Dr. Ruchi Gajjar** for providing valuable guidance. Moreover, I am thankful for her moral support as well as constant encouragement which was most cardinal part for this project to work successfully.

Also, I would like to thank to **Dr.N.P.Gajjar**, PG Coordinator of M.Tech Embedded Systems, for his guidance during the Project. I appreciate him for his technical support and motivation throughout the entire project period.

I also thank **Dr.Dhaval Pujara**, Hon'ble Head of Electronics and Communication Engineering Department, Institute of Technology, Nirma University for providing me the required facility to carry out this project work.

In addition, I wish to thank my parents, family members for their constant support and encouragement during this project work. I am also thankful to all my friends for their motivation and every possible help.

- Stavan Ruparelia

18MECE14

Contents

List of Figures

List of Tables

Abstract

Damage in yield due to crop diseases is a great concern to the farmers and India being an agriculture-based country, a large portion of the country's GDP also depends on it. Loss of yield due to undetected or untimely dispersion of pesticides affects the overall crop production. In this project, a method to identify the type of disease present in a crop based on leaf images using machine learning is proposed. First, the leaves are individually detected in real-time from the field using Single Shot Detector (SSD). To classify the type of disease present in the crop, a Convolutional neural networks architecture is proposed which is trained on the PlantVillage dataset and the proposed hybrid network is deployed on the embedded platforms namely NVIDIA Jetson TX1 and NVIDIA Jetson Nano, for real-time detection and identification. The disease classification accuracy achieved is around 96.88%. Moreover, the proposed Classification model's accuracy is compared with the AlexNet model's accuracy. As a result, the proposed classification model accuracy is higher than AlexNet CNNs model accuracy, which is 95.53%. Furthermore, the proposed plant disease detection system also tested at a tomato farm. This in-field real-time testing shows that the proposed model is robust and gives efficient results in real-time testing.

This project aims to aid the farmers by embedding the machine learning model for detection and classification in a handheld device that would inform the farmers about the disease type on time, thereby providing them sufficient help to take precautions and countermeasures for its removal.

Abbreviations

AI	Artificial Intelligence
ANN	Artificial Neural Network
API	Application Programming Interface
CNN	Convolutional Neural Network.
CNNs	Convolutional Neural Networks.
CSV	Comma Seperated Values
CPU	Central Processing Unit
DNN	Deep Neural Networks
FPS	Frames Per Second
GPU	Graphical Processing Unit
GDP	Gross Domestic Product
ML	Machine Learning
mAP	Mean Average Precision
ReLU	Rectified Linear Unit
R-CNN	Region-Convolutional Neural Networks
R-FCN	Region-based Fully Convolutional Network
SVM	Support Vector Machine
SSD	Single Shot Detector
TPU	Tensor Processing Unit
TF	TensorFlow
YOLO	You Only Look Once

Chapter 1

Introduction

1.1 Motivation

Agriculture plays a cardinal role in the global economy. India is well known for its agriculture and around sixty percent of the population depends on it. Now a Days, the majority of crop vegetation fails as disease detection of a crop does not effectively come under farmer's harvesting methods. Thus, there is a requirement of an automated system that can predict the disease in crops before the entire harvest gets damaged.

Machine learning can be used for plant disease detection and can help farmers to identify the disease in crops. This project aims to develop a real time plant disease detection system using the concept of a deep learning. The method aims to predict plant diseases efficiently. So, farmers can be able to effectively take counter measures before the damage spreads in the crops.

Figure 1.1: Indian farm

1.2 Objective of Dissertation

The main objectives of the project are as follows:

- Analysis and exploration of different plant disease image datasets.
- Real-time plant leaf detection using trained SSD MobileNet.
- Real-time plant leaf disease identification using proposed trained convolutional neural networks (CNNs) architecture.
- Deployment of the developed Machine Learning models on different embedded platforms for inference.

1.3 Problem Statement

Effective detection of leaves on the plant and then accurate identification of the type of disease present on the leaf is the main aim of the project. The plant leaf detection was done using the SSD model as well as leaf disease identification was done using proposed CNNs architecture. The SSD model and proposed CNNs are combined to make a hybrid model which is capable to detect leaf and identify diseases at the same instance of time. In addition to this, for real-time testing, this proposed hybrid model is deployed on various embedded platforms to solve the problem of detection plant leaf disease in real-time.

1.4 Project Scope

In this project, the system approach is divided into two major parts - plant leaves detection and disease classification using machine learning. Firstly, this proposed system detects the location of leaves on the plant with good accuracy then accurately predicts the disease on that leaf. As seen in Figure 1.2, the proposed method will predict the disease class based on the risk of a disease and its position in the image displayed as a bounding box containing the plant's infected region.

The model can detect the diseased plant also with real-time data taken from a farm. For carrying out the above procedure on-field, the system is deployed on the em-

bedded platform with minimum requirements to execute a machine learning model onto it.

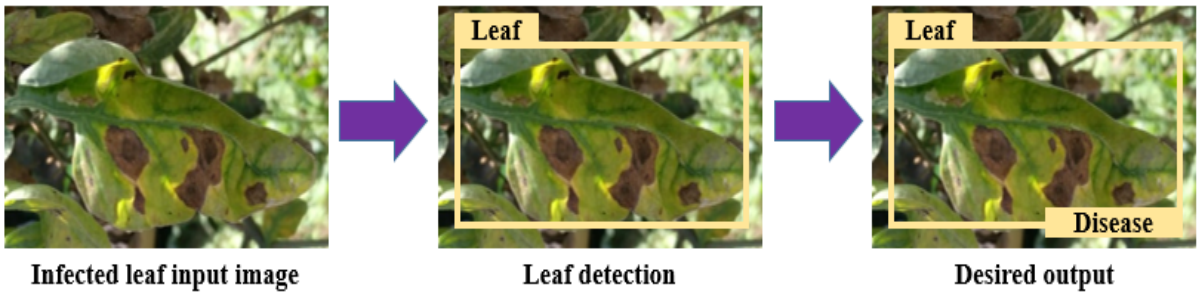

Figure 1.2: System Flow of plant leaf detection and disease classification

1.5 Gantt Chart

Figure 1.3 presents the Gantt Chart of this project.

Project Timeline		2019	2019	2019	2019	2020	2020	2020	2020
No.	Tasks	September	October	November	December	January	February	March	April
1.	Learn Basics of Machine Learning	Stage 1							
2.	Literature Study, Choice Of Algorithm		Stage 2						
3.	Algorithms Deploy on Hardware			Stage 3					
4.	Visit in Farm with Developed Model					Stage 4			
5.	Algorithms Deploy on another Embedded Hardware						Stage 5		
6.	Thesis & Research paper Writing							Stage 6	
		Completed							

Figure 1.3: Project Work-Flow

1.6 Thesis Outline

The thesis consists of six subsequent chapters that deal with:

- **Chapter-1** This chapter describes the brief information about motivation and objective of the project along with the Gantt charts which is indicating project development workflow. Moreover, it illustrates the project overview in brief.

- **Chapter-2** In the second chapter, the literature review of plant disease detection techniques and approaches available in literature is done. Also a brief summary is presented.

- **Chapter-3** The third chapter describes the proposed methodology for plant leaves detection and disease classification. Moreover, it briefly describes the machine learning libraries used in the implementation of the project.

- **Chapter-4** In the fourth chapter, details on the dataset used and embedded hardware specifications are described.

- **Chapter-5** The fifth chapter showcases experimental results of plant leaf detection, disease classification, and real-time in-field testing. Also, the experimental parameters involved in the proposed structure are described in this chapter.

- **Chapter-6** Finally, in the sixth chapter, the work carried out during this project is summarised and the knowledge gained is discussed to draw some important conclusions. In addition, various topics that need to be explored in the future are discussed.

Chapter 2

Literature Survey

For centuries, food losses due to crop infections from pathogens such as bacteria, viruses, and fungi is a vital issue that needs to be addressed across the globe. So, to ensure agricultural sustainability, crop disease detection at an early stage can drastically reduce economic losses. There are several methods to detect disease based on where it has occurred in a plant like Leaf, Node, or Stem. To detect diseased plants infected on the leaf there are two methods: Direct and Indirect methods of detection [5].

Direct methods or laboratory-based techniques are Polymerase Chain Reaction (PCR), Immuno Fluorescence (IF), Fluorescence In-Situ Hybridization (FISH), Enzyme-Linked Immuno Sorbent Assay (ELISA), Flow Cytometry (FCM) and Gas Chromatograp hy-Mass Spectrometry (GC-MS). They require large numbers of data set and gives the most accurate result. The only limitation while applying these methods are they are time-consuming. With accuracy, these methods provide information about the main reason that causes the disease with their concentration and severity [6].

Apart from the direct method of detecting the disease, plant stress profiling and volatile profiling can be used for identifying the pathogenic disease as well as the biotic and abiotic stress levels in crops. This method uses optical sensors that provide a large amount of information in the form of electromagnetic spectra which can be processed by various methods to predict plant health. Some of the known indirect methods are thermography, fluorescence imaging, hyperspectral imaging, and gas chromatography.

With rapidly changing technical areas, the involvement of artificial intelligence (AI) in agriculture and vegetation can contribute to sustainable development, and also efficient uses of the resources are required for the same. The evolution of AI has taken place so rapidly that in less than a century, AI has found its application in almost all fields. There are two methods of detecting disease on leaf namely segmentation and machine learning. With the popularity of machine learning in every field, and various algorithm namely convolutional neural networks (CNN), Support Vector Machine (SVM), K-means clustering, machine learning approach proves to be an efficient way in the detection of diseases in the plant.

2.1 Plant Disease Classification using Machine Learning Algorithms

Geetharamani G.A and **Arun Pandian J.** [7] have proposed a model that can achieve 96.46% accuracy in classification for plant leaf disease using deep Convolutional Neural Network (Deep CNN). The proposed model is trained using 39 different classes with six data augmentation method namely: Image Flipping, Gamma Correction, Noise Injection, Principal Component Analysis (PCA) color augmentation, Rotation, and Scaling. This model was also tested concerning its consistency and reliability. It concludes the maximum pooling method is better than the average pooling method in a CNN. Furthermore, they increased the image data from 49,598 to 55,636 using data augmentation.

Authors in paper [8] have proposed a new model named INAR-SSD (SSD with Inception module and Rainbow concatenation) that uses deep CNN along with GoogLeNet Inception and Rainbow concatenation to detect disease from apple leaves. The proposed model was trained with a data set of 26,377 images of a disease apple leaves. Experimental results showed a high-performance solution with 78.80% mAP and a detection speed of 23.13 FPS. It concludes that the proposed INAR-SSD model provides higher accuracy and faster detection speed when compared to the previous model like Faster R-CNN.

Xihai Zhang et al. [9], the paper aims to reduce the number of parameters and

to improve the identification of maize leaf disease using improved GoogLeNet and Cifar10 models. These four models test and train nine kinds of maize leaf images by altering a few parameters such as adding dropouts, altering the pooling combination, and some linear functions. In this process, GoogLeNet was able to give an average identification accuracy of 98.8% and Cifar10 achieves the average accuracy of 98.8%. It also shows that altering the parameters like the addition of activation function, dropout operations, and few pooling combinations can improve the accuracy of the model. This algorithm can be implemented on a mobile device for better judgment of plant disease.

Konstantinos P. Ferentinos [10] investigates various CNN model on the open database of 87,848 images, containing 25 different plants in a set of 58 distinct classes of combination, including healthy plants. The paper investigates various architectures such as AlexNet, AlexNetOWTBn, VGG, GoogLeNet, and Over-feat with parameters such as batches, epochs, size, momentum, weight decay, and learning rate. The most successful rates were 99.53% observed in VGG. **Jihen Amara et al.,** [11] have proposed a model based on LeNet architecture and successfully detected banana plants diseases. In this work, they used Convolutional neural networks (CNNs) with multi-layers convolution network. In Conclusion, they detect two famous banana disease namely Sigatoka and Banana speckle.

According to [12] paper study, they proposed a novel rice disease identification method based on deep convolutional neural networks (CNNs) techniques, trained on around 500 images. Moreover, under the 10-fold Cross-validation strategy, the proposed CNNs-based model achieves higher accuracy. Also, this paper concluded that using of this proposed CNN's-based the model achieves an accuracy of 95.48%. This accuracy is much higher than the conventional machine learning model.

Alvaro Fuentes et al., [13] presented a deep-learning-based approach to detect diseases and 5 pests in tomato plants using images captured in-place by the camera with different resolutions. This paper's goal was to find a more suitable architecture for this task. They worked with three major detector groups, namely the Faster Region-based Convolutional Neural Network (Faster R-CNN), the Region-based Fully Convolutional

Network (R-FCN), and the Single Shot Multibox Detector(SSD), which are named "deep learning meta-architectures" for this work. These meta-architectures are combined with "deep feature extractors," such as VGGNet and Residual Network (ResNet). In fact, a confusion matrix of the final detection effects is also available. To conclude, this detector was applied photos collected by cameras, instead of following the method of gathering and examining physical objects (leaves, plants) in the laboratory. In fact, they observed that improved output effects were obtained by the usage of data classification and augmentation techniques. Table 2.1. provides the analysis of the various machine learning approaches employed for disease detection in different crops by examining leaves of that plant.

Table 2.1: Literature analysis of various architectures for plant disease identification

Sr. No	Plant Species	Model Architectures	Outcomes	Reference
1.	Rice	SVM	87.9 % accuracy	[14]
2.	Green citrus	SVM	80.4 % accuracy	[15]
3.	Strawberry	SVM	MPE = 2.25%	[16]
4.	Tomato	Clustering/EM	Recall: 0.6066 Precision: 0.9191 F-Measure: 0.7308	[17]
5.	Cotton	SVM	95.00 % Classification accuracy	[18]
6.	Grape Tomato Orange Apple	NASNet as CNN	93.82 % accuracy	[19]
7.	Guava	AlexNet as CNN	98.74 % average accuracy	[20]
8.	Tomato	CNNs/LVQ	86.00 % average accuracy	[21]
9.	Corn	Proposed CNN architecture	97.09 % accuracy	[22]
10.	Tomato Corn Apple	CNNs	89.83 % accuracy	[23]

2.2 Crop disease Classification Detection using Machine Learning algorithms on Embedded Hardware

Crop disease detection work also done with the use of embedded hardware (Inference Engine). Here, it is used as an inference engine for real-time output.

Halil Durmus and **Ece Olcay Güne** [24] deployed a convolutional neural network on hardware successfully. They used two distinct architectures of deep learning: AlexNet and SqueezNet to identify diseases in fields of tomato plants. This algorithm is implemented on a robot in real-time to detect the diseases on the farm as well as in Greenhouse. The architectures used were trained and tested on the tomato images from the plant village. Using GPU the network was validated on Nvidia Jetson Tx1. The paper compares the architectures AlexNet and SqueeZeNet and concludes AlexNet performed better than SqueeZeNet. It is intelligibly seen that the SqueeZeNet model is nearly 80 times smaller than AlexNet. Models size is taken directly from Cafe model files and the inference time is varying in different tests by only 5 milliseconds.

Authors in paper [25] have proposed one real-time embedded system for Apple fruit detection. In this proposed system they used the YOLOv3-tiny algorithm. Moreover, this proposed system deployed on different embedded platforms namely Raspberry Pi 3 B+ in combination with Intel Movidius Neural Computing Stick (NCS), Nvidia Jetson Nano, and Jetson AGX Xavier for inference. The proposed architecture model is capable to detect small objects. In real-time, this deployment model achieved 83.64% average accuracy as well as a frame rate of up to 30 fps even for the difficult scenarios. The proposed system is deployed on Robert's vehicle to detect, count, and measure the size of the apples in real-time.

2.3 Literature Review Summary

Thus, based on this literature survey, it is observed that Machine Learning approaches are a convenient and efficient method to detect disease present in a leaf of a plant. For the present thesis work, a CNN architecture is proposed for identify plant

leaf disease as well as SSD MobileNet model architecture is used for leaves detection. Moreover, Tensorflow deep learning library is used for image detection. Besides this, the proposed system is deployed on various embedded platforms namely Nvidia Jetson TX1 and Nvidia Jetson Nano for inference and real time in-field testing.

Chapter 3

Proposed Method

The third chapter describes the proposed architecture to identify plant leaves and diseases in it. Apart from this, the technical principles of CNN and SSD versions are discussed in brief. Deep learning libraries such as Tensorflow and Keras are also described in this chapter.

3.1 Proposed System Overview

A system for plant leaf disease identification has been proposed in this project work. The developed system approach involves the two fundamental levels - plant leaves detection and then the disease classification on those leaves. The block diagram of the proposed system for leaf identification and disease classification is shown in Figure 3.1.

Before initiating the training and testing process of the proposed architectures, it was important to collect leaf images for training the detector model as well as it required a large dataset of infected images for the disease classification model. For leaf detection, a dataset of 500 images was collected, consisting of images from manually clicked leaf images with different orientation and size at different time as well as images from the internet. For disease classification, images from PlantVillage dataset are used which contained a total of 21,948 leaf images.

Each of the leaf images are manually annotated for leaf detection and used as input to train the SSD object detection model and a CNN architecture is proposed for

the identification of crop disease from images of the leaves. The healthy and infected leaf images are pre-processed for this leaf classification model and then fed to the proposed CNN model.

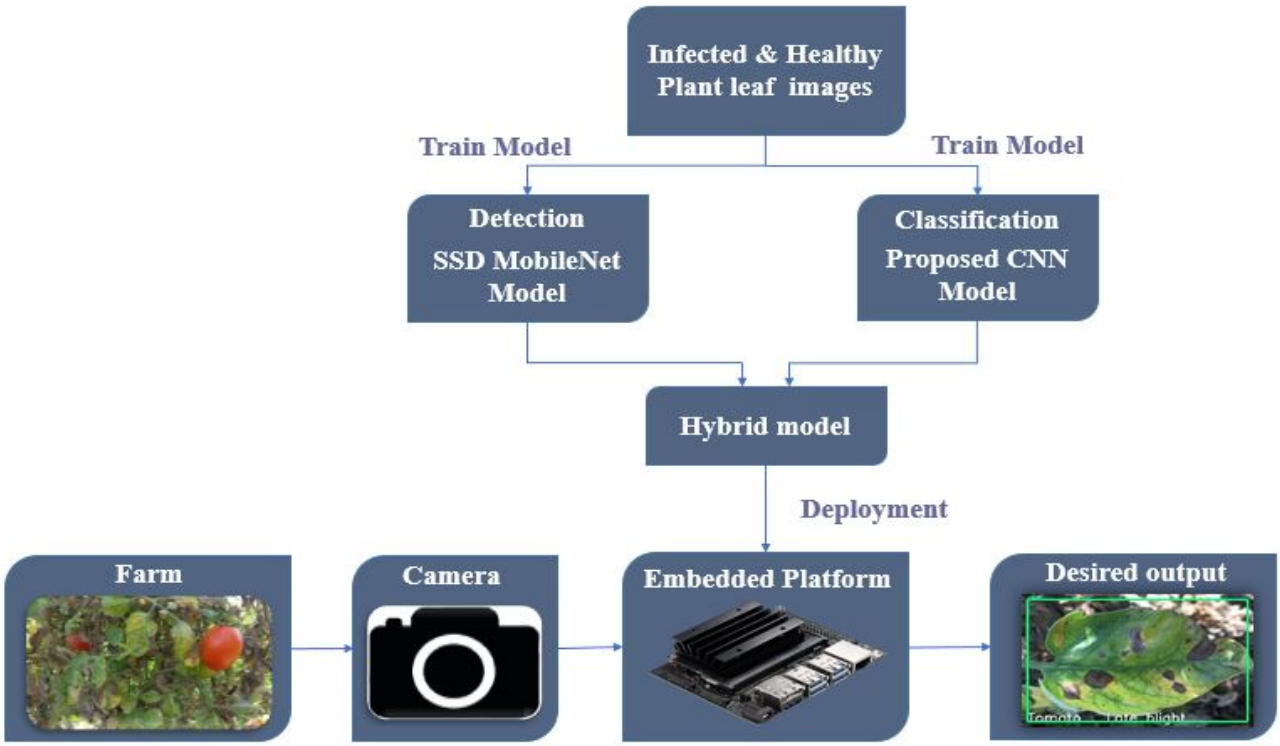

Figure 3.1: Proposed system block diagram of crop disease detection

Next, a hybrid model is combines both, the trained leaf detection and disease classification models. This proposed hybrid model was deployed for inference on various embedded platforms. Additionally, one camera module connected to embedded hardware, which feeds input from the field to the proposed hybrid model for real-time identification of crop disease in the field.

Hence, proposed model first detects the leaf on the plant and then discriminates it as healthy leaf or as diseased leaf, identifying the type of disease, with help of real time data from the camera connected through the hardware.

3.2 Leaf Detection Model

3.2.1 Introduction to Object Detection

Object detection is a generic concept to identify a set of computer vision-based activities concerning the identification of objects in digital images. Classification of images requires predicting the class of a particular entity in an image. Localization of objects involves defining the position of one or more items in an image and drawing an ample box around its length. Object recognition integrates these two tasks and locates one or more objects in an image, and classifies them.
There are three primary architectures to identify objects: [26].

1. Faster Region-based Convolutional Neural Network (Faster R-CNN)

The detection method in Faster R-CNN is performed in two stages [27]. A Region Proposal Network (RPN) takes an image as input in the first stage and processes it by an extractor of the features. Intermediate-level features are used to predict propositions of objects, each with a score. To train the RPNs, the device finds anchors that hold an object or not, depending on the Intersection-over-Union (IoU) between the proposals for the object and ground-truth.

In the second stage, the previously created box proposals are used to crop characteristics from the same feature map. Consequently, those cropped features are fed into the remaining layers of the feature extractor to predict the class likelihood and bounding box for each proposed area. The whole process operates on a single distributed network, enabling the device to share full-image convolution functionality with the detection network, making for almost cost-free regional proposals. On all the tested instances, faster R-CNN utilizing Inception Resnet with 300 propositions provides the best accuracy at 1 FPS [28].

2. You Only Look Once (YOLO)

YOLO uses a totally different approach. YOLO is a clever convolutional neural network (CNN) for doing object detection in real-time. The algorithm applies a single neural network to the full image, and then divides the image into regions and predicts

bounding boxes and probabilities for each region. These bounding boxes are weighted by the predicted probabilities.

YOLO is popular because it achieves high accuracy while also being able to run in real-time. The algorithm "only looks once" at the image in the sense that it requires only one forward propagation pass through the neural network to make predictions. After non-max suppression (which makes sure the object detection algorithm only detects each object once), it then outputs recognized objects together with the bounding boxes.

With YOLO, a single CNN simultaneously predicts multiple bounding boxes and class probabilities for those boxes. YOLO trains on full images and directly optimizes detection performance.

3. Single-short Multi-box Detector (SSD)

The SSD model architecture tackles the object recognition issue by utilizing a convolutional feed-forward network that creates a fixed-size array of bounding boxes and scores for the inclusion of an object type in each box. This network will accommodate objects of various sizes by integrating predictions from multiple feature maps and specific resolutions. In addition, SSD encapsulates the mechanism into a single network, eliminating the generation of proposals and thereby saving time on the computation. SSD has the highest mAP on MobileNet among the models targeted for real-time processing [29].

For this project, SSD is used as an object detector to detect the leaves of the plant from an input image or video as well as in real time in the field.

3.2.2 Single-short Multi-box Detector (SSD)

The Single Shot Detector (SSD) is one of the first attempts to use the pyramidal feature hierarchy of the convolutional neural network to accurately identify objects of various sizes. SSD uses the pre-trained VGG-16 platform on ImageNet as its base code for extracting useful image functions [28]. In addition to VGG16, SSD introduces multiple layers of through sizes to conv functionality. They can be shown in various sizes

as a pyramid representation of the pictures. Intuitively large fine-grained feature maps are excellent for detecting tiny objects at an earlier stage, and small coarse-grained feature maps can identify large objects well. In SSD, tracking happens in each layer of the pyramid, addressing items of varying sizes.

The SSD object detection composes of 2 parts:

- Extract feature maps
- Apply convolution filters to detect objects

In SSD, one-shot only needs to be taken to detect several objects within the image, whereas regional proposal network (RPN)-based approaches such as R-CNN series require two shots, one to produce area proposals, one to detect the object of each proposal. SSD is therefore much faster than other versions used for target detection [30].

The SSD solution is focused on a feed-forward convolutional network that generates a fixed-size set of bounding boxes and scores for the inclusion of object type instances in such boxes, preceded by a non-maximum deletion stage to generate the final detections. The early layers of the network are based on a common design used for classifying high-quality images. Additional networks are then added to create enhanced functionality [31]. Upon the completion of the truncated base network, convolutional feature layers are then applied. Such layers are gradually through in size and enabling multi-scale predictions of detection. Using a series of convolutional filters, each applied function layer will generate a defined collection of detection predictions. The basic item for predicting possible detection parameters for a feature layer of size m x n with p channels is a 3 x 3 x p small kernel which produces either a score for a group or a shape offset relative to the default box coordinates. At each of the m x n locations where the kernel is applied, it produces an output value.

The values of the bounding box offset output are calculated against a default box position relative to every location of the feature map. For several feature maps at the top of the network, a collection of default bounding boxes is connected with every feature map node. The default cells tile the function map in a convolutional fashion, such

that each box's location relative to its corresponding cell is set. The map predicts the offsets concerning the default box shapes in the cell for each function grid row, as well as the per-unit scores that signify the location of a unit instance in each of those boxes. The architecture of the SSD model is shown in figure 3.2.

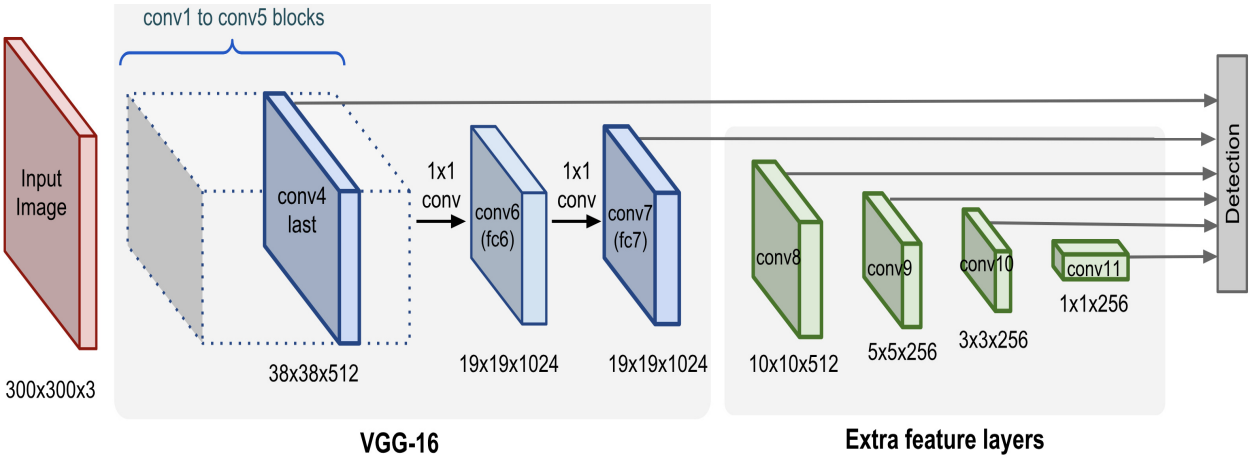

Figure 3.2: The model architecture of Single Shot MultiBox Detector (SSD) [1]

The main difference between training SSD and training a standard detector using regional proposals is that different outputs in the defined range of detector outputs need to be allocated to ground-truth knowledge. Most of the default boxes are negative particularly when there is a large number of potential default boxes. Instead of selecting all the negative cases, they are sorted using the highest confidence loss for each default box and select the top ones such that the ratio between the negatives and positives is at most 3:1. This leads to faster optimization and efficient preparation [29].

3.2.3 Tensorflow Object Detection API

The Tensorflow Object Detection API allows identification of objects through the use of pre-trained object detection models. Object detection API workflow is shown in figure 3.3. Using this API to build one's own leaf detector. Custom object detection model needs to be trained according to the following steps:

1. Collecting Data

2. Labelling Data

3. Creating TFRecords files for Training

4. Configuring Training

5. Training Model

6. Exporting Inference Graph

7. Testing Object Detector

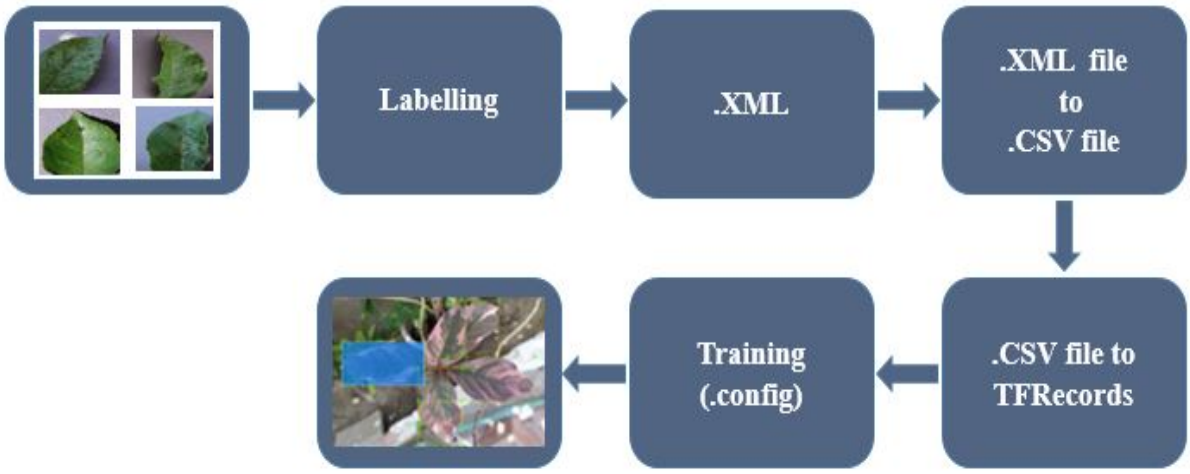

Figure 3.3: Object detection API workflow

Collecting data

Large amount of data is needed to train the classifier. Tensorflow need hundreds (thousands is better) of images of various background to train a model with a good precision.

Labelling data

In this work, labelImg software was used to label our data. LabelImg [32] is a fantastic image tagging tool, and is freely available. Labelling is a graphical image annotation method and the bounding boxes for objects in images. Figure 3.4 displays the LabellImg App window with illustration of the label leaf.

The XML (Extensible Markup Language) file for that specific image would be generated automatically after the bounding box is formed around the labelling leaf. This

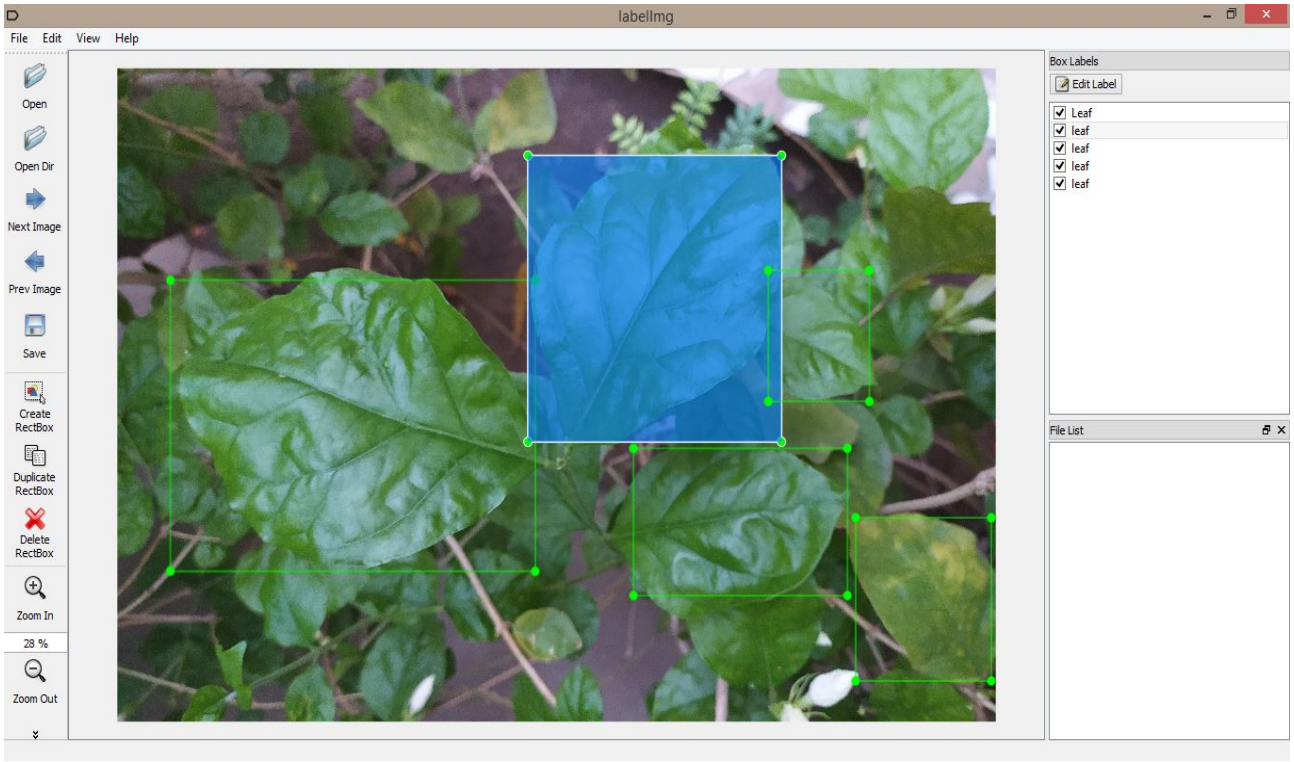

Figure 3.4: Labelling leaf images using LabellImg Software

process is performed with all images that are available in our dataset. It took around 20 hours for all the leaves in our data collection to be marked.

Generating TFRecords for training

With the images labeled, created the TFRecords which will be served as input data for the training of the object detector. To create the TFRecords, two TensorFlow object detection API scripts namely the xml_to_csv.py and generate_tfrecord.py files are used. The first file will convert all the XML data generated from the labelimg software to the CSV (Comma Separated Values) format and the other script will convert the CSV file to TFRecords.

Configuring training

Before training, the last thing we need to do is create a label map and a training configuration script.

Creating a label map

The label map maps an id to a name. The label map for our detector can be seen below. The id number of each item should match the id of specified in the generate_tfrecord.py file.

```
item {
    id: 1
    name: 'leaf'
}
```

Figure 3.5: Label mapping

Creating a training configuration

In this task, the SSD MobileNet platform has been chosen in our detector since it comprises approximately 3 M parameters and would be useful for distributing the trained model for real-time deployment to an embedded device or android application. In this step, we configured the .config file for the SSD MobileNet which will be used further for the training purpose.

Training of the model

In this step, we have to start the training of our model using the our model training script train.py from the TensorFlow Object Detection API. We trained our model for almost 75,000 steps which almost took 20 hours for the training purpose and achieved the minimum loss of around 1.47. The loss vs steps graph seen in Figure 3.6.

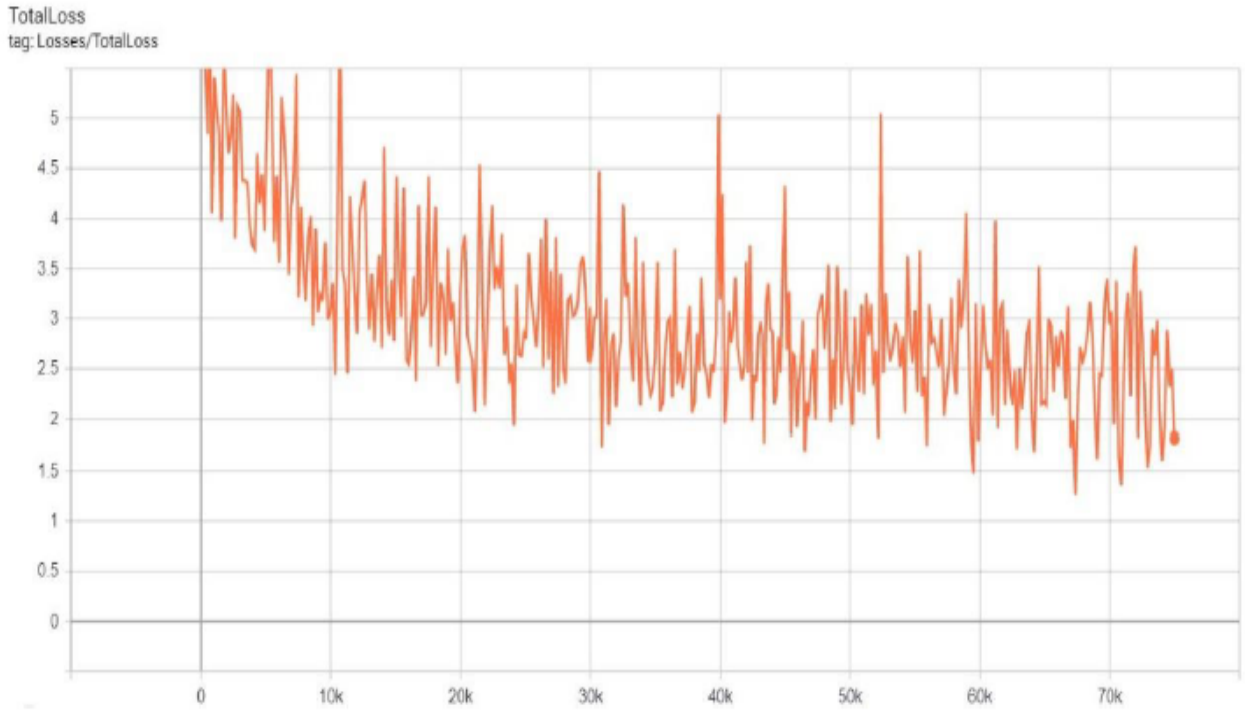

Figure 3.6: Loss vs Step

3.3 Disease Classification Model

The main task in plant disease detection model is to provide a specific class of disease as an output for the given input image. In deep learning, classification models give a probability of different classes, for the given input data. These models use CNN based architecture which takes the image pixels as input and output different class probabilities. Some state of the art classification models are InceptionNet, VGG16, GoogleNet, AlexNet and ResNet.

3.3.1 Introduction of Convolutional Neural Network (CNN)

A Convolutional Neural Network (CNN) is a deep, feed-forward artificial neural network in which the network maintains the hierarchical structure by learning internal character representations and generalizing the attributes in specific picture problems such as object detection and other computer vision problems. CNN is also known as a **"ConvNet"**.

Convolutional Neural Network consists of an input layer, an output layer, and

a hidden layer or more. A Convolutional Neural Network is distinct from a regular neural network since the neurons in its layers are organized in three dimensions, such as the dimensions of width, height and depth. This helps CNN to transform a three-dimensional input volume into an output volume. The hidden layers are a combination of layers of convolution, layers of pooling, layers of normalization, and layers of fully connected. CNN's use several Conv layers to filter volumes of inputs to greater abstraction levels.

CNN's improve their capacity to identify oddly arranged objects by utilizing pooling layers with a minimal invariance in translation and rotation. Pooling often reduces resource use by enabling the usage of more convolutional layers. Standardization layers are used for normalization over local input regions by shifting all inputs in a layer to a mean of zero and one variance.

Other regularization methods, such as batch normalization, where we normalize during the whole batch activations, or dropout, where we neglect randomly chosen neurons throughout the training process, may also be implemented.

Fully- connected layers have neurons that functionally match convolutional layers (compute dot products), but are different in that they are linked to all prior layer activations. All layers of the Convolutional Neural Networks are defined in sections ahead.

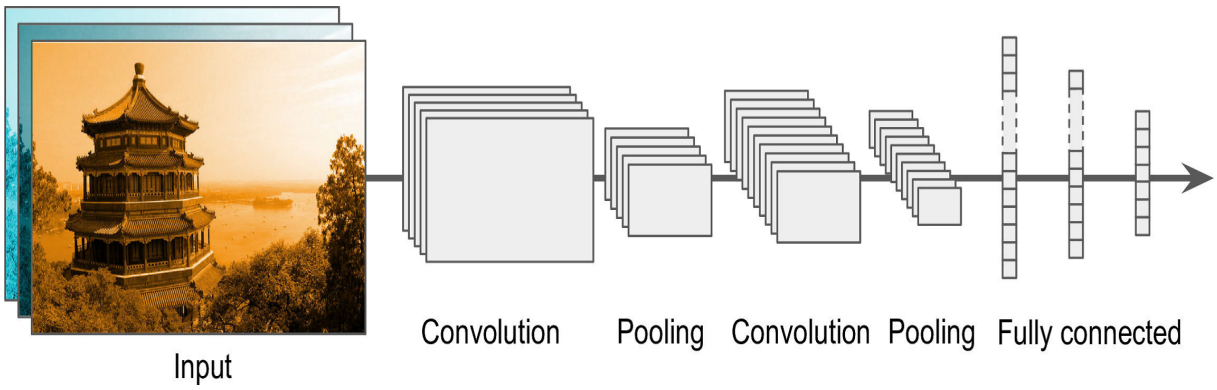

Figure 3.7: Typical CNN architecture

Typical CNN architectures stack several convolutional layers (each usually accompanied by a ReLU layer), then a pooling layer, then a few convolutional layers (+

ReLU), then another pooling layer, etc. The image gets smaller and smaller as it moves across the network, but typically still deeper and deeper. The typical CNN architecture is shown in figure 3.7.

1. Convolution Layer

The convolutional neural network takes an image as the input gives importance to a particular object present in the image input and is capable of identifying the object present in an image. CNNs most significant building block is the convolutional layer. As shown in figure 3.8, the neurons in the first convolutional layer are not connected in the input picture to a single pixel, but rather to pixels in their receptive fields.

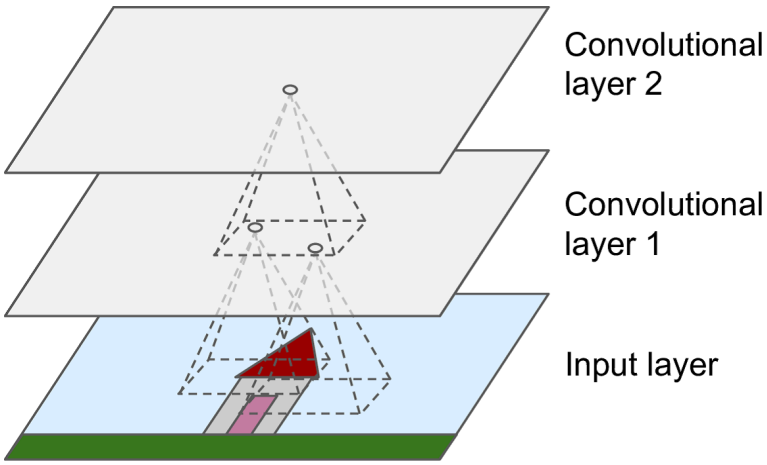

Figure 3.8: Convolutional layer[2]

Besides, each neuron in the second convolutional layer is only linked to the neurons located in the first layer inside a small rectangle. This architecture allows the network to focus in the first hidden layer on smaller low-level features, then combine them into larger higher-level features in the next hidden layer, and so on. In real-world images, this hierarchical design is popular, which is one of the reasons why CNN performs so good for image recognition.

A neuron in row i, column j of a layer, is connected in rows to the neuron outputs of the previous layer located in rows i to i + fh – 1, columns j to j + fw – 1,where fh and fw are the height and width of the receptive field. In order for a layer to have the

same height and width as the previous layer, as seen in the figure 3.9, it is normal to add zeros around the data. This is called zero padding.

A large input layer can also be connected to a much smaller layer by spacing

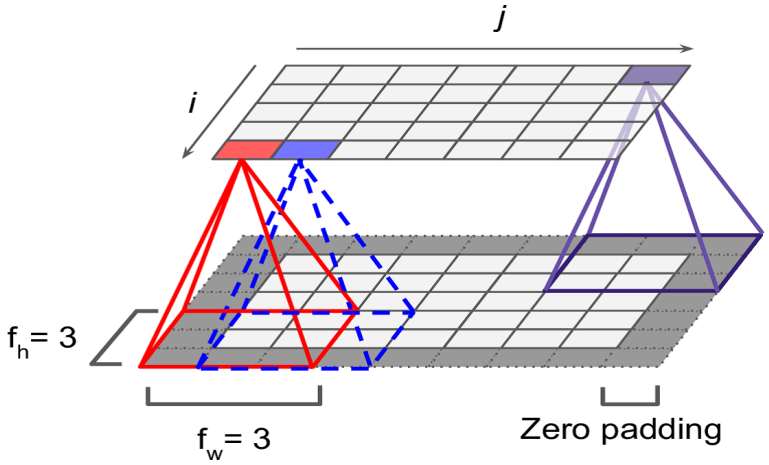

Figure 3.9: Connections between layers and zero padding [2]

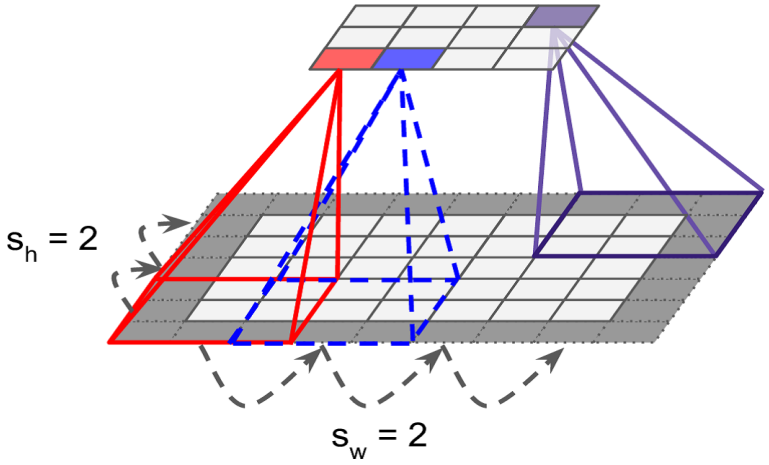

Figure 3.10: Reducing dimensionality using a stride of 2 [2]

out the receptive fields, as shown in figure 3.10 .The shift from one receptive field to the next is called the stride. The diagram links a 5 x 7 input layer (plus zero padding) to a 3 x 4 layer using 3 x 3 receptive fields and a 2 stride.

2. Activation Function

The activation function decides whether a neuron should be activated or not by calculating the weighted sum. The purpose of the activation function is to introduce

non-linearity into the output of a neuron.The activation function shown in figure 3.11. Without an activation function, our neural network would not be able to learn and model other complicated kinds of data such as images, videos, audio, speech, etc.

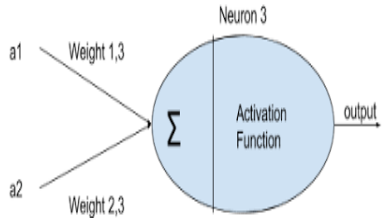

Figure 3.11: Activation Function

Within a neural network, an activation function performs a non-linear transformation on weighted input data. ReLu or rectified linear function is a popular activation function for CNNs that zeros out negative inputs and are defined in the below equation. The rectified linear function speeds up training while not noticeably sacrificing accuracy.The graph of ReLu function is shown in below figure 3.12.

$$\mathbf{f(x) = max(0, x)} \tag{3.1}$$

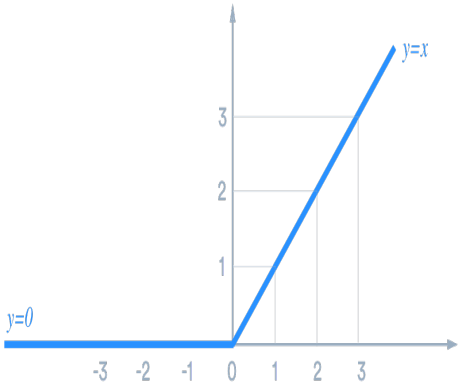

Figure 3.12: ReLu-Rectified linear units

3. Pooling Layer

Pooling is a procedure which reduces the input to a single value (subsampling) over a given area.Pooling is a procedure that reduces inputs over a given area to a single value (subsampling).Their objective is to subsample (i.e., shrink) the input image to

reduce computational load, memory usage and number of parameters (thereby limiting the risk of overfitting).

There are two types of Pooling: Max Pooling and Average Pooling. Max Pooling returns the maximum value from the portion of the image covered by the kernel. On the other hand, Average Pooling returns the average of all the values from the portion of the image covered by the kernel. The Max Pooling process is displayed in figure 3.13.

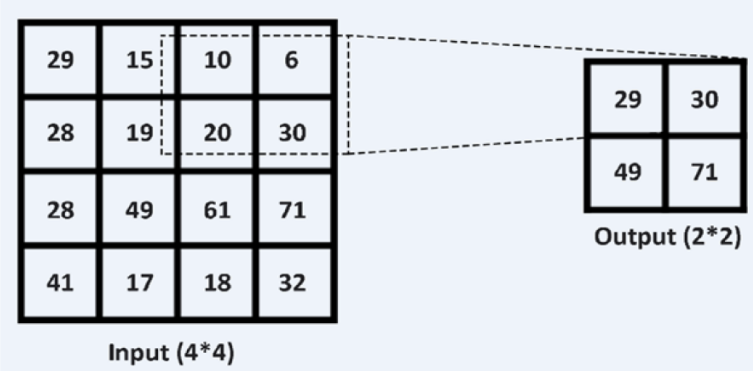

Figure 3.13: Max Pooling-Reducing the number of pixels

Figure 3.14 displays the max pooling layer and is the most common form of layer of pooling. In each receptive layer, only the max input value makes it to the next step, while the other inputs are dropped.

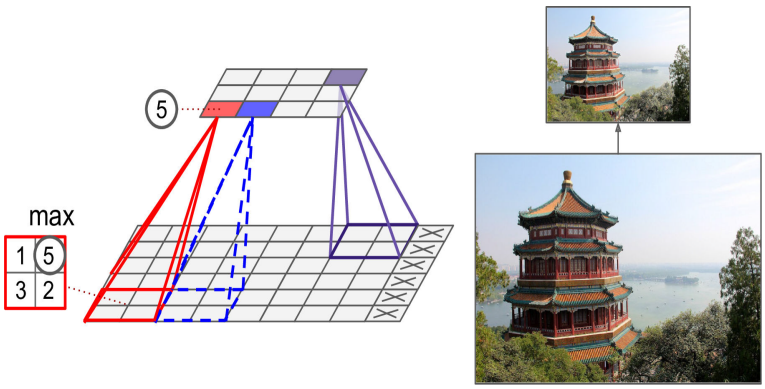

Figure 3.14: Max pooling layer (2 x 2 pooling kernel, stride 2, no padding)

4. Fully Connected Layer

Fully connected layers are an important component of the Convolutional Neural Networks (CNNs), which have proved to be very effective in the detection and classification of computer vision images. The CNN process begins with convolution and pooling, breaking down the picture into attributes, and analysing them separately. The outcome of this process feeds into a fully connected neural network framework which drives the final decision on the classification. The Fully Connected Layer in a CNN is shown in figure 3.15

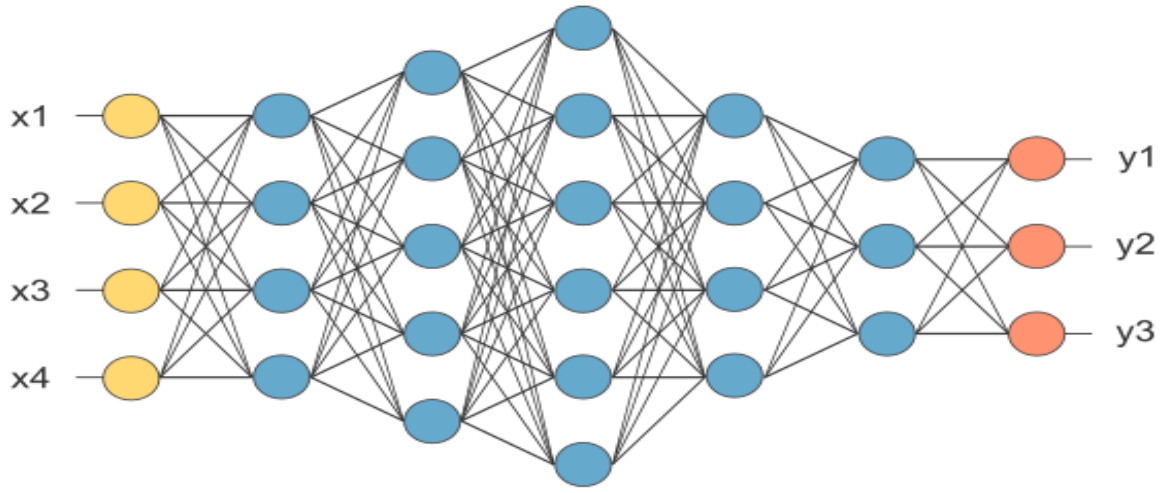

Figure 3.15: Fully connected layer of CNN [3]

A fully connected layer's purpose is to take the effects of the convolution and pooling process and use them to classify the image into a label (in a basic illustration of classification) [33]. Dense (fully connected) layers, performs classification on the features extracted by the convolutional layers and down sampled by the pooling layers. In a dense layer, every node in the layer is connected to every node in the preceding layer.

The convolution and pooling performance is flattened into one single value vector, each representing a probability that a certain function belongs to a label. For instance, whether image is of a dog, features that represent objects like whiskers or fur will have a high probability for the label "dog".

To decide the most reliable weights, the fully connected section of the CNN

network goes through its own back-propagation process. Each neuron is given weights that give priority to the label that is most suitable. Finally, the neurons "vote" on each of the labels, and the winner of that vote is the classification decision.

CNN Training and Inference

Like multi-layer perceptrons and recurrent neural networks, convolutional neural networks can also be trained using gradient-based optimization techniques. Stochastic, batch, or mini-batch gradient descent algorithms can be used to optimize the parameters of the neural network. Once CNN has been trained, it can then be used to accurately predict outcomes for a given input for inference.

Applications of Convolutional Neural Network

The applications of Convolutional Neural Networks include various image and speech processing systems, as well as state-of-the-art AI systems, listed as under:

- Image recognition
- Image classification
- Video labeling
- Text analysis
- Speech recognition
- Natural language processing
- Text classification
- Robots
- Virtual assistants
- Self-driving cars
- Analyzing Documents
- Anomaly Detection

- Drug discovery
- Checkers game
- Health risk assessment

3.3.2 Architecture of proposed CNN model for disease classification

To classify crop diseases effectively from the leaves, we have proposed our own machine learning model that is inspired by the Inception model. Instead of having only one kernel of fixed size, the implementation expands this concept by adding multi-size kernels at each layer and concatenating their outputs. The key concept behind module creation is to factorize the convolutional process into smaller convolutions.

For example, factorizing the 5 x 5 convolutional layer into two 3 x 3 convolutional layers not only saves computation power but provides same effect. Using this concept of the Inception module, we designed new architecture of CNNs for plant disease classification. The architecture of the proposed model is shown in figure 3.16.

In this proposed model, after the inception modules, CNN layers with multiple filters size are used. At the end of the model, a fully connected layer with softmax activation is used to produce probability for 20 classes as the output.

To initialize the weights of network, He normal [34] method is used. For every neuron, Exponential Linear Unit [ELU] [35] is used as an activation function. As a regularization method to make the model less prone to overfitting, dropout [36] with rate 0.5 is used in a fully connected network.

Figure 3.16: Architecture of Proposed CNN model for disease classification

3.3.3 AlexNet

The performance of the proposed CNN model for plant leaf disease identification is compared with a standard CNN model, AlexNet. AlexNet was created by Ilya Sutskever, Alex Krizhevsky, and Geoffrey Hinton. It is somewhat similar to LeNet-5, albeit much larger and deeper, and it was the first to stack convolutional layers directly over each other, rather than stacking a pooling layer on top of each convolutional layer [2]. It consisted of 11x11, 5x5, 3x3, convolutions, max pooling, dropout, data augmentation, activation of the ReLU, momentum SGD [37]. After every convolutional and fully connected layer, it attached ReLU activations. Figure 3.17 depicts the architecture of the AlexNet model.

Layer	Type	Maps	Size	Kernel size	Stride	Padding	Activation
Out	Fully Connected	–	1,000	–	–	–	Softmax
F9	Fully Connected	–	4,096	–	–	–	ReLU
F8	Fully Connected	–	4,096	–	–	–	ReLU
C7	Convolution	256	13 × 13	3 × 3	1	SAME	ReLU
C6	Convolution	384	13 × 13	3 × 3	1	SAME	ReLU
C5	Convolution	384	13 × 13	3 × 3	1	SAME	ReLU
S4	Max Pooling	256	13 × 13	3 × 3	2	VALID	–
C3	Convolution	256	27 × 27	5 × 5	1	SAME	ReLU
S2	Max Pooling	96	27 × 27	3 × 3	2	VALID	–
C1	Convolution	96	55 × 55	11 × 11	4	VALID	ReLU
In	Input	3 (RGB)	227 × 227	–	–	–	–

Figure 3.17: AlexNet Model Architecture [2]

The authors used two regularization strategies to minimize overfitting: first they applied **dropout** with a dropout rate of 50 percent to the outputs of the layers F8 and F9 during testing. Second, by randomly shifting the training images by various offsets, flipping them horizontally and adjusting the lighting conditions, they conducted **data augmentation**.

Dropout

Dropout is one of the most prominent regularization strategies for deep neural networks. It was introduced in 2012 by Geoffrey Hinton [38] and Nitish Srivastava et al. [36] and proved to be extremely successful. Even state-of-the-art neural networks were granted a 1–2 percent improvement in accuracy by simply adding dropout. It is a relatively basic algorithm: each neuron (including the input neurons but still without the output neurons) has a probability p of being momentarily "dropped out" during each training step, implying it would be skipped during this training step, but it will be involved in the next phase. Example of dropout regulation is shown in figure 3.18. The hyperparameter p is called the dropout rate, and it is typically set to 50%. Neurons no longer get dropped after the exercise.

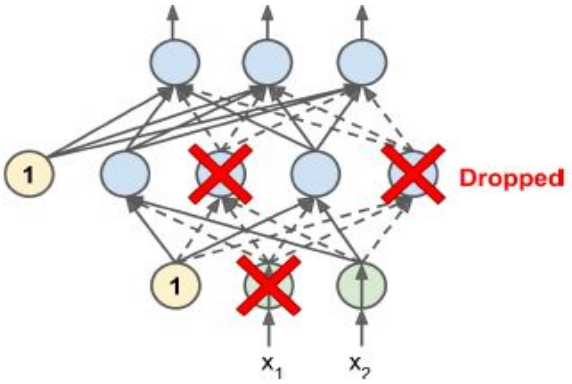

Figure 3.18: Dropout regularization [2]

The power of dropout is to realize that at each training step, a unique neural network is generated. Since each neuron may be either present or absent, a total of 2N networks are possible (where N is the total number of droppable neurons). That is such a large amount that sampling of the same neural network is nearly unlikely twice. If 10,000 training phases are done, there are ultimately 10,000 separate neural networks which are equipped. These neural networks are not autonomous since they share much of their weights but they are still distinct. The combined neural network with all such smaller neural networks can be used as an typical ensemble.

Data augmentation

Data augmentation reduces the size of the training set artificially, creating several realistic variants of every training instance. This eliminates overfitting, rendering it a strategy of regularization. The instances produced should be as reasonable as possible: ideally, provided an image of the enhanced training collection, a human being should not be able to say whether it has been increased or not. In addition, incorporating white noise obviously does not help; the changes will be learnable (normal noise is not) [2]. Figure 3.19 demonstrates the steps by which new training instances are created from current ones.

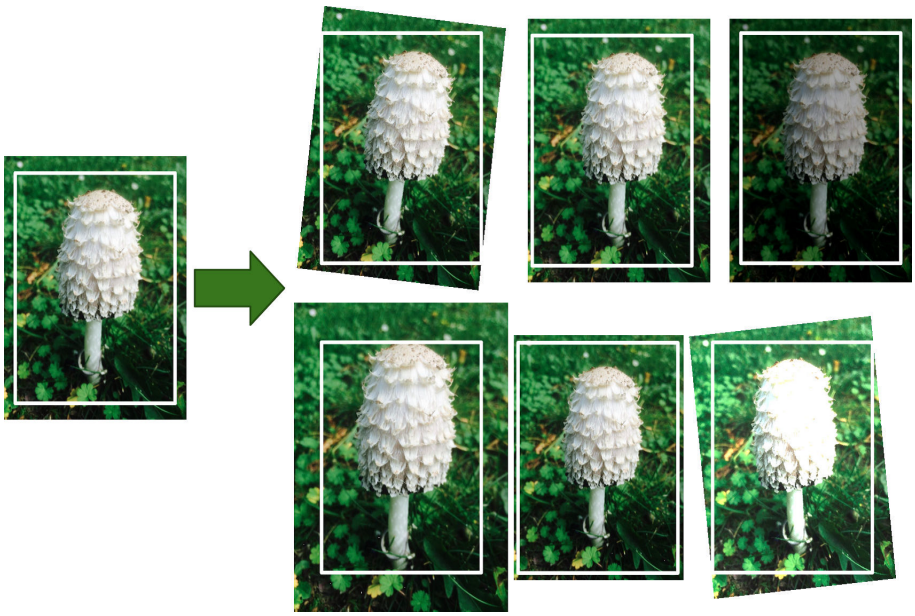

Figure 3.19: Steps of Generating new training instances from existing ones [2]

For instance, any image in the training set may be slightly moved, rotated, and resized by various amounts and the resulting images applied to the training set. This causes the model to be more tolerant of differences in the object's location, orientation, and size in the pictures. Through integrating such changes the scale of the training range will be significantly improved.

3.3.4 Introduction of Inception Model

The basic idea for any CNN based architecture is to increase the number of layers and going more deeper. By increasing the network's depth, a good classification result can be achieved given a well-labeled dataset. But in the case of less amount of datasets or where the human expert is required for labeling dataset, this network prone to overfitting. Apart from this problem, the deeper network requires number of parameters and will require more time for training.

Google's team came up with an innovative idea for CNN architecture in their published paper "Going deeper with convolutions". They were motivated by the internet meme "we need to go deeper" from the Inception movie [4] . Salient features of images can have a large variation in size. Consider an image of a cat, where the cat can be of any size and in part of the image. That's why choosing the right kernel size is very important in architecture. A filter with a large size tends to capture features spread in the whole image while a filter with a small size tends to capture features located in a small region of the image. If a filter with proper size is not selected at a given position of the network, it might miss some features [39].

Instead of using only one filter at a time, Inception paper authors proposed to use multiple filters with different sizes and concatenate the outputs of these filters as shown in figure 3.20. Here filters with sizes 1x1, 3x3, 5x5 and max-pooling layer are combined in one layer that means the output of this layer is a combination of all vectors generated from different filters. These modules are stacked on top of each other in the network. But here one problem is that, as we go deeper the number of channels in the input of the module increases. At this level, 5x5 and even 3x3 convolutional filters are heavy computation filters.

This problem leads the authors of Inception V1 paper to the second idea for the Inception module. Instead of directly taking the output from the previous module as an input, first, these vector's dimensions are decreased to lower the computation. Signals coming from lower layers are first passed through 1x1 convolutional layers with the lower temporal resolution to decrease its temporal dimensions while maintaining the spatial

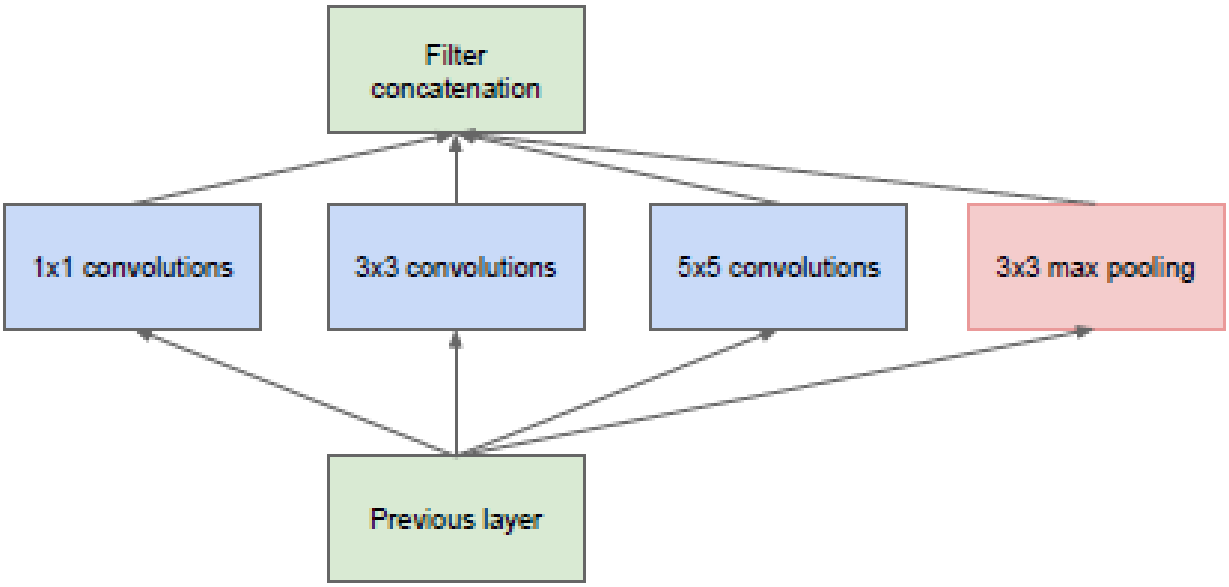

Figure 3.20: Frist Proposed Inception Model [4]

dimensions. The second proposed module is shown in figure 3.21.

In general, Inception consists of these modules stack on top of each other with some max-pooling layers with stride 2 in between. The authors suggested using these modules at a higher level while keeping the lower level of network as it is. Authors used GoogleNet as a base architecture to build the InceptionNet on top of it. To overcome the problem of dead relus, authors used 2 auxiliary classifiers and total loss is the weighted summation of loss of final layer plus these 2 auxiliary classifier's loss.

After the success of the Inception V1, the architecture was continued to be improved in the next paper "Rethinking the Inception Architecture for Computer Vision"[40]. The main idea behind this paper is to factorizing the convolution operation further as it is likely to decrease the number of parameters and decrease the computation complexity. A 5x5 convolution layer compares to the 3x3 convolution layer is computationally expensive 25/9=2.78 times. But the use of a low size convolution layer comes with a loss of information. So instead of using one 5x5 convolution layer, it can be factor-

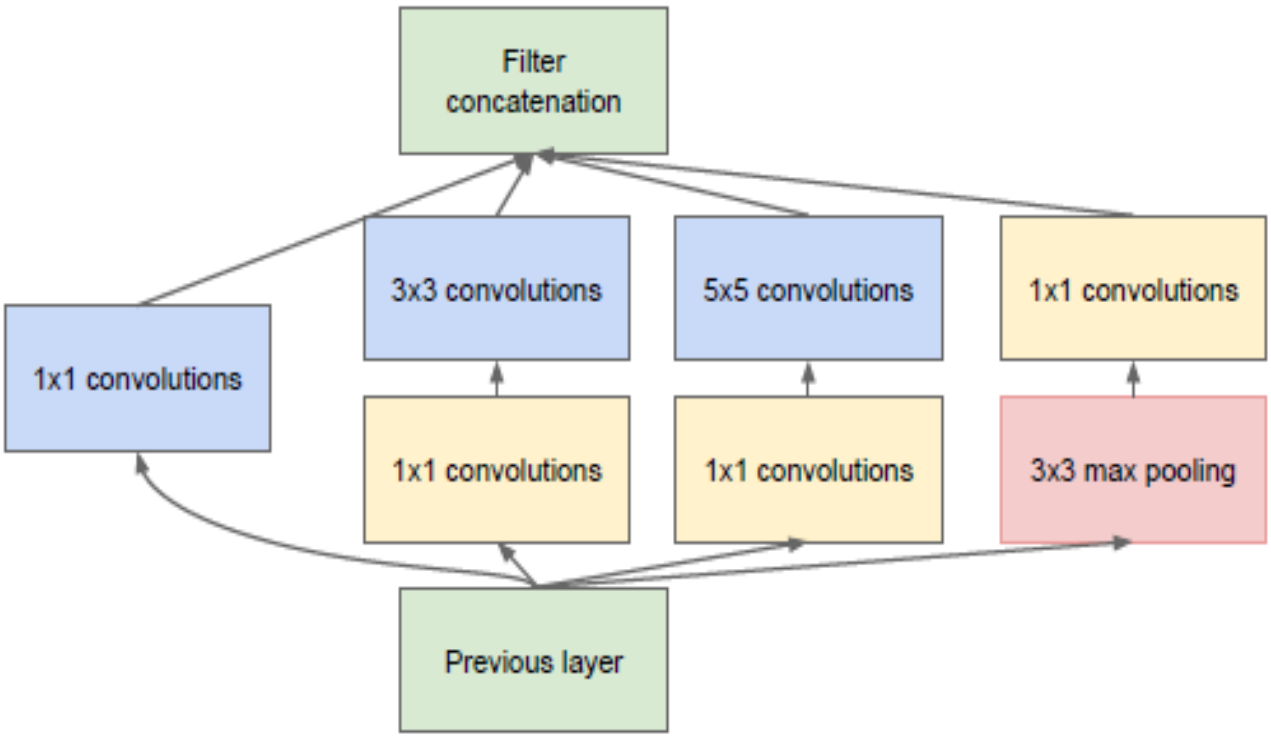

Figure 3.21: Second Proposed Inception Model [4]

ized to two 3x3 convolution layers as shown in figure 3.21. We can think of a convolving 5x5 layer on the input layer as convolving two 3x3 layers on top of each other. In this way, number of parameters are reduced from 5*5=25 to 3*3=9.

The authors proved in paper that mathematically it is not advisable to use convolution filters with size more than 3x3 [39]. They suggested of factorizing any given filter into stacked 3x3 convolution filters. The proposed idea is shown in figure 3.23. The authors also proposed that this 3x3 convolution filter can further be factorized into smaller asymmetric convolution filters like 1x3 and 3x1, in general 1xn and nx1. Using 1x3 and 3x1 filters are the same as convolving the input with a 3x3 filter, but this further reduces the number of parameters used in the network. They suggested replacing any nxn filter with a series of 1xn and nx1 filters as n increases. This is shown in figure 3.23.

The authors trained Inception V2 with stochastic gradient descent using the Tensorflow framework on Nvidia Kepler GPUs. Their best model was trained using RMSporp with momentum 0.9, learning rate 0.045 with an exponential decay factor of 0.94

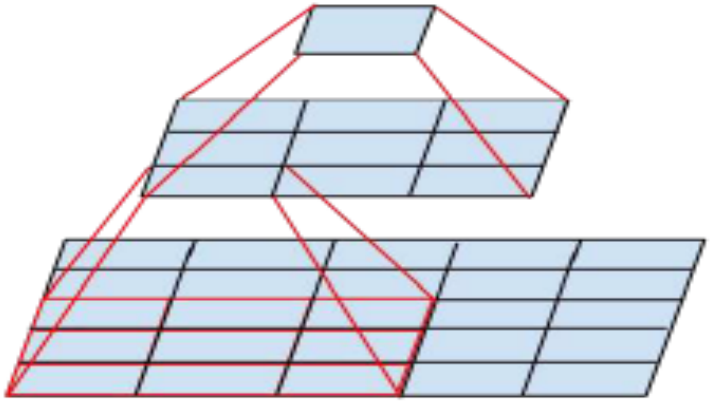

Figure 3.22: Small network replacing the 5 x 5 convolutions [2]

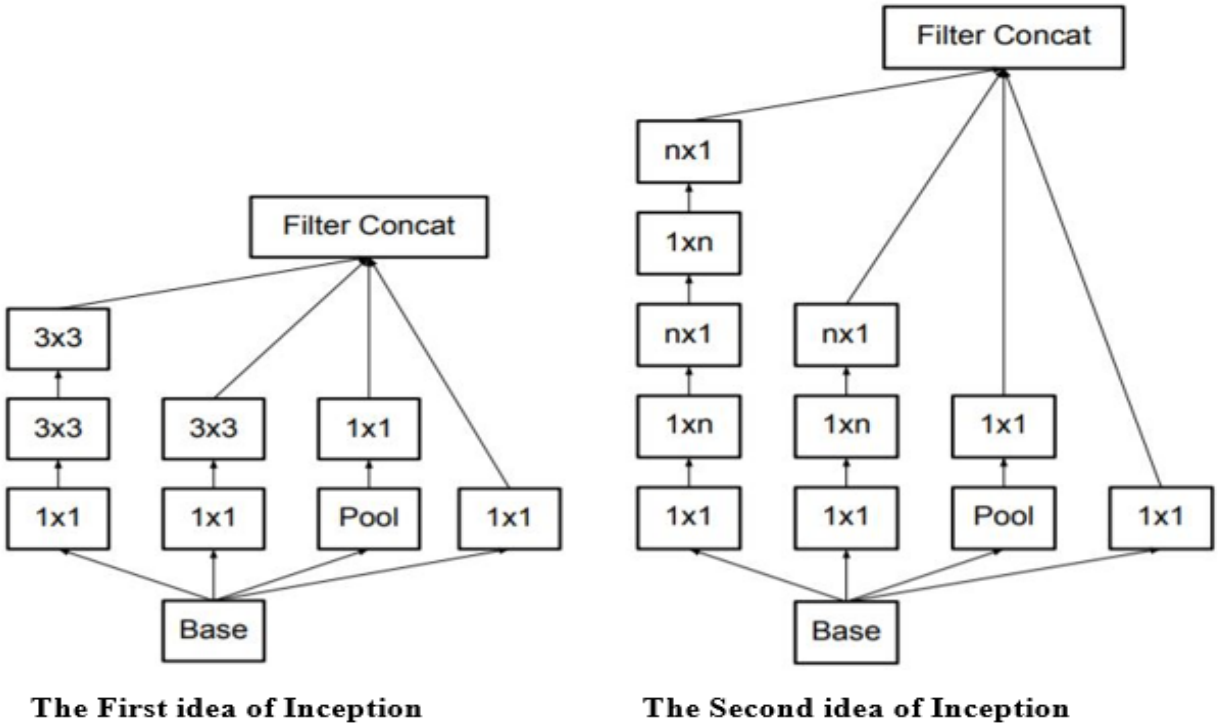

Figure 3.23: Inception modules [4]

after every two epochs. It performed exceptionally well with the top-1 error of 17.2% and top-5 error of 3.58% beating all other architectures of its time.

3.4 Tensorflow 2.0

Tensorflow is one of the commonly used machine learning libraries and other algorithms that require a huge number of mathematical operations. Tensorflow was developed by Google and it's one of the most popular Machine Learning libraries on GitHub [41]. TensorFlow central feature is the computational graph and tensors that move around edges of all the nodes.

3.4.1 Tensor

Mathematically a tensor is an N-dimensional vector, which implies that it is possible to use a tensor to describe N-dimensional information. As the dimensions keep on increasing, data representation becomes more and more complex. For example, if take a Tensor of the form (3x3) then we can simply call it a matrix of 3 rows and columns. If select another Tensor of the form (1000x3x3), can call it as a vector or set of 1000 3x3 matrices. Here we call (1000x3x3) as the shape or dimension of the resulting Tensor. Tensors can either be a constant or a variable [2].

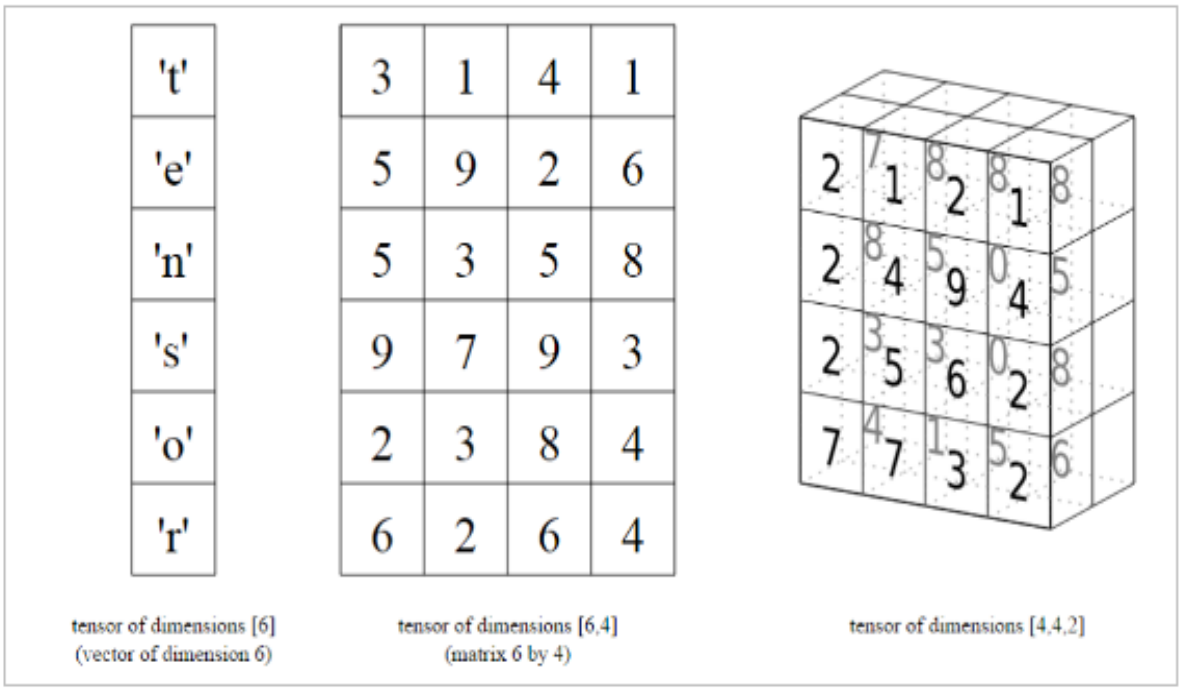

Figure 3.24: Tensors [2]

3.4.2 Keras

Keras is a high-level deep learning framework developed by François Chollet as a part of the research project in 2015. It allows one to very quickly and effectively build various forms of neural networks. It became popular as it was easy to use, flexible and beautiful design. In the new version of TensorFlow tf-2.0.0, the developers have made Keras the default API. So it comes with advantages of both Tensorflow and Keras.

Using Keras we can build our neural networks in three ways: (1) Sequential API (2) Functional API (3) Subclassing API. Sequential API allows us to build simple models in a sequential manner. In functional API, we can develop more complex networks where two or more kinds of layers are being connected. Subclassing API provides the most flexibility among all three APIs. Models are created by defining each layer as a class then all layers are connected. Models containing branches, loop or conditions can easily be developed. Hence, this API is very useful for researchers. Keras also provides a way to save our trained model which can be used later on.

3.4.3 Tf.data

As Keras provides good features to develop our model, we can take advantage of TensorFlow data API. It handles tasks related to creating a dataset, data augmentation, pipelining of data, and flow of data into and out of our model. TensorFlow is responsible for all the specifics of the implementation, such as multithreading, queuing, batching, and prefetching.

TensorFlow's data API can read input data into CSV or binary file format called tf.records. tf.record is a flexible and efficient binary format usually containing protocol buffers (an open-source binary format). Data API can also be used with the SQL database. Reading huge datasets efficiently is not the only difficulty: the data also needs to be pre-processed, usually normalized. Moreover, it is not always composed strictly of convenient numerical fields: there may be text features, categorical features, and so on. These need to be encoded, for example using one-hot encoding, bag-of-words encoding, or embedding. This all can be done using tf.transform and tf.dataset.

From the input CSV files, tensors are created and they grouped together in batch to form a tf.dataset. Then using chaining of transformation, different operations like repeat, shuffling, interleaving on the dataset can be applied. Instead pre-processing is applied of the dataset required for model training. To efficiently prepare the batch for training, a concept called prefetching is used. When one batch is being processed for the training on GPU, the next batch is being prepared by tf.data for the next training step. This way training does not need to be stopped and can be done continuously. Further, we can use multithreading on CPU for preparing datasets. The example is shown in figure 3.25 [2].

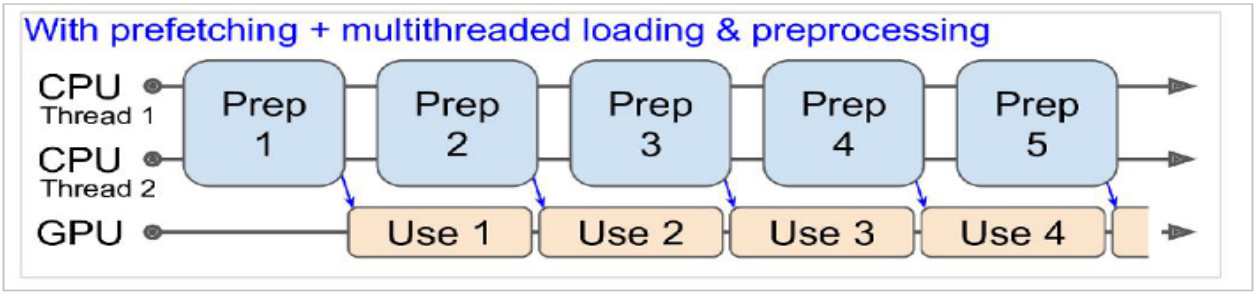

Figure 3.25: Prefetching works in parallel with CPU and GPU [2]

3.5 Training and Testing

3.5.1 Leaves Detection

The training process was done on the system with Intel coreTM i5-8300H CPU @ 2.3 GHz(8 cores) and NVidia R GTX 1050 ti 4 GB VRAM using CUDA R parallel processing library and Tensorflow framework. The dataset for leaves detection was divided into train and test set 90% and 10% respectively. For leaf detection used the pre-trained SSD on the COCO dataset and trained on our dataset for 75000 steps. The hyper parameters used for the training of the proposed leaf detection model are mentioned in Table 3.1.

Rms-prop Optimizer with batch size 8, and learning rate 0.004 with exponential decay with decay factor 0.95 was used to train the leaf detector. The learning

Table 3.1: Hyper parameters of Leaf Detection Model

Hyper Parameters	Value
Learning Rate	0.004
Exponential decay	5000 steps with decay factor 0.95
Optimizer	Rms-prop
Batch Size	8
Number of Steps	75000

rate was initially set to 0.004 and it decays exponentially after every epoch.

3.5.2 Disease Classification

Data Augmentation techniques such as brightness, flipping image left-right, flipping up-down randomly was done on the images during training to make the model less prone to overfitting and also helped in making the model more robust. This proposed model consists of nearly 6M million parameters. The hyper parameters chosen for the training are shown in Table 3.2.

Table 3.2: Hyper parameters of Disease Classification Model

Hyper Parameters	Value
Learning rate	10e-3
Exponential decay	Decay factor 0.95
Optimizer	SGD
Epochs	100
Dropout rate	0.5
Kernel initializer	He Normal

3.6 Combining disease Classification model and Leaf Detection model (Hybrid model)

The classification model for plant disease detection is alone not enough to detect disease in the field as it is having limitations of not being capable of classifying multiple leaves in the image. And on the other side, the leaf detector overcomes the limitation of the classification model, but it only detects the leaves in the image. Hence, to overcome the limitations of both the models, a combination of both the models is required for efficient plant disease detection using machine learning.

For the efficient plant leaf disease detection, firstly we collected image dataset and trained both, the detection model as well as the classification model. Secondly, we detected the leaves on the plant and saved all the detected leaves images. Then we run the proposed classification model feeding those detected leaves images. Finally, the result was correct identification of plant leaf disease through a bounding box which are shown in the figure 3.26.

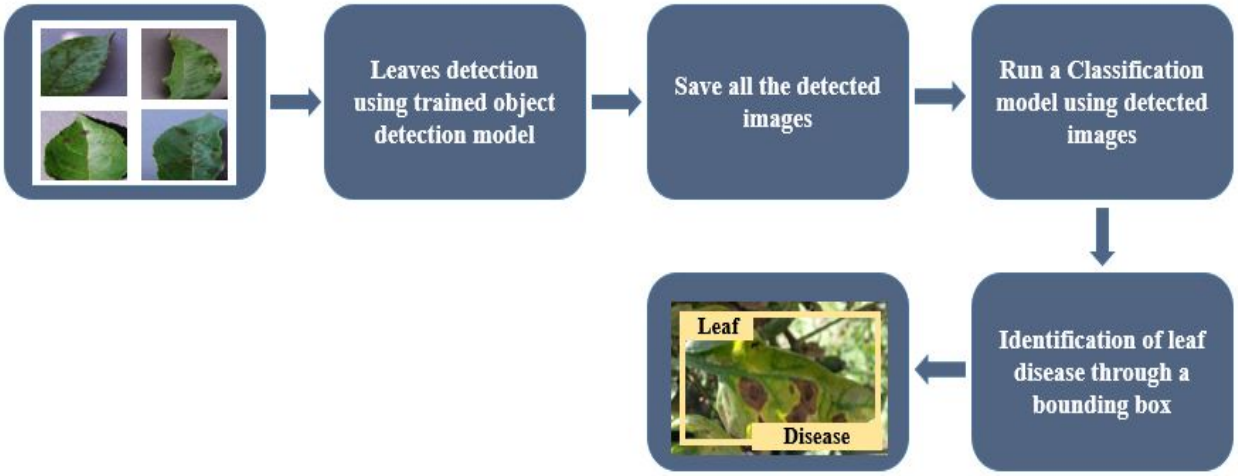

Figure 3.26: Efficient plant disease detection workflow of hybrid model

The bounding boxes which are created around the leaf by the leaf detector get cropped by the hybrid model. These cropped images become the input to the classification model, which then uses this image and predicts the disease classification results. This result, with the help of OpenCV, is printed over the bounding boxes on the detected leaf, thereby labeling the disease present on that leaf.

Chapter 4

Materials and Data

This chapter highlights the leaves identification, disease recognition from the leaf and in-field image dataset and it presents the sample images from all these dataset. In addition to this, the embedded hardware details used in this project are discussed as well as the comparison between these hardware is presented.

4.1 Dataset Description

Training and testing of the model for leaf identification and classification of disease were done on different leaf images from the PlantVillage [42] dataset as well as images collected manually. Throughout the next two subsections, the description of the dataset used to train the leaf identification model as well as the model of classification of diseases is explained.

4.1.1 Leaf Detection Dataset

To precisely train the model for leaf detection, a large number of images of the leaves are needed. Thus, this dataset is created by gathering 501 photos of single and multiple leaves with multiple variations such as shape, size, background, illumination. Figure 4.1 displays reference images of the leaf and annotated leaf images which were used to train the model for leaf detection. As the dataset was created, these images manually annotated individual images for the training of the SSD [29] model. For annotating the leaves, LabelImg [32] tool was used which was generating .XML file, which contains the class and the coordinates of all the ground-truth boxes annotated. Figure 4.1 (f) to

(j) displays sample labeled picture.

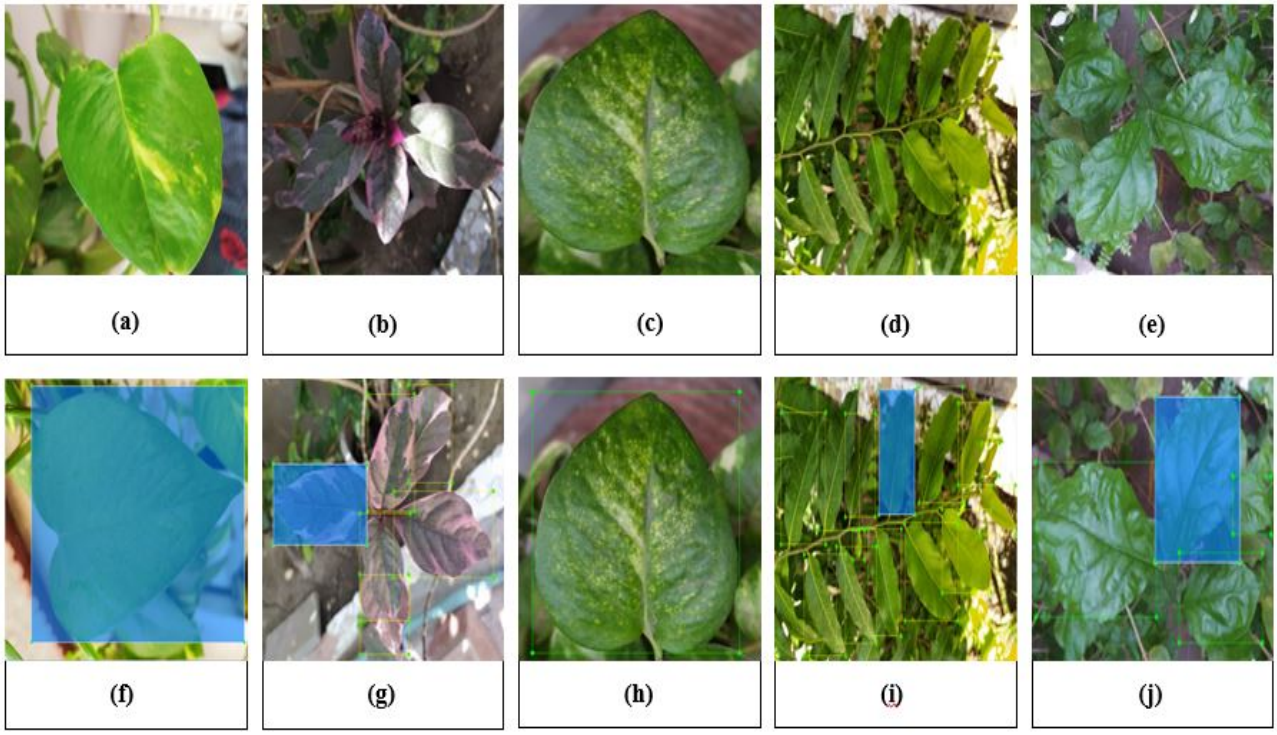

Figure 4.1: Sample leaf images of leaf detection dataset. (a) to (e) Single and multiple leaves captured at different time of the day. (f) to (j) Annotation of individual leaves from the images

4.1.2 Disease Classification Dataset

To build the deep learning model for identifying plant diseases, a rich dataset is required. AN image dataset that includes images of disease-infected plants and healthy plants of different plant species is used to train the disease classification model. A total of 21,948 images are collected from standard dataset, such as the PlantVillage [42] dataset, which is divided into 20 different categories. These 20 classes are representing infected leaves as well as healthy leaves of 4 different crops namely Apple, Corn, Tomato and Potato. Table 4.1 presents the crops considered and the type of disease present in that crop, the class label given to each type and the number of images in each class.

The sample images of these 4 different plants consisting of healthy and 16 different kinds of leaf diseases are shown in figure 4.2. Diseases for Apple: scab, black rot,

Table 4.1: Classes of healthy and infected leaves images dataset

Class label	Class name	Number of images
0	Apple healthy	1645
1	Apple scab	630
2	Apple black rot	621
3	Apple cedar rust	275
4	Corn healthy	1162
5	Corn common rust	1162
6	Corn northern leaf blight	985
7	Corn cercospora gray leaf spot	513
8	Potato healthy	152
9	Potato early blight	1000
10	Potato late blight	1000
11	Tomato healthy	1591
12	Tomato bacterial spot	2127
13	Tomato early blight	1000
14	Tomato late blight	1909
15	Tomato leaf mold	952
16	Tomato Septoria leaf spot	1771
17	Tomato spider mites	1676
18	Tomato target spot	1404
19	Tomato mosaic virus	373
Total number of images		**21,948**

cedar rust. Diseases for Corn: common rust, northern leaf blight, cercospora gray leaf spot. Diseases for Potato: early blight, late blight. Diseases for Tomato: bacterial spot, early blight, leaf mold, late blight, septoria leaf spot, spider mites, target spot, mosaic virus. These are the prime diseases investigated in this project work. The dataset for classification was divided into Train, Validation and Test set in a ratio of 80%, 10%, and 10% respectively.

All these images download from the PlantVillage dataset and then are resized for further process using the developed Python script.

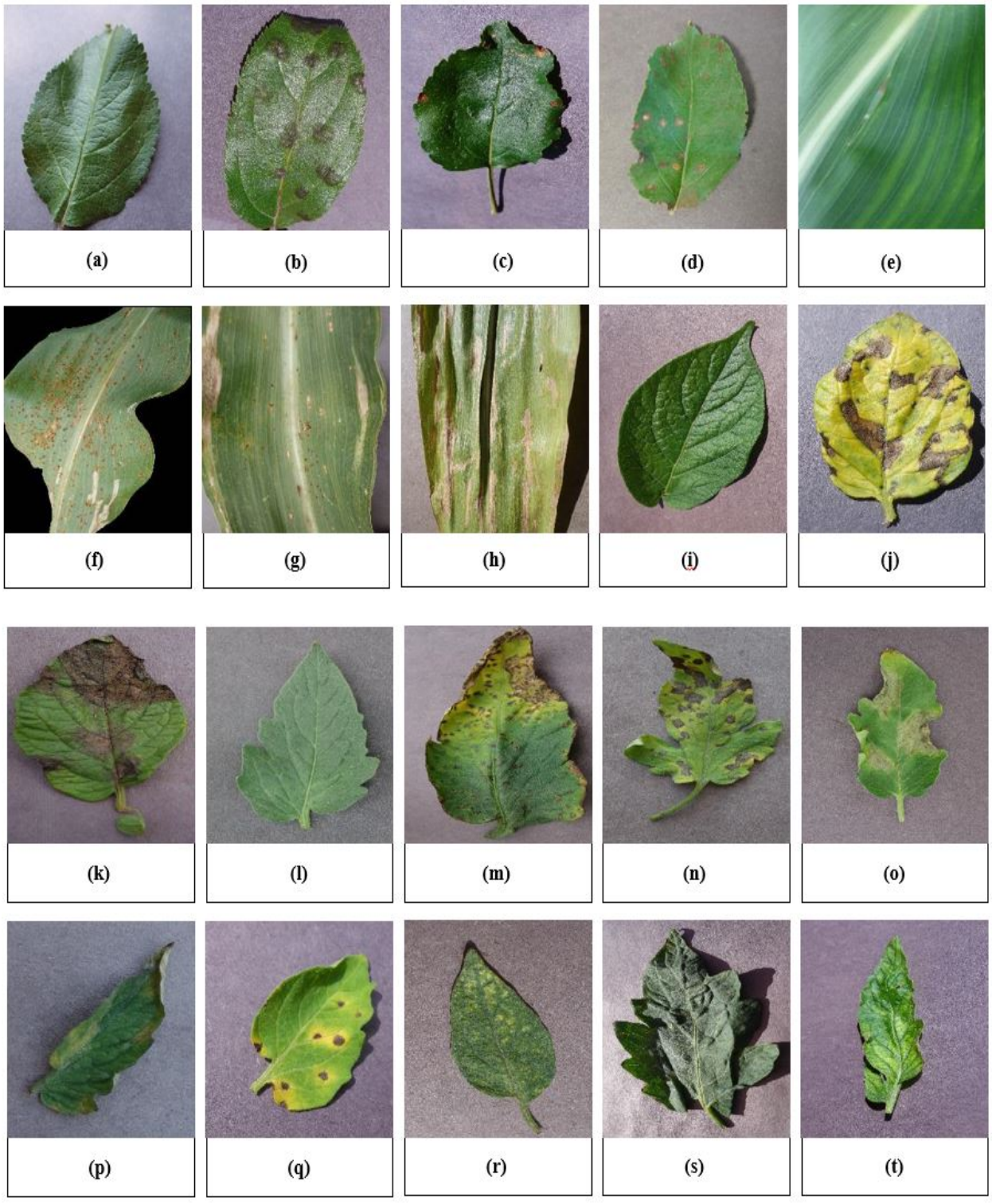

Figure 4.2: Sample images of healthy and infected plant leaves from the PlantVillage Dataset (a) Apple healthy (b) Apple scab (c) Apple black rot (d) Apple cedar rust (e) Corn healthy (f) Corn common rust (g) Corn northern leaf blight (h) Corn cercospora gray leaf spot (i) Potato healthy (j) Potato early blight (k) Potato late blight (l) Tomato healthy (m) Tomato bacterial spot (n) Tomato early blight (o) Tomato leaf mold (p) Tomato late blight (q)Tomato Septoria leaf spot (r) Tomato spider mites (s) Tomato target spot (t) Tomato mosaic virus.

4.1.3 In-field dataset

The proposed disease detection system also tested on a farm to verify the robustness and precision of the proposed architecture in identifying the leaf disease. So, a tomato plant farm was visited and pictures of healthy and infected plant leaves was collected. Some same images of the collected leaves are displayed in figure4.3. Frames from the captured crop field's real-time video is fed to the trained models for detection and classification which are ported on embedded hardware.

Figure 4.3: In-field dataset

4.2 Hardware Description

The Edge AI principle consists of performing computations locally in real-time on an embedded device. Although the training phase needs much more computing resources compared to the inference process, it is not carried out on the embedded device, but a specialized, high computer system or computations through cloud services. Then, for execution, the model with the obtained weights is deployed on the target embedded hardware.

The computer machine used in this project for training was fitted with an NVIDIA GeForce GTX 1050 GPU with DDR4 SDRAM CUDA 10 and 64 GB. This GPU model features 544 Tensor Cores, an NVIDIA technology specially developed to improve the efficiency of matrix multiplication, thereby allowing the training of deep

learning models to be accelerated. This GPU's processing capacity helps it to achieve peak efficiency.

In this project, two embedded platforms used as an inference engine for the real-time plant disease detection. The hardware platforms were considered for the model, as seen in Figures 4.4 and 4.5.

4.2.1 NVIDIA Jetson TX1

The NVIDIA Jetson TX1 System-on-Module integrates the NVIDIA Maxwell GPU design with an ARM Cortex -A57 MP Core (Quad-Core) CPU cluster to provide the output and power efficiency needed by GPU processing, computer graphics, and artificial intelligence technologies of the next generation. Jetson TX1 has 256 core Maxwell design architecture GPU sharing memory with 4 GB LPDDR4 (Low Power DDR4) ROM. Jetson TX1 has an internal 16 GB eMMCflash capacity. NVIDIA Jetson TX1 Board seen in figure 4.4. Nvidia's application operates onboard Ubuntu.

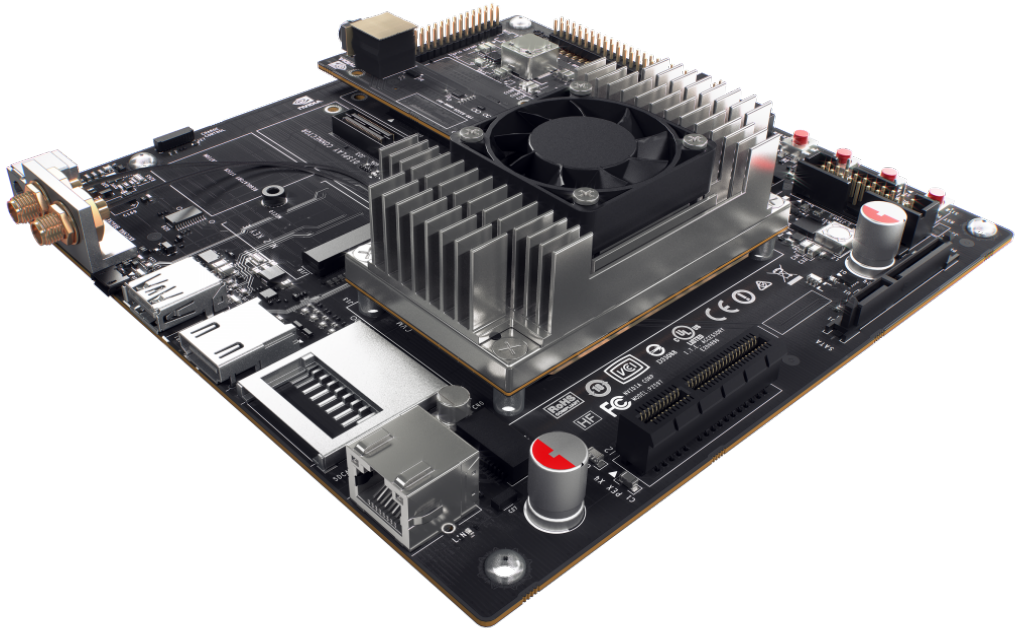

Figure 4.4: Nvidia Jetson TX1 board

4.2.2 NVIDIA Jetson Nano

NVidia's Jetson Nano was introduced in June 2019 specifically for target applications where it is necessary to reduce the board size, power consumption, and price. It includes an NVIDIA Maxwell GPU with a peak performance of 472 GFLOPs for the hardware acceleration. The Nano board does not include any particular deep learning accelerator and can work in two 5W or 10W power modes [43]. This Jetson Nano Board seen in figure 4.5 and the key specs of this board are listed in Table 4.2.

Figure 4.5: NVIDIA Jetson Nano board

4.2.3 Intel Up AI Vision Camera

A camera module is required to fed real time videos and images to the embedded hardware for the identification of plant disease in real-time. So, the high-resolution Intel UP AI Vision camera module is used in this job, which is attached via USB port to embedded platforms. This USB camera has a fixed 1920p x 1080p resolution at 30 frames per second [44], as is seen in Figure 4.6.

Figure 4.6: Intel Up AI Vision Camera module

4.2.4 Comparison of embedded hardware

The comparison between Nvidia Jetson TX1 and Nvidia Jetson Nano used for deploying the model is shown in the Table 4.2 below:

Table 4.2: Main Specification Comparison Of the Embedded Hardware Platforms

	Hardware	
	NVIDIA Jetson Tx1	**NVIDIA Jetson Nano**
Memory	4 GB LPDDR4	4 GB 64-bit LPDDR4
Storage	16 GB eMMC 5.1	MicroSD
CPU (ARM)	4-core ARM Cortex A57	4-core ARM A57
USB	1x USB 3.0 and 1x USB 2.0	4x USB 3.0, USB 2.0 Micro-B
HW Accelerator	256-Cores NVIDIA Maxwell GPU	128-core NVIDIA Maxwell GPU
Operating System	Ubuntu 16.04.2 LTS(64-bit)	Ubuntu 16.04.2 LTS(64-bit)
Nominal Power	10 W	5/10 W
Price	39,675.00	8,799.00

Chapter 5

Experimental Results and Discussion

This chapter discusses the experimental results for the detection of leaves, classification of diseases and presents the results of real-time disease identification on-field. Also the classification results are compared and presented.

5.1 Leaves Detection Results

The SSD model used for leaf detection gives the most accurate results for detecting objects. The leaf detector model in this project detects the leaves and produces a bounding box around the leaves. The detection accuracy is then displayed over the leaf in the input image or video as shown in Figure 5.1. Using the SSD detection model, the leaf detection accuracy achieved is around 98.00 - 99.00 %. The mean average precision (mAP) of 0.4062 is obtained at 0.5 IOU. Table 5.1 presents the result for this leaf detection model. In addition, loss vs epochs curve for detection model is shown in figure 5.2.

Table 5.1: Results of Leaf Detection

Test Parameter	Value
Mean Average Precision (mAP@0.5 IOU)	0.4062
Classification loss	5.53
Localization loss	1.082

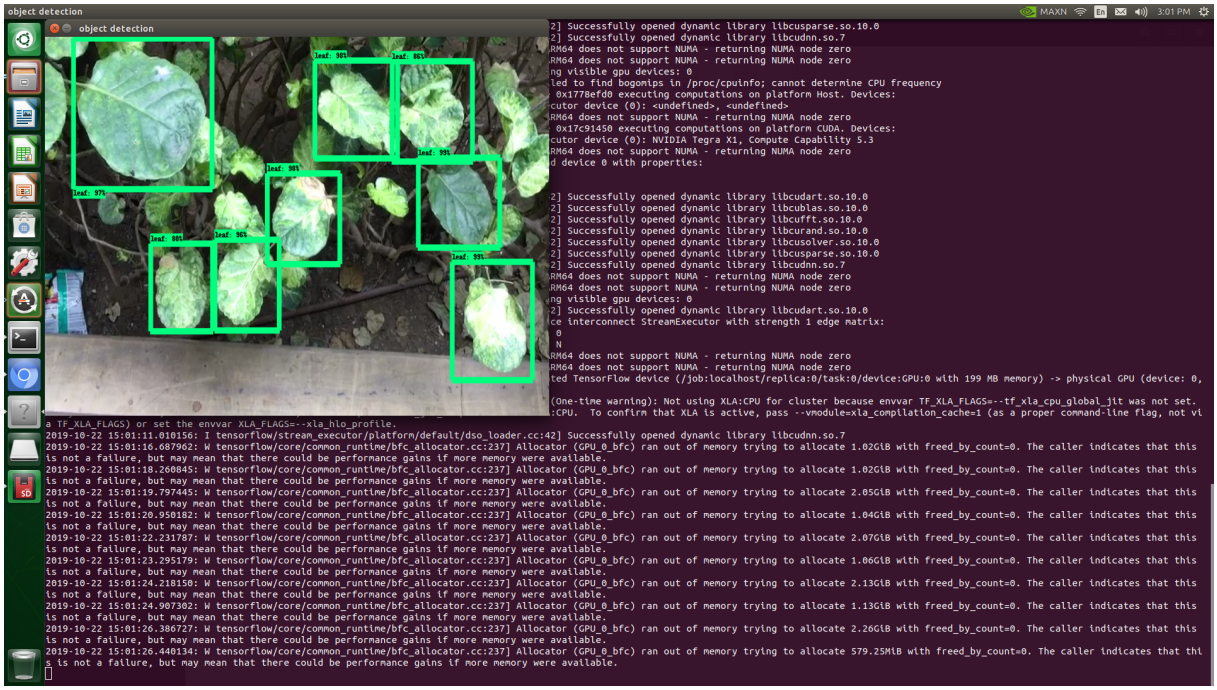

Figure 5.1: Leaves detection simulation result

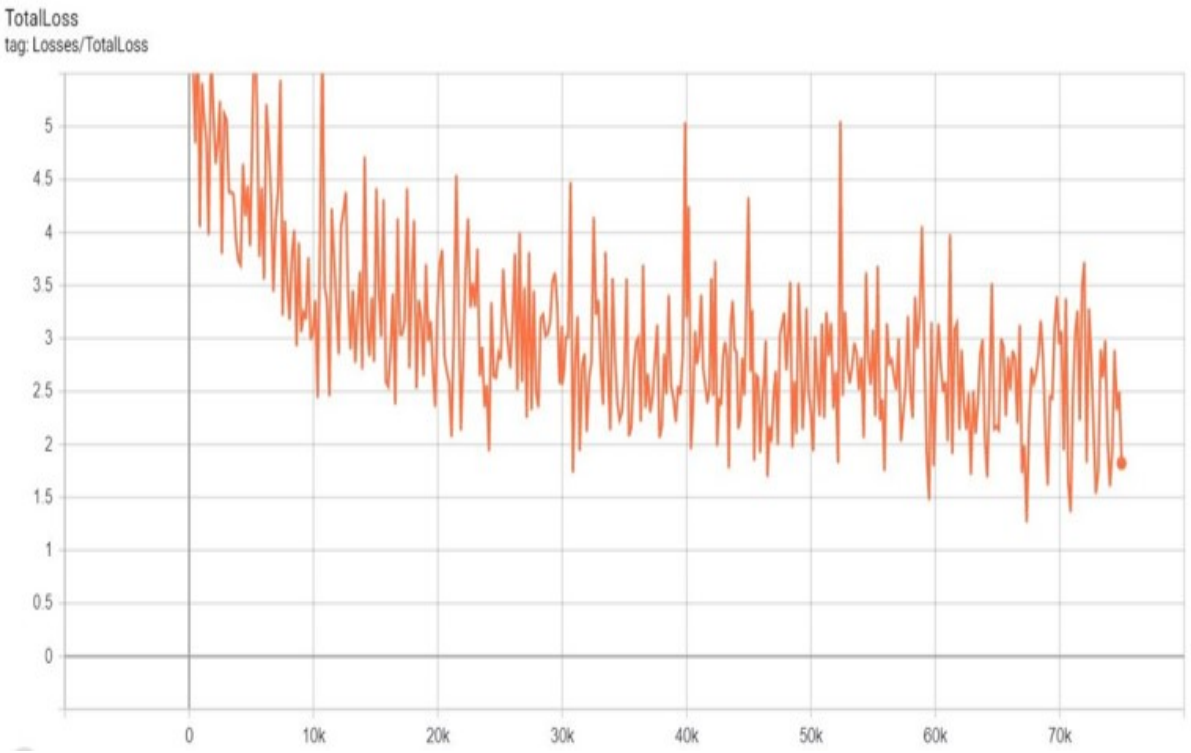

Figure 5.2: Loss vs Epochs curve for leaf detection model

5.2 Disease Classification Results

The categorical accuracy obtained after the training of the proposed classification model for 100 epochs is 96.88 percent. The performance of the proposed plant leaf disease classification model was measured based on precision, recall and F1 score. The expressions [45] for calculating the model's precision, recall and F1 score are shown in equations (5.1), (5.2) and (5.3) respectively. Table 5.2 comprises of values for the model testing parameters determined using the equations described below. Moreover, also display the result curves of the Loss vs Epoch as well as accuracy vs Epoch for proposed classification model in figures 5.5 and 5.6.

$$Precision = \frac{\text{True Positive}}{\text{True Positive} + \text{False positive}} \tag{5.1}$$

$$recall = \frac{True\ Positive}{True\ Positive + False\ Negative} \tag{5.2}$$

$$F1\ Score = 2 \cdot \frac{precision \cdot recall}{precision + recall} \tag{5.3}$$

Table 5.2: Disease Classification Model Results

Test Parameter	Value
Average Precision	93.78
Average Recall	94.91
F1 Score	95.34

Table 5.3 provides a comparison of the proposed model for classifying the disease with the performance of AlexNet. AlexNet has roughly 62 M parameters which require more time to train and it gives a classification accuracy of 95.53 percent. Although our proposed model has comparatively fewer parameters, it takes about 6 M parameters to train and predict the class with 96.88 percent accuracy. The simulation results of proposed disease identification model are shown in Figures 5.3 and 5.4.

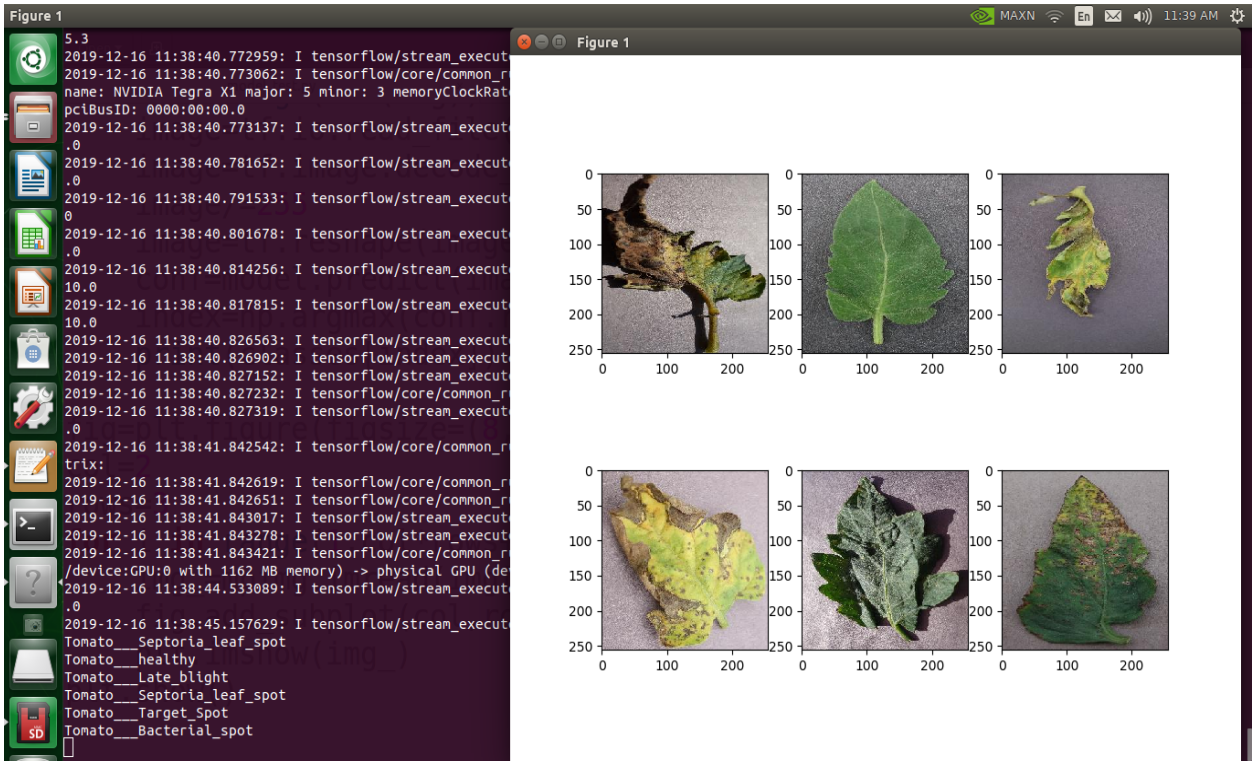

Figure 5.3: Tomato plant leaf disease identification simulation result

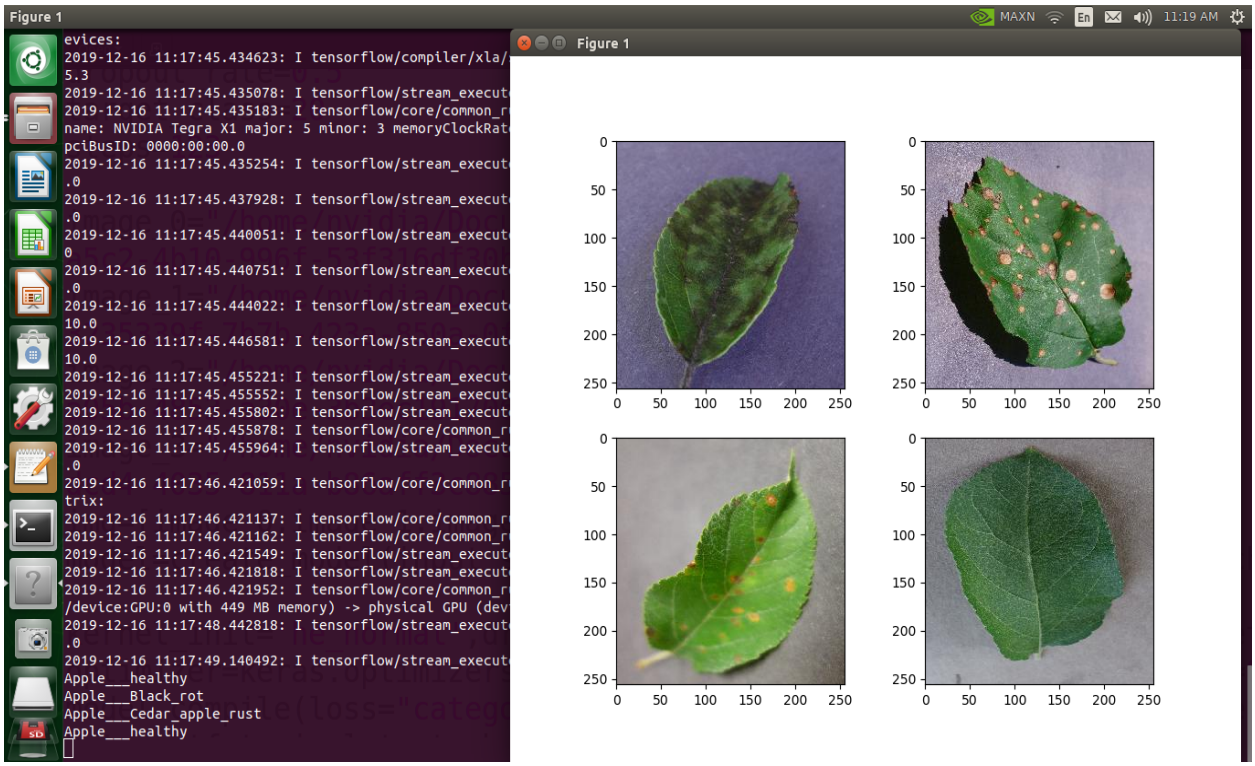

Figure 5.4: Apple plant leaf disease identification simulation result

Table 5.3: Disease Classification results Comparison Table

Model	No. of Parameters	Epoch	Accuracy
AlexNet	62,378,344	150	95.53 %
Proposed Model	6,076,980	100	96.88 %

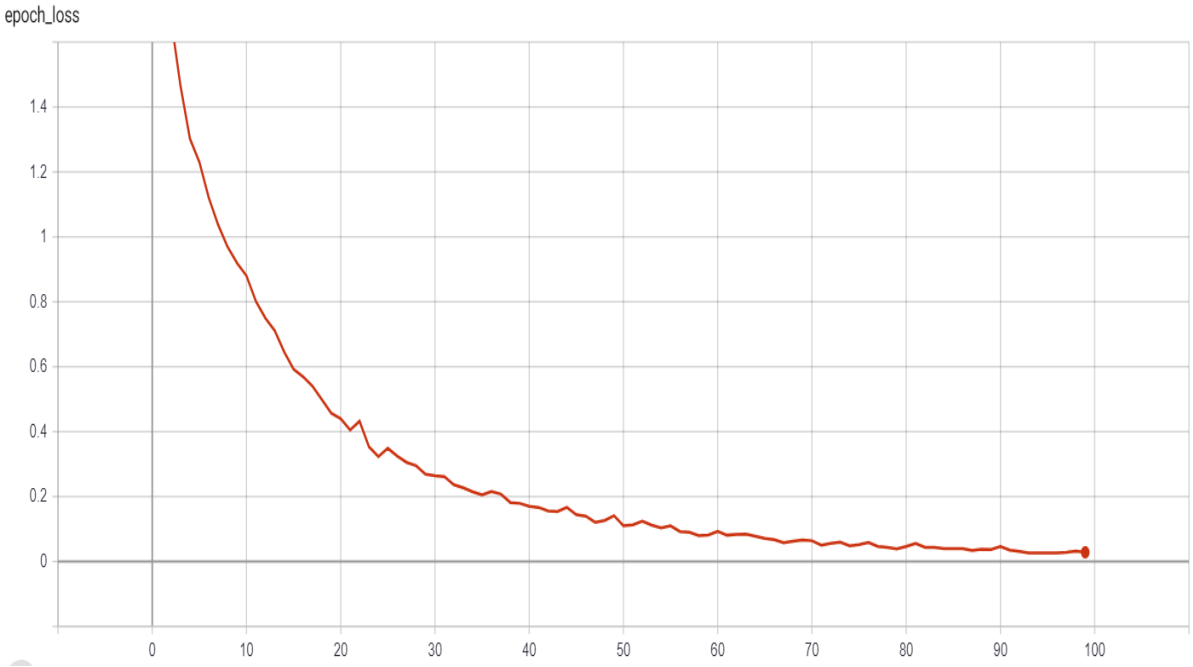

Figure 5.5: Loss vs Epoch curve for proposed classification model

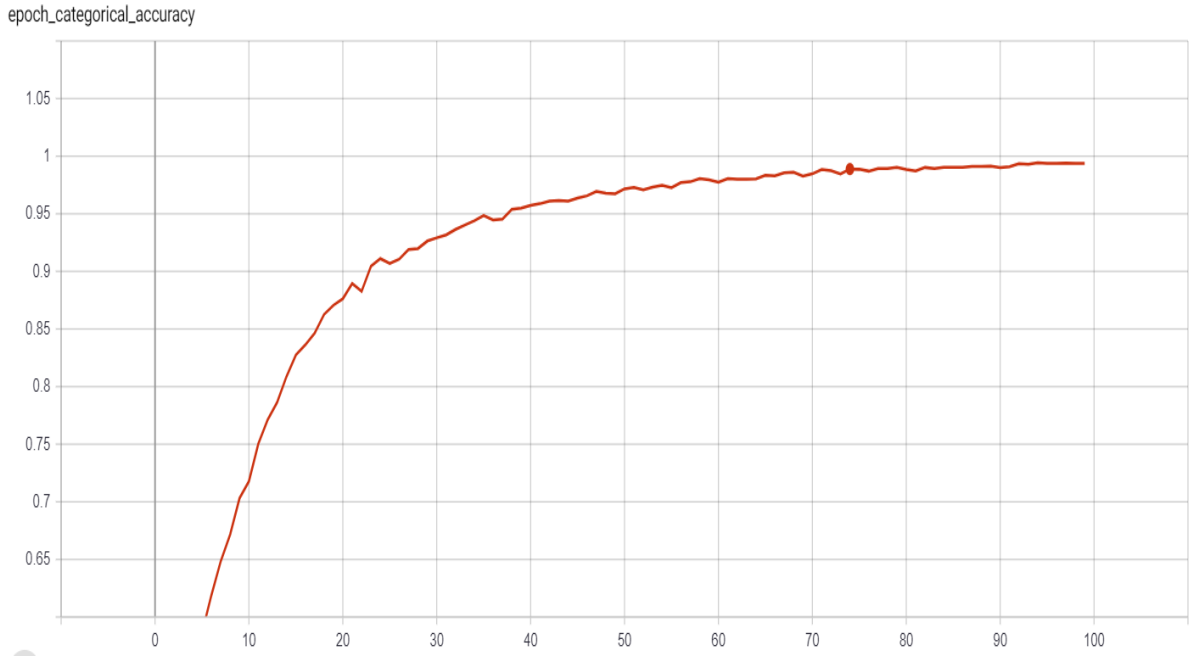

Figure 5.6: Accuracy vs Epoch curve for proposed classification model

5.2.1 Confusion Matrix

When faced with several groups with identical forms, classifiers may be not give accurate result. Infected plant leaf images can often result in high complexity of the patterns shown in the same class at different stages or against different backgrounds, resulting in lower results. The classification accuracy of a model can be visually estimated with a confusion matrix.

A confusion matrix of the outcomes of the diagnosis of diseases is shown in Figure 5.7. This confusion matrix explains the efficiency of our proposed classification model on the test data set for all 20 groups from the PlantVillage dataset, of the four distinct plant diseases as mentioned in Table 4.1

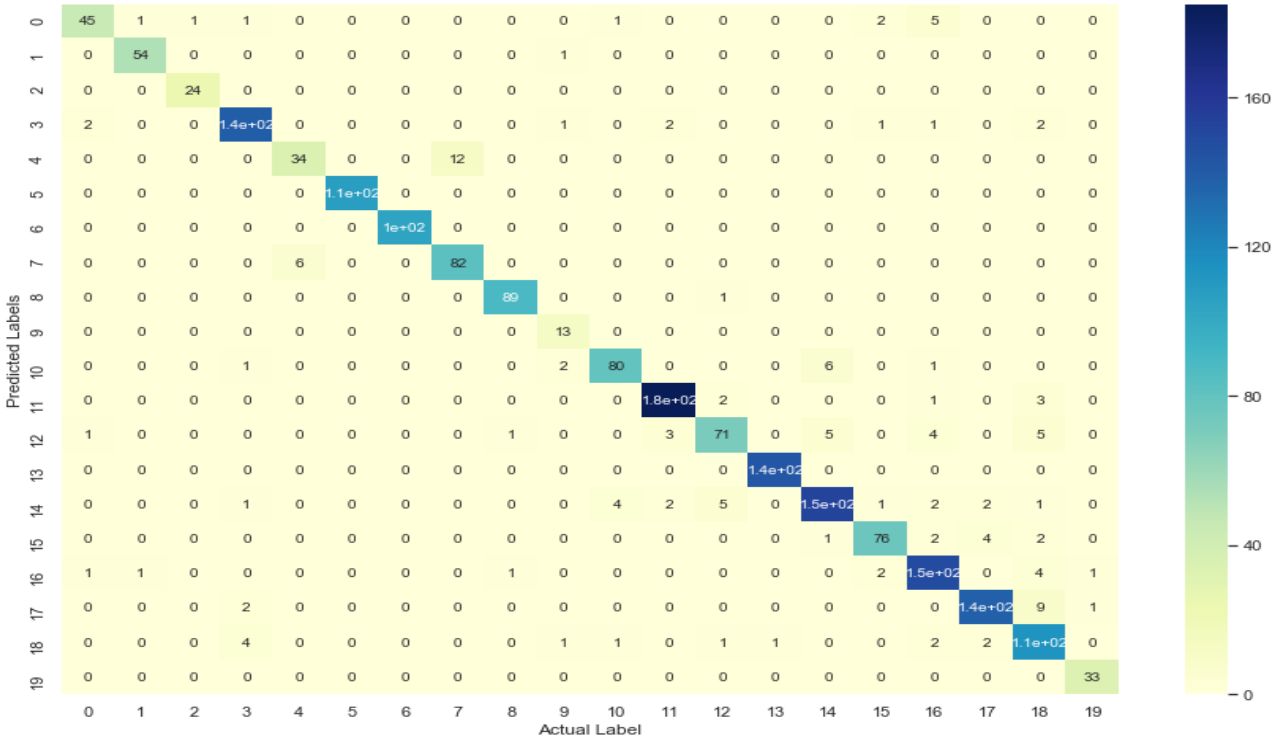

Figure 5.7: Confusion Matrix of proposed disease classification model results

5.3 Real-time In field testing Results

For real-time In-field testing, we visited tomato farm with hardware setups. The NVidia Jetson TX1 and Jetson Nano were used to do actual field testing of the proposed disease identification system. The proposed hybrid model was deployed on both these hardware. The effectiveness of the model for leaf detection and disease recognition was checked by taking the system to a tomato area that was infected with diseases.

Figure 5.8 displays the results of the field test in real-time. The leaves infected with Leaf Mold and Early Blight are correctly identified by the proposed system. It is evident in from these real-time field testing that the proposed model provides effective results in all conditions such as atmosphere, background, soil, and illumination.

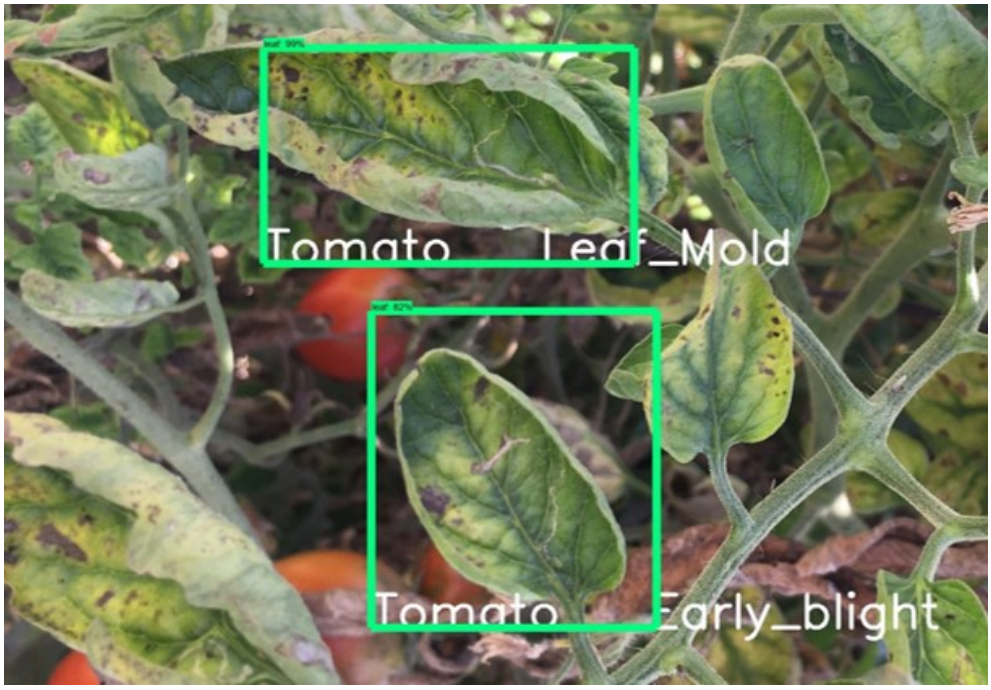

Figure 5.8: Real-time in-field testing result of the proposed system

Chapter 6

Conclusion and Future Scope

Finally, the work performed during this project is summarized in this chapter and the information obtained is discussed to draw some important conclusions. Additionally, specific topics needing to be explored in the future are being mentioned.

6.1 Conclusion

Detection of diseases in crops is of utmost importance to minimize the loss of crops and getting higher yield by providing suitable pesticides. A system for detecting plant diseases in real-time was built and evaluated on several edge AI (embedded) devices in this project work. The convolutional neural networks (CNNs) based architecture has been proposed and adopted to increase its accuracy to identify plant leaf diseases. It has been trained with a custom dataset and tested with in-field farm images. The proposed model obtained higher accuracy as compared to AlexNet for the identification of the plant diseases. The accuracy for the proposed model was 96.88% while with AlexNet it was 95.53% on a similar dataset

Moreover, experimental evaluations have been carried out to highlight performances achieved in terms of power consumption by different embedded solutions selected. Finally, the proposed real-time system ensures the capability of the model to detect plant leaf diseases in single or multiple leaves, under any conditions of the real field.

6.2 Future Scope

In the future, the proposed model could be used for different dataset of several other types of plants for detection of disease which may not be detected easily. The proposed solution can be deployed on the field using an embedded system which can also include IoT enabled devices.

However, using IoT enabled devices the scope of the project can be increased by adding user interface which can be operated remotely through the cloud and also the real-time data can be stored on the cloud for future training and testing.

6.3 Challenges Faced

The most challenging part of the project was collecting the relevant dataset for the diseased plant leaves for training and testing. The filtering of the data was also an interesting part of the project work. After completing the simulation of the proposed machine learning model, it was challenging to deploy it on an embedded system for on-field detection on a farm.

Bibliography

[1] L. Weng, "Object detection part 4: Fast detection models," (Dec 27, 2018) (accessed on Sep 19, 2019). Website Link.

[2] A. Géron, *Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow: Concepts, Tools, and Techniques to Build Intelligent Systems*. O'Reilly Media, 2019.

[3] "Understanding of convolutional neural network (cnn) — deep learning," (2018) (accessed on Sep 9, 2019). Website Link.

[4] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going deeper with convolutions," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 1–9, 2015.

[5] Y. Fang and R. P. Ramasamy, "Current and prospective methods for plant disease detection," *Biosensors*, vol. 5, no. 3, pp. 537–561, 2015.

[6] V. Singh and A. K. Misra, "Detection of plant leaf diseases using image segmentation and soft computing techniques," *Information processing in Agriculture*, vol. 4, no. 1, pp. 41–49, 2017.

[7] G. Geetharamani and A. Pandian, "Identification of plant leaf diseases using a nine-layer deep convolutional neural network," *Computers & Electrical Engineering*, vol. 76, pp. 323–338, 2019.

[8] P. Jiang, Y. Chen, B. Liu, D. He, and C. Liang, "Real-time detection of apple leaf diseases using deep learning approach based on improved convolutional neural networks," *IEEE Access*, vol. 7, pp. 59069–59080, 2019.

[9] X. Zhang, Y. Qiao, F. Meng, C. Fan, and M. Zhang, "Identification of maize leaf diseases using improved deep convolutional neural networks," *IEEE Access*, vol. 6, pp. 30370–30377, 2018.

[10] K. P. Ferentinos, "Deep learning models for plant disease detection and diagnosis," *Computers and Electronics in Agriculture*, vol. 145, pp. 311–318, 2018.

[11] J. Amara, B. Bouaziz, A. Algergawy, *et al.*, "A deep learning-based approach for banana leaf diseases classification.," in *BTW (Workshops)*, pp. 79–88, 2017.

[12] Y. Lu, S. Yi, N. Zeng, Y. Liu, and Y. Zhang, "Identification of rice diseases using deep convolutional neural networks," *Neurocomputing*, vol. 267, pp. 378–384, 2017.

[13] A. Fuentes, S. Yoon, S. Kim, and D. Park, "A robust deep-learning-based detector for real-time tomato plant diseases and pests recognition," *Sensors*, vol. 17, no. 9, p. 2022, 2017.

[14] C.-L. Chung, K.-J. Huang, S.-Y. Chen, M.-H. Lai, Y.-C. Chen, and Y.-F. Kuo, "Detecting bakanae disease in rice seedlings by machine vision," *Computers and electronics in agriculture*, vol. 121, pp. 404–411, 2016.

[15] S. Sengupta and W. S. Lee, "Identification and determination of the number of immature green citrus fruit in a canopy under different ambient light conditions," *Biosystems Engineering*, vol. 117, pp. 51–61, 2014.

[16] M. Ebrahimi, M. Khoshtaghaza, S. Minaei, and B. Jamshidi, "Vision-based pest detection based on svm classification method," *Computers and Electronics in Agriculture*, vol. 137, pp. 52–58, 2017.

[17] J. Senthilnath, A. Dokania, M. Kandukuri, K. Ramesh, G. Anand, and S. Omkar, "Detection of tomatoes using spectral-spatial methods in remotely sensed rgb images captured by uav," *Biosystems engineering*, vol. 146, pp. 16–32, 2016.

[18] M. Zhang, C. Li, and F. Yang, "Classification of foreign matter embedded inside cotton lint using short wave infrared (swir) hyperspectral transmittance imaging," *Computers and Electronics in Agriculture*, vol. 139, pp. 75–90, 2017.

[19] A. Adedoja, P. A. Owolawi, and T. Mapayi, "Deep learning based on nasnet for plant disease recognition using leave images," in *2019 International Conference on Advances in Big Data, Computing and Data Communication Systems (icABCD)*, pp. 1–5, IEEE, 2019.

[20] M. R. Howlader, U. Habiba, R. H. Faisal, and M. M. Rahman, "Automatic recognition of guava leaf diseases using deep convolution neural network," in *2019 International Conference on Electrical, Computer and Communication Engineering (ECCE)*, pp. 1–5, IEEE, 2019.

[21] M. Sardogan, A. Tuncer, and Y. Ozen, "Plant leaf disease detection and classification based on cnn with lvq algorithm," in *2018 3rd International Conference on Computer Science and Engineering (UBMK)*, pp. 382–385, IEEE, 2018.

[22] P. Ouppaphan, "Corn disease identification from leaf images using convolutional neural networks," in *2017 21st International Computer Science and Engineering Conference (ICSEC)*, pp. 1–5, IEEE, 2017.

[23] J. Li, J. Jia, and D. Xu, "Unsupervised representation learning of image-based plant disease with deep convolutional generative adversarial networks," in *2018 37th Chinese Control Conference (CCC)*, pp. 9159–9163, IEEE, 2018.

[24] H. Durmuş, E. O. Güneş, and M. Kırcı, "Disease detection on the leaves of the tomato plants by using deep learning," in *2017 6th International Conference on Agro-Geoinformatics*, pp. 1–5, IEEE, 2017.

[25] V. Mazzia, A. Khaliq, F. Salvetti, and M. Chiaberge, "Real-time apple detection system using embedded systems with hardware accelerators: An edge ai application," *IEEE Access*, vol. 8, pp. 9102–9114, 2020.

[26] D. Mwiti, "A 2019 guide to object detection," (Jul 18, 2019) (accessed on Sep 10, 2019). Website Link.

[27] S. Ren, K. He, R. Girshick, and J. Sun, "Faster r-cnn: Towards real-time object detection with region proposal networks," in *Advances in neural information processing systems*, pp. 91–99, 2015.

[28] J. Hui, "Object detection: speed and accuracy comparison (faster r-cnn, r-fcn, ssd, fpn, retinanet and yolov3)," (Mar 18, 2018) (accessed on Jan 9, 2020). Website Link.

[29] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. Reed, C.-Y. Fu, and A. C. Berg, "Ssd: Single shot multibox detector," in *European conference on computer vision*, pp. 21–37, Springer, 2016.

[30] J. Hui, "Ssd object detection," (Mar 14, 2018) (accessed on Sep 20, 2019). Website Link.

[31] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam, "Mobilenets: Efficient convolutional neural networks for mobile vision applications," *arXiv preprint arXiv:1704.04861*, 2017.

[32] Tzutalin, "Labelimg label soft.," (2015) (accessed on Sep 9, 2019). Website Link.

[33] "Fully connected layers in convolutional neural networks: The complete guide," (2019) (accessed on Sep 9, 2019). Website Link.

[34] K. He, X. Zhang, S. Ren, and J. Sun, "Delving deep into rectifiers: Surpassing human-level performance on imagenet classification," in *Proceedings of the IEEE international conference on computer vision*, pp. 1026–1034, 2015.

[35] D.-A. Clevert, T. Unterthiner, and S. Hochreiter, "Fast and accurate deep network learning by exponential linear units (elus)," *arXiv preprint arXiv:1511.07289*, 2015.

[36] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: a simple way to prevent neural networks from overfitting," *The journal of machine learning research*, vol. 15, no. 1, pp. 1929–1958, 2014.

[37] "Cnn architectures: Lenet, alexnet, vgg, googlenet, resnet," (Nov 16, 2017) (accessed on Nov 9, 2019). Website Link.

[38] G. E. Hinton, N. Srivastava, A. Krizhevsky, I. Sutskever, and R. R. Salakhutdinov, "Improving neural networks by preventing co-adaptation of feature detectors," *arXiv preprint arXiv:1207.0580*, 2012.

[39] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna, "Rethinking the inception architecture for computer vision," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 2818–2826, 2016.

[40] L. Lazebnik, "Convolutional neural network architectures: from lenet to resnet," *Presentation, University of Illinois, accessed*, vol. 20, 2018.

[41] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, *et al.*, "Tensorflow: Large-scale machine learning on heterogeneous distributed systems," *arXiv preprint arXiv:1603.04467*, 2016.

[42] kaggle PlantVillage Dataset, "Dataset of diseased plant leaf images and corresponding labels.," (2019) (accessed on Feb 9, 2020). Website Link.

[43] N. hardware Department, "Nvidia jetson nano specification," (2019) (accessed on Feb 9, 2020). Website Link.

[44] I. I. Department, "Intel camera module specification," (2019) (accessed on Jan 9, 2020). Website Link.

[45] "Accuracy, precision, recall or f1?," (Mar 15, 2017) (accessed on Nov 9, 2019). Website Link.