**Deep Learning for Sustainable Farming: Multi-Class Disease Detection in Apple and Tomato Leaves**

**A CAPSTONE PROJECT REPORT**

*Submitted in partial fulfillment of the*

*requirement for the award of the*

*Degree of*

**BACHELOR OF TECHNOLOGY**

**IN**

**COMPUTER SCIENCE & ENGINEERING**

*by*

**Dachepalli Sasank (21BCE9133)**

**Ryali Purna Sri Sai Sumanth (21BCE9227)**

**Chennupati Venkata Lakshman (21BCE9055)**

**Chennupati Venkata Ram (21BCE9790)**

*Under the Guidance of*

**DR. Naga Jagadesh Bommagani**



SCHOOL OF COMPUTER SCIENCE AND ENGINEERING

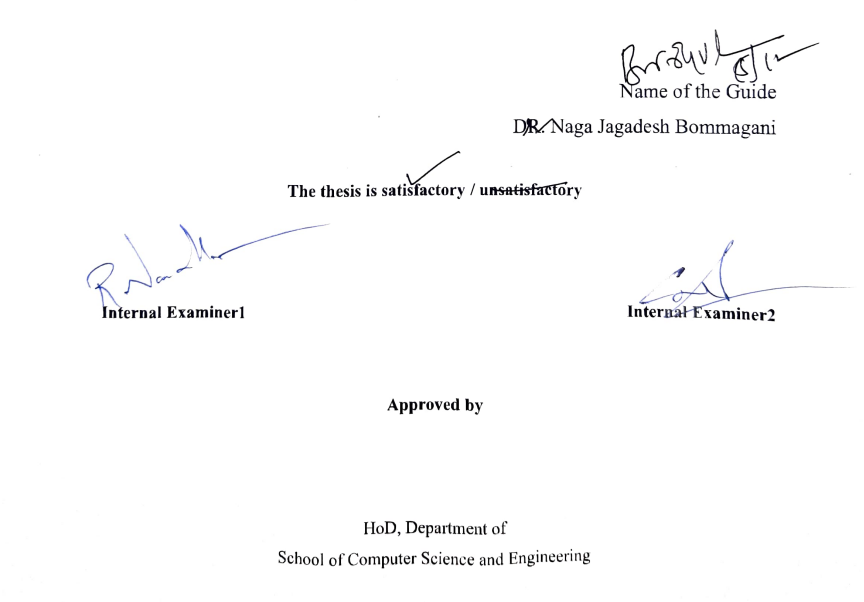
VIT-AP UNIVERSITY

AMARAVATI- 522237

*DECEMBER 2024*

**CERTIFICATE**

This is to certify that the Capstone Project work titled “**Deep Learning for Sustainable Farming: Multi-Class Disease Detection in Apple and Tomato Leaves**” that is being submitted by **Dachepalli Sasank (21BCE9133), Ryali Purna Sri Sai Sumanth (21BCE9227), Chennupati Venkata Lakshman (21BCE9055), And Chennupati Venkata Ram (21BCE9790)** is in partial fulfillment of the requirements for the award of Bachelor of Technology, is a record of Bonafide work done under my guidance. This Project's contents, in whole or in part, have not been taken from another source or submitted to another university or institute for the granting of a degree or certificate. It is certified.

****

Name of the Guide

DR. Naga Jagadesh Bommagani

**The thesis is satisfactory / unsatisfactory**

**Internal Examiner1 Internal Examiner2**

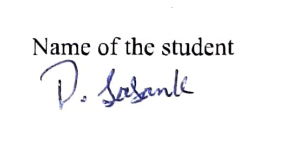
**Approved by**

HoD, Department of Artificial Intelligence and Machine Learning,

School of Computer Science and Engineering

**ACKNOWLEDGEMENTS**

I am delighted to convey my profound appreciation to **Dr. Naga Jagadesh Bommagani**, Professor (Grade 1), SCOPE, VIT-AP University, for his unwavering support, encouragement, and comprehension. Most importantly, he instilled in me the virtue of forbearance in my pursuit. My association with him is not confined to academics only, but also it is an exceptional opportunity for me to collaborate with a distinguished intellectual and expert in the fields of Artificial Intelligence and Machine Learning. I am grateful to **Dr. G. Viswanathan, Dr. Sekar Viswanathan, Dr. S. V. Kota Reddy, and Dr. S Sudhakar Ilango, SCOPE**, for the pleasant work environment and for their encouragement throughout the duration of the course. In jubilant mood I express ingeniously my whole-hearted thanks to **HoD - Artificial Intelligence and Machine Learning, Dr Reeja S R**, all of the instructors and personnel that serve as branches of our university for their selfless passion and prompt support, which I much appreciate, which prompted the acquirement of the requisite knowledge to finalize my course study successfully. I would like to thank my parents for their support. It is indeed a pleasure to thank my friends who persuaded and encouraged me to take up and complete this task. At last, but not least, I express my gratitude and appreciation to all those who have helped me directly or indirectly toward the successful completion of this project.



Name of the student

Place: Amaravati

Date: 22-11-2024

**ABSTRACT**

Improving agricultural production and overall food quality requires accurate identification of diseases affecting tomato and apple leaf. Traditional disease detection techniques are time-consuming and require specific knowledge, making them unsuitable for widespread agricultural observation. An automated approach for disease classification is presented in this paper. The comparison research was conducted using a dataset of apple and tomato plant leaf images to compare seven different deep-learning architectures, which are EfficientNet, ResNet, DenseNet, MobileNet, AlexNet, Xception, and GoogleNet. Regarding efficiency and compatibility in limited in resources contexts, GoogleNet outperformed other models with an impressive accuracy rate of 95.84% at the end of the training process. Data augmentation methods were used with a variety of data prior treatment methods, including picture scaling and normalisation, to increase the model's possibility of generalisation. The results of our investigation suggest that the lightweight GoogleNet architecture is especially advantageous for real-world applications in the detection of agricultural diseases, providing a swift, accurate, and mobile-adaptable solution. Subsequent research initiatives will focus on broadening the dataset while simultaneously optimizing the model to attain improved accuracy across a more extensive range of agricultural diseases.

**TABLE OF CONTENTS**

|  |  |  |  |
| --- | --- | --- | --- |
| **Sl.No.** | **Chapter** | **Title** | **Page Number** |
| **1.** |  | **Acknowledgement** | **3** |
| **2.** |  | **Abstract** | **4** |
| **3.** |  | **List of Figures and Tables** | **6, 7** |
| **4.** | **1**  **1.1**  **1.2**  **1.3** | **Introduction**  **Objectives**  **Background and Literature Survey**  **Organization of the Report** | **8**  **9**  **9**  **12** |
| **5.** | **2**  **2.1**  **2.2**  **2.3**  **2.3.1**  **2.3.2** | **Apple & Tomato Leaves Disease Detection**  **Proposed System**  **Working Methodology**  **System Details**  **Software**  **Hardware** | **13**  **13**  **14**  **16**  **16**  **17** |
| **6.** | **3** | **Results and Discussion** | **18** |
| **7.** | **4** | **Conclusion And Future Works** | **23** |
| **8.** | **5** | **Appendix** | **24** |
| **9.** | **6** | **References** | **45** |

**List of Tables**

|  |  |  |
| --- | --- | --- |
| **Table No.** | **Title** | **Page No.** |
|  | Classification Report of GoogleNet | **19** |
|  | Accuracy Table of All Models | 19 |

**List of Figures**

|  |  |  |
| --- | --- | --- |
| **Figure No.** | **Title** | **Page No.** |
|  | GoogleNet Workflow | 12 |
|  | GoogleNet Architecture | 13 |
|  | Inception Module with Dimensionality Reduction | 14 |
|  | Screen 1 | 16 |
|  | Screen 2 | 16 |
|  | GoogleNet Accuracy and Loss plot | 18 |
|  | GoogleNet Confusion matrix | 18 |
|  | DenseNet Accuracy plot | 20 |
|  | AlexNet Accuracy plot | 20 |
|  | ResNet Accuracy plot | 20 |
|  | Xception Accuracy plot | 21 |
|  | EfficientNet Accuracy plot | 21 |
|  | Mobilenet Accuracy plot | 21 |

**CHAPTER 1**

**INTRODUCTION**

To ensure the maintenance of crop health and mitigate production deficits, modern agricultural practices are profoundly reliant on the precise identification of plant diseases. In particular, species of apple and tomato are extensively cultivated owing to their considerable nutritional benefits and economic importance. However, a maximum of diseases that negatively influence plant health can severely impact these crops, resulting in a decrease in both the quality and yield of the harvest. Prevalent diseases that compromise the leaves of Apple cultivation include apple scab, black rot, and cedar apple rust, while bacterial spots, early blight, and late blight are notable diseases that adversely affect the leaves of Tomato. In these agricultural systems, the timely and precise identification of diseases can greatly reduce the spread of infections and boost total agricultural productivity.

Automated methodologies that leverage leaf imagery for the diagnostic assessment of plant diseases have emerged, attributable to recent advancements in image processing and computer vision technologies. These algorithms adeptly classify diseases predicated upon the visual attributes of leaves, utilizing either deep learning techniques or manually engineered feature extraction methodologies. Although handcrafted methods have demonstrated efficacy across a wide array of classification tasks, including the identification of plant diseases, deep learning-based approaches have gained heightened favor due to their superior accuracy and robust generalization capabilities in complex classification contexts. The predominant inclination within the realm of the study of plant diseases classification favors the utilization of deep learning frameworks, particularly Convolutional Neural Networks (CNNs), which have garnered prominence due to their operational effectiveness and performance optimization. In the present study, we introduce an optimized and efficient GoogleNet model specifically designed for the classification of diseases afflicting tomato and apple foliage.

The exceptional computational efficiency inherent to the GoogleNet architecture constitutes one of its principal advantages, rendering it particularly advantageous for mobile and edge devices that face inherent limitations concerning memory capacity and processing power. The primary aim of this academic investigation is to construct a model that achieves improved accuracy while concurrently reducing computational demands, thereby enabling its deployment in resource-constrained, real-time environments, such as compact unmanned aerial vehicles or mobile application frameworks. We performed GoogleNet architecture compared against deep learning frameworks, encompassing EfficientNet, ResNet, DenseNet, MobileNet, AlexNet, Xception employing a substantial dataset of images that depict both healthy and infected leaves of tomato and apple plants

* 1. **Objectives**

The objectives of the project are:

* Develop a robust deep learning-based system to accurately identify and classify diseases in Apple and Tomato leaves across 14 distinct categories.
* Evaluate the performance of various deep learning models, Like EfficientNet, GoogleNet, etc. to determine the most effective architecture for this task.
* Find the model with maximum of accuracy, precision, recall, and F1 score in the classification of leaf diseases for practical usage.
* Design and implement a computationally efficient model with low latency and minimal resource requirements to support real-time disease detection in agricultural settings.
* Include data augmentation and regularization techniques to improve the model's ability to applicable across different and unseen data samples.
* Conduct an analysis of training and validation metrics, including accuracy, loss, and overfitting patterns, to ensure the stability and reliability of the final model
* Develop a solution that is both user-friendly and scalable, which will assist agricultural professionals and producers in the early detection and prevention of diseases, thereby increasing crop yield and reducing losses.
  1. **Background and Literature Survey**

Convolutional neural networks (CNNs) have been employed to establish a methodological framework designed for the detection of diseases affecting tomato leaf, thereby highlighting the paramount importance of timely identification to mitigate economic detriments for agricultural stakeholders. By automating both the feature extraction and classification procedures, this framework—which categorizes diseases into ten distinct classifications—attained a notable accuracy rate of 93%, thereby highlighting the practical implications of the model and its prospective utility for immediate interventions in the management of tomato-associated diseases [1]. A focused investigation was conducted to discern ailments in custard apple foliage, thereby emphasizing the critical importance of precise classification to reduce pesticide utilization and improve agricultural productivity. Methods like image segmentation and support vector machine (SVM) classifiers were applied to classify custard apple diseases, showing the benefits of advanced image processing techniques in agricultural applications. [2]

In light of the significant economic implications for apple cultivators, a considerable number of scholarly research endeavors concentrate on the detection of leaf diseases in Apple employing advanced deep learning methodologies. One investigation examined the progress of Convolutional Neural Networks (CNNs) alongside the limitations inherent in widely utilized datasets such as PlantVillage, which is commonly deployed for the assessment of agricultural ailments. This investigation highlights the imperative for the advancement of methodologies proficient in the prompt identification of leaf diseases to protect Apple plantations and augment the financial sustainability of agricultural producers [3]. In an alternative investigation, the diseases of apple and grapevine plant leaves were arranged using Convolutional Neural Networks (CNNs) in conjunction with Multi-Layer Perceptron (MLP) architectures The results indicated that CNNs exhibited superior performance relative to MLPs with respect to both precision and effectiveness. Employing the PlantVillage dataset, this research substantiated that CNNs are exceedingly suitable for applications in real-time agriculture [4].

BY comparing Support Vector Machine (SVM), Random Forest, Naive Bayes, K-Nearest Neighbor, and VGG-16 models, the research demonstrated the advantage of machine learning and deep learning methodologies in the identification of diseases affecting apple crops. The VGG-16 model attained the apex of validation accuracy at 97.23%, thus asserting its position as a reliable tool for the categorization of apple leaf diseases. The investigation also highlights the importance of carefully selecting appropriate models in accordance with specific project requirements, thereby clarifying the complex nature of automated disease detection across various agricultural paradigms [5]. In the case of tomato disease identification, a well-balanced dataset was constructed using data augmentation techniques which enabled Convolutional Neural Network (CNN) models such as MobileNetV2 in achieving accuracy rate of 98.01% in disease classification. This study highlights the critical role of balanced datasets and deep learning methodologies in the management of heterogeneous agricultural data, thereby positioning MobileNetV2 as a superior option for real-time applications due to its proficiency in the detection of tomato diseases [6].

A study on the detection of diseases affecting tomato leaves using the YOLOv8 framework, attaining an classification accuracy of 99% and a mean Average Precision (mAP) of 98.9%. Assessed against a dataset consisting of 1,650 images, this model proficiently distinguished between healthy and diseased tomato foliage, thereby highlighting the efficacy of real-time computer vision in the enhancement of agricultural health management [7]. For the classification of diseases affecting tomatoes, hybrid models that integrate Convolutional Neural Networks (CNN) with Long Short-Term Memory (LSTM) architectures were also used, these models attained a accuracy rate of 99.8%. Evaluated on a dataset, this methodology exemplified the efficacy of hybrid deep learning models in reliably discerning various plant diseases, thus offering a robust approach for timely agricultural interventions [8].

To achieve an accuracy rate of 96%, an alternative system specifically targeting Black rot and Cedar apple rust in apple foliage employed preprocessing and image segmentation techniques prior to the application of Multiclass Support Vector Machine (SVM) for classification purposes. This investigation illustrates the effective synergy of image processing and SVM methodologies in addressing specific diseases afflicting apple crops [9]. In conclusion, a specialized online platform has been developed to detect tomato illnesses, making it easier for agronomists to quickly evaluate the health of tomato leaves by uploading images. The model's remarkable 93% accuracy rate highlights the fascinating potential of easily available internet tools for smallholder farmers in areas like Zimbabwe, where economic stability is largely determined by agricultural productivity [10].

In conclusion, our research methodologies employing Convolutional Neural Networks (CNNs) for the detection of diseases in tomato and apple agriculture demonstrated that automated feature extraction and classification can achieve a significant degree of precision, which prompted us to explore analogous architectural paradigms. Our objectives were centered on both precision and computational efficacy, leading us to select and optimize our model based on academic literature that underscored the necessity of dataset equilibrium to improve the feasibility of real-time implementations, utilizing robust architectures such as ResNet, DenseNet, EfficientNet, and GoogLeNet. Our decision to investigate a spectrum of architectures with the objective of identifying the most suitable model for our investigation was informed by studies that similarly emphasized the significance of specific algorithms designed for a diverse range of agricultural crops.

**1.3 Organization of the Report**

The remaining chapters of the project report are described as follows:

* Chapter 2 contains the proposed system, methodology, hardware and software details.
* Chapter 3 discusses the results obtained after the project was implemented.
* Chapter 4 concludes the report.
* Chapter 5 consists of codes.
* Chapter 6 gives references.

**CHAPTER 2**

**Deep Learning for Sustainable Farming: Multi-Class Disease Detection in Apple and Tomato Leaves**

This Chapter describes the proposed system, working methodology, software and hardware details.

**2.1 Proposed System**

In this study, we employed GoogleNet (also known as Inception V1) as the final model for identifying diseases in the leaves of Apple and Tomato plants. GoogleNet was selected based on its excellent performance in categorizing leaf images into one of fourteen distinct disease classes. The model achieved impressive results during the evaluation phase, yielding the following validation metrics.

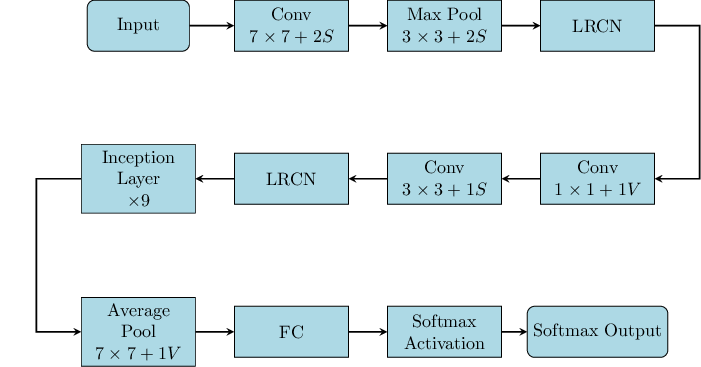


Figure 1. GoogleNet Workflow

**2.2 Working Methodology**

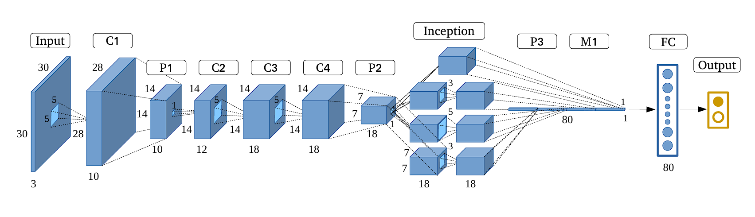
GoogleNet epitomizes an advanced deep convolutional neural network (CNN) architecture that has been rigorously designed to improve both precision and computational efficacy It introduces an innovative architectural component known as the Inception module, which empowers the network to assimilate diverse levels of abstraction by concurrently applying multiple convolutional filters with varying kernel dimensions. This architectural framework exhibits exceptional efficacy in capturing intricate features within images while concurrently preserving a compact model size GoogleNet, commonly designated as Inception v1, represents a convolutional neural network (CNN) architecture that initiated the deployment of the Inception module to improve both computational efficacy and model precision. This network utilizes a multi-branch arrangement within the Inception module, amalgamating 1x1, 3x3, and 5x5 convolutional filters, alongside max-pooling, all executed concurrently. The amalgamation of these outputs enables the extraction of multi-scale spatial features, thereby augmenting the model's capability to discern intricate patterns.

Figure 2. GoogleNet Architecture

GoogleNet implements dimensionality reduction via 1x1 convolutions to diminish computational burden. The architecture is deeper than its predecessors, consisting of 22 layers, yet achieves efficiency with a reduced number of parameters owing to the Inception modules. Furthermore, auxiliary classifiers are incorporated at intermediate stages of the network to facilitate gradient propagation during the training phase and to enhance overall model performance. The design of GoogleNet underscores a critical equilibrium between computational expense and accuracy, marking it as a seminal advancement in the realm of deep learning. For this work, we have taken a pre-trained GoogleNet architecture, which had undergone prior training utilizing the ImageNet dataset.

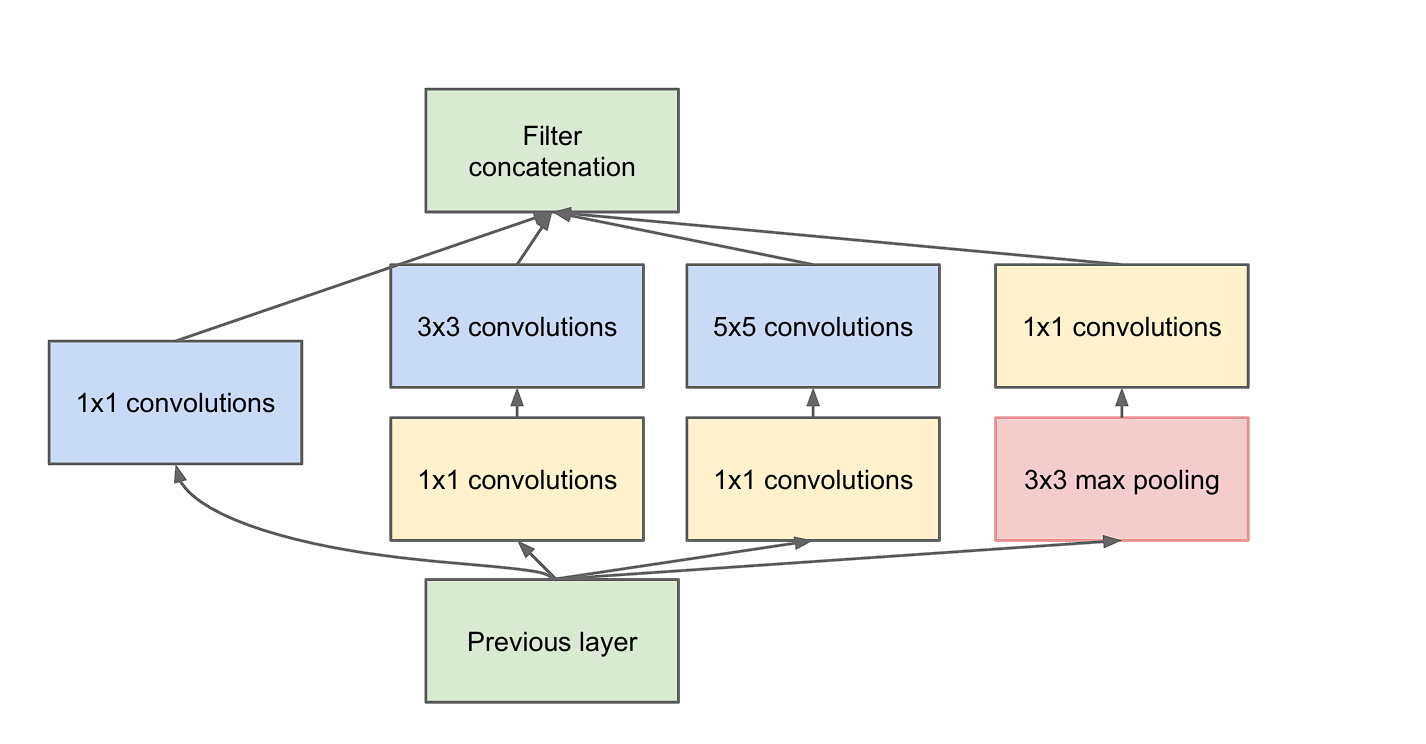
This methodology of transfer learning enabled us to capitalize on the pre-trained model's capacity to discern general visual patterns, thereby significantly curtailing training duration and enhancing accuracy. Subsequently, the pre-trained model underwent fine-tuning through the incorporation of custom output layers specifically designed for the 14 disease categories pertinent to our dataset We implemented the Adam optimization algorithm during the training phase, commencing with a learning rate of 0.001 for the first 20 epochs. The Adam optimization algorithm amalgamates the beneficial attributes of the Adaptive Gradient Algorithm (AdaGrad) and Root Mean Square Propagation (RMSProp) to dynamically adjust the learning rate for individual parameters, thus rendering it especially beneficial for datasets that demonstrate traits of noisy or sparse gradients.

Figure 3. Inception Module with Dimensionality Reduction

During this phase, only the custom output layers were subjected to training, while the pre-trained base layers were retained in a frozen state to safeguard the acquired features. Subsequent to the preliminary training phase, we unblocked the terminal layers of the GoogleNet architecture and conducted fine-tuning for an extended duration of 10 epochs, simultaneously diminishing the learning rate to 0.0001 to facilitate stable convergence. The model was trained utilizing a batch size of 32 images, which optimized memory utilization and processing efficiency. Throughout the training regimen, both training and validation accuracy and loss metrics were meticulously monitored. The model exhibited a consistent enhancement in accuracy, ultimately attaining a peak validation accuracy of 95.84%. This improvement was concomitant with a steady reduction in loss, signifying that the model was effectively acquiring the capability to classify the leaf images. The accuracy plots for both training and validation illustrated a stable training trajectory with negligible indications of overfitting.

In summary, GoogleNet was chosen as the final model because of its outstanding results, which included a computationally efficient solution for real-time disease detection in plant leaves and an accuracy of 95.84%. The model is a great option for real-world applications where accuracy and efficiency are remarkable due to its capacity to capture a variety of image data and the advantages of transfer learning.

**2.3 System Details**

This section describes the software and hardware details of the system:

**2.3.1 Software Details**

* **Language**: Python 3. 10. x
* **Libraries**: numpy, pandas, matplotlib.pyplot, os, glob, seaborn, cv2, io, tensorflow, sklearn, tqdm, keras, PIL.
* **IDE**: Jupyter Notebook, VS Code, or PyCharm.
* **GPU Support**: CUDA Toolkit (if using NVIDIA GPU).
* **Cloud Option**: Google Colab with GPU.

1. **Web Application**

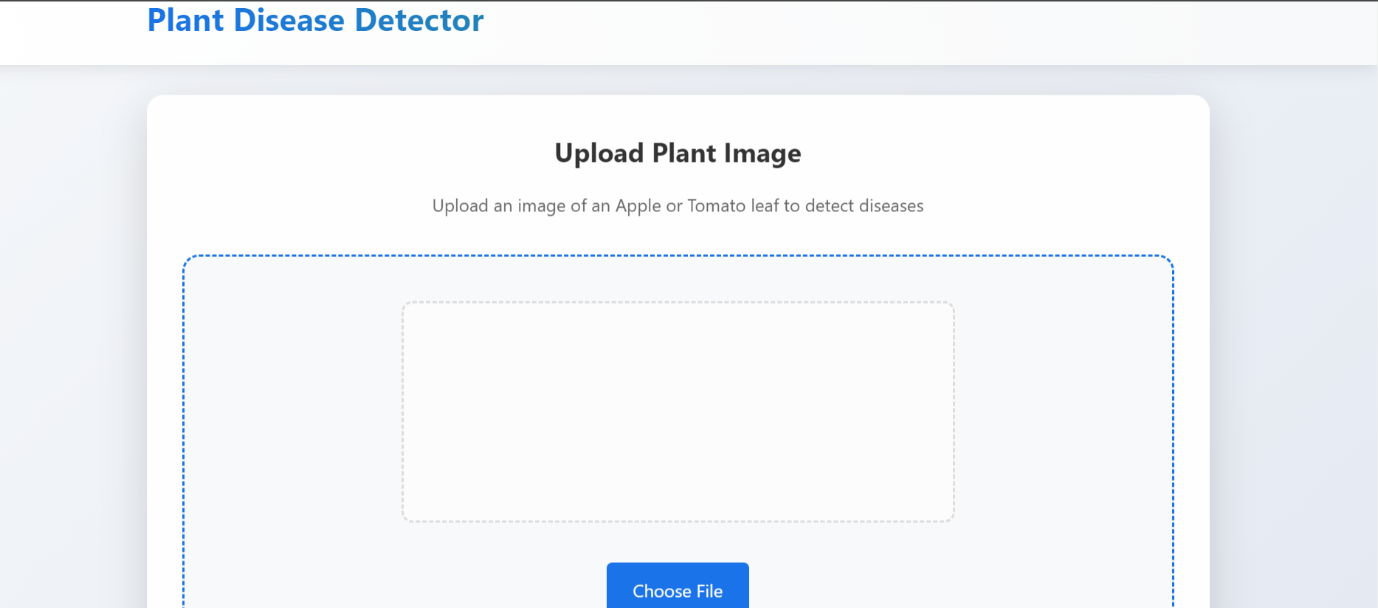


Figure 4. Screen 1

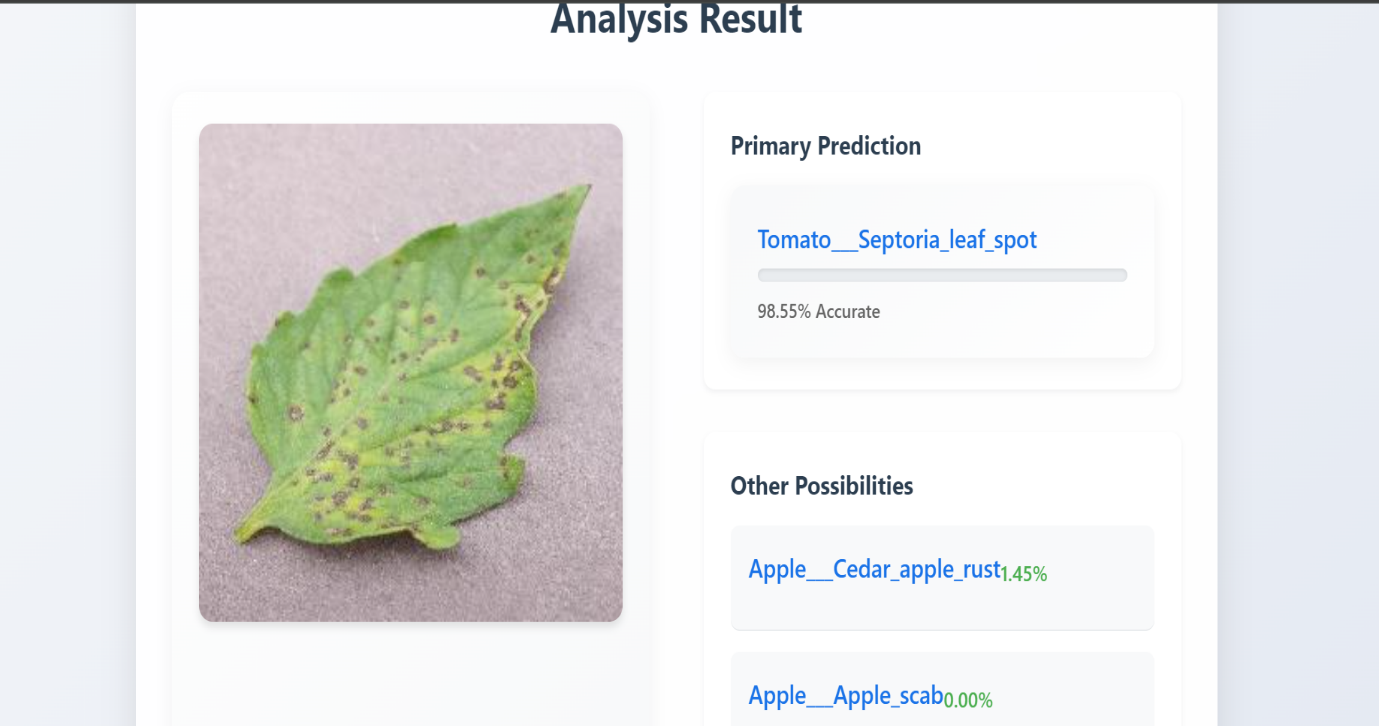


Figure 5. Screen 2

**2.3.2 Hardware Details**

* **CPU**: Intel i5 or AMD Ryzen 7 (or higher), 6+ cores.
* **GPU**: NVIDIA GTX 1650 with 4 GB VRAM, or cloud-based GPU (Google Colab).
* **RAM**: 8GB (16 GB recommended).
* **Storage**: 512 GB SSD (1 TB recommended).
* **OS**: Windows 10/11, Linux (Ubuntu), or macOS.

**CHAPTER 3**

**RESULTS AND DISCUSSIONS**

The empirical findings substantiate that GoogleNet (Inception v1) exhibits exceptional efficacy in the identification of diseases affecting tomato and apple foliage, attaining an impressive validation accuracy of 95.84%. This establishes GoogleNet as one of the foremost models within the analytical framework, illustrating its capability to harmonize accuracy with computational efficiency. The variance in performance among different architectures is pronounced, as evidenced by models such as Xception, which registered a significantly lower accuracy of 56.41%, while alternative configurations like DenseNet secured a commendable 93.14%. In comparison, MobileNet, characterized as a lightweight architecture, achieved a marginally lower accuracy of 89.07%, reflecting the heterogeneity in performance across various models.

The Inception module-based architecture of GoogleNet, meticulously designed to adeptly process multi-scale features, is a significant factor contributing to its outstanding performance. The neural network demonstrated exceptional performance on critical evaluative indicators, encompassing precision (96.02%), recall (95.84%), and F1-score (95.85%), thus substantiating its resilience in classification endeavors. These outcomes underscore the dependability of GoogleNet in real-world scenarios pertaining to disease identification.

Furthermore, a comparative assessment was undertaken to scrutinize model parameters, inference duration, and memory prerequisites. GoogleNet achieves a commendable equilibrium between accuracy and computational requisites, surpassing resource-demanding models such as DenseNet in terms of efficiency while attaining comparable or superior accuracy. Its performance regarding computational expense and inference velocity renders it particularly advantageous for real-time applications, especially in contexts necessitating on-the-go disease detection.

The proficiency of GoogleNet in accurately identifying leaf diseases with high precision and reliability accentuates its potential as a viable solution for agricultural diagnostics. Its equilibrium of performance and efficiency designates it as an optimal candidate for deployment in settings with constrained computational resources, such as mobile or edge devices utilized in the field. These findings validate the model's appropriateness for assisting farmers and agricultural specialists in acquiring immediate and precise assessments of plant health, providing a robust instrument for contemporary precision agriculture

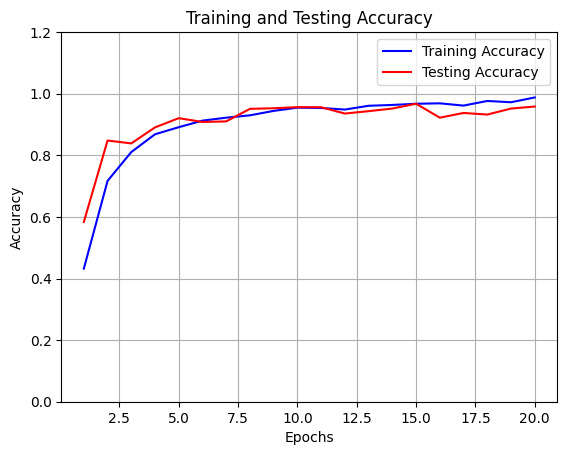
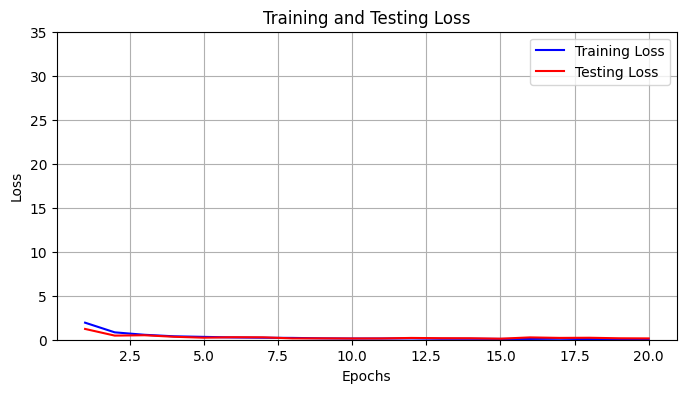


Fig. 6: GoogleNet Accuracy and Loss plot

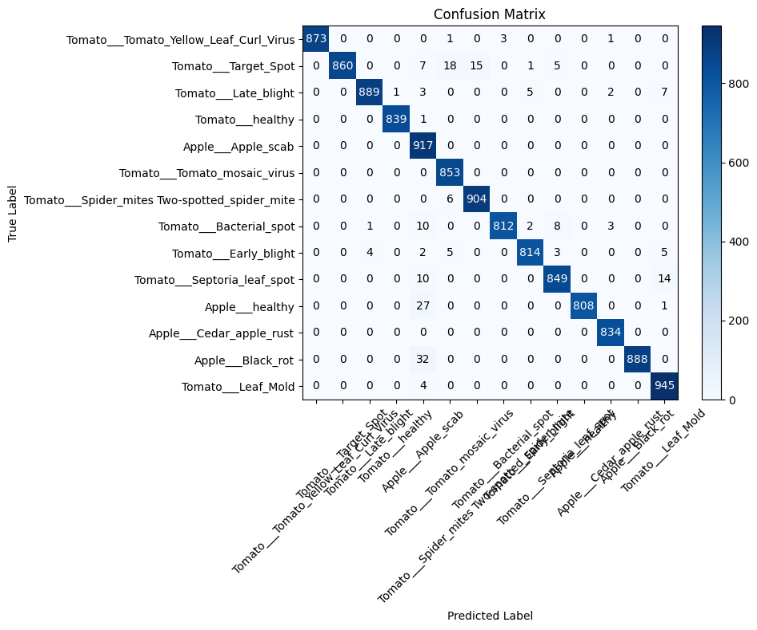


Fig. 7: GoogleNet Confusion matrix

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Class** | **Precision** | **Recall** | **F1-Score** | **Support** |
| Tomato Yellow Leaf Curl Virus | 1.00 | 0.96 | 0.98 | 224 |
| Tomato Target Spot | 0.98 | 0.91 | 0.95 | 217 |
| Tomato Late blight | 0.99 | 0.93 | 0.96 | 225 |
| Tomato healthy | 1.00 | 0.99 | 1.00 | 233 |
| Apple scab | 0.87 | 0.99 | 0.93 | 225 |
| Tomato mosaic virus | 0.95 | 1.00 | 0.97 | 204 |
| Tomato Spider mites Two-spotted spider mite | 0.96 | 0.97 | 0.97 | 207 |
| Tomato Bacterial spot | 0.98 | 0.93 | 0.95 | 233 |
| Tomato Early blight | 0.94 | 0.92 | 0.93 | 223 |
| Tomato Septoria leaf spot | 0.93 | 0.93 | 0.93 | 229 |
| Apple healthy | 0.99 | 0.94 | 0.97 | 198 |
| Apple Cedar rust | 0.96 | 1.00 | 0.97 | 215 |
| Apple Black rot | 0.99 | 0.95 | 0.97 | 213 |
| Tomato Leaf Mold | 0.91 | 1.00 | 0.95 | 228 |

Table. 1: Classification Report of GoogleNet

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Model** | **Validation Accuracy** | **Validation Precision** | **Validation Recall** | **Validation F1 Score** |
| **EfficientNet** | 95.84% | 95.88% | 95.84% | 95.83% |
| **DenseNet** | 93.14% | 93.36% | 93.14% | 93.17% |
| **GoogleNet** | 95.84% | 96.02% | 95.84% | 95.85% |
| **Xception** | 56.41% | 57.33% | 56.41% | 54.94% |
| **ResNet** | 95.84% | 95.91% | 95.84% | 95.83% |
| **AlexNet** | 90.73% | 90.85% | 90.73% | 90.70% |
| **MobileNet** | 89.07% | 89.25% | 89.07% | 89.00% |

Table. 2: Accuracy Table of All Models

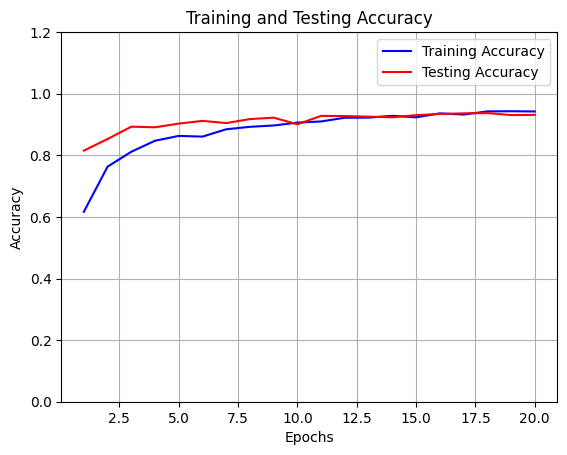


Fig. 8: DenseNet Accuracy plot

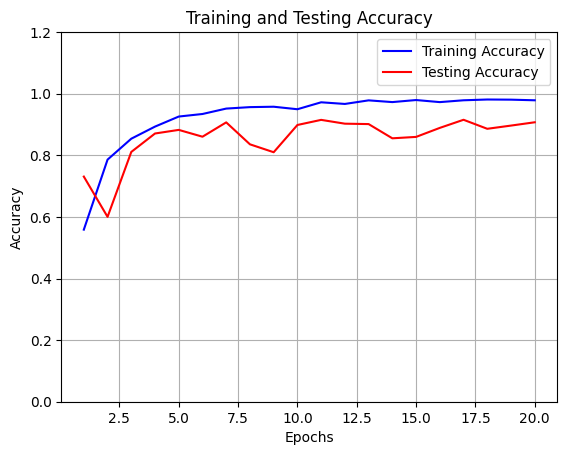


Fig. 9: AlexNet Accuracy Plot

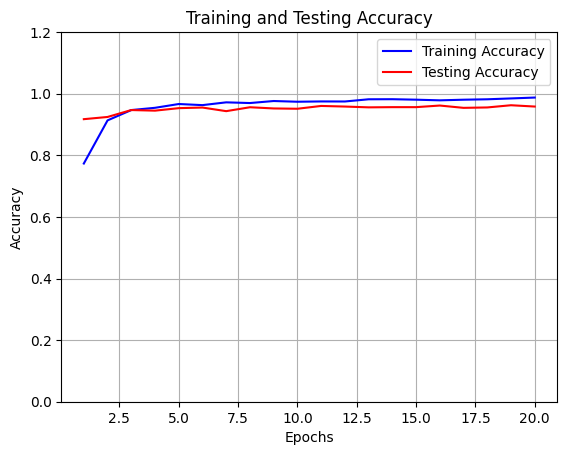


Fig. 10: ResNet Accuracy Plot

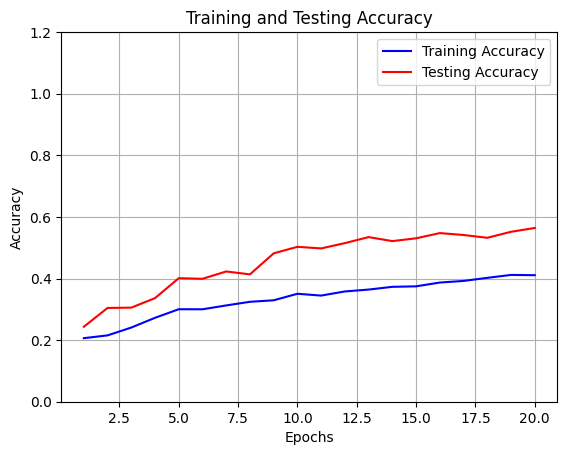


Fig. 11: Xception Accuracy Plot

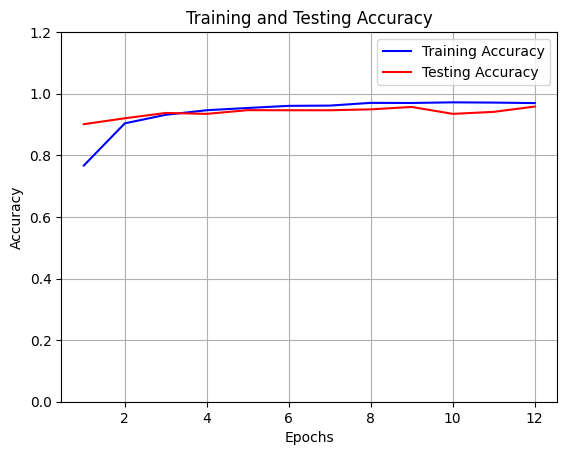


Fig. 12: EfficientNet Accuracy Plot

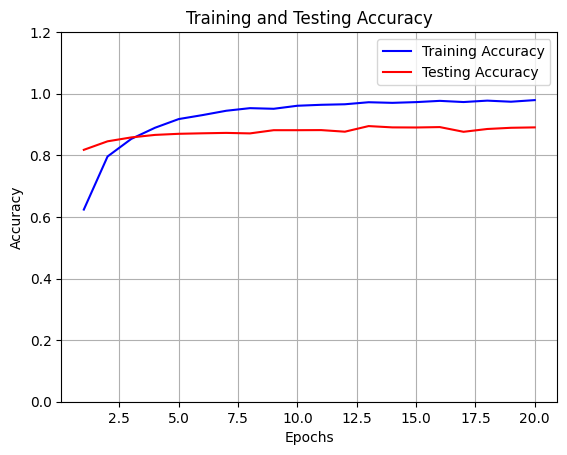


Fig. 13: MobileNet Accuracy Plot

**CHAPTER 4**

**CONCLUSION AND FUTURE WORK**

**5.1 Conclusion:**

In order to discern diseases within the leaf structures of Tomato and Apple, the current research initiative elucidates a highly efficacious model based on GoogleNet, which achieves an exceptional validation accuracy of 95.84% while demanding minimal computational resources. In comparison to other sophisticated deep learning architectures, such as EfficientNet, ResNet, DenseNet, MobileNet, AlexNet, and Xception, GoogleNet exhibits the great balance between augmented accuracy and operational efficiency The optimized and computationally frugal architecture of GoogleNet enables real-time disease detection, thus providing a pragmatic solution for agricultural practitioners and specialists who necessitate rapid, on-site identification of leaf diseases.

The findings accentuate the efficacy of GoogleNet as a cost-effective and scalable tool for the identification of disease, which holds the potential to transform disease management strategies within the agricultural domain by providing accessible and precise diagnostic tools. This study constitutes a substantial contribution to the realm of agricultural technology through the development of a model that facilitates efficient, instantaneous monitoring of plant health, thereby augmenting the capabilities for timely intervention.

**5.2 Future Work:**

Future research will be on how GoogleNet may be integrated with Internet of Things (IoT) sensors to enable automated and ongoing plant health monitoring in agricultural settings. Furthermore, using bigger and more varied datasets could improve our model's development and increase its flexibility in response to changing environmental circumstances. To optimize performance on computing systems with increasingly reduced resources, alternative model changes such as pruning and quantization will be systematically explored. In remote locations with limited resources, these advancements could significantly increase the precision and efficacy of Leaf disease monitoring systems.

**CHAPTER 5**

**APPENDIX**

**GoogleNet Code:**

import os

folder\_path = '/content/UpdatedDataset/Updated'

subfolders = [f for f in os.listdir(folder\_path) if os.path.isdir(os.path.join(folder\_path, f))]

print(subfolders)

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

import os

import glob

import seaborn as sns

import cv2

import io

import tensorflow as tf

from sklearn.model\_selection import train\_test\_split

from tqdm import tqdm

import keras

from keras.preprocessing import image

#from keras.preprocessing.image import ImageDataGenerator

from PIL import Image

from sklearn.utils import shuffle

from keras import layers,models,optimizers

import ipywidgets as widgets

from keras.utils import to\_categorical

from keras.models import Sequential

from keras.layers import Dense, Activation, Dropout, Flatten, Conv2D, MaxPooling2D,Dropout,BatchNormalization,Concatenate,AveragePooling2D

from sklearn.metrics import classification\_report, confusion\_matrix

from keras.models import Sequential

from keras.regularizers import l2

from keras.callbacks import EarlyStopping

import glob

import cv2

X=[]

Y=[]

image\_size=(227,227)

for i in subfolders:

path="/content/UpdatedDataset/Updated/"+i+"/"

print(path)

fileRead=glob.glob(path+"\*")

print(len(fileRead))

for j in fileRead:

image=cv2.imread(j)

image=cv2.resize(image,image\_size)

X.append(image)

Y.append(i)

X = np.array(X)

Y = np.array(Y)

labels=subfolders

import numpy as np

Temp\_y = []

for i in Y:

Temp\_y.append(labels.index(i))

Y = np.array(Temp\_y)

Y = to\_categorical(Y)

X\_train,X\_test,Y\_train,Y\_test=train\_test\_split(X,Y,test\_size=0.2,random\_state=42)

from keras.models import Model

from keras.layers import Input, Conv2D, MaxPooling2D, AveragePooling2D, Dropout, Flatten, Dense, Concatenate

def inception\_module(prev\_layer, filters):

tower\_1x1 = Conv2D(filters[0], (1, 1), padding='same', activation='relu')(prev\_layer)

tower\_3x3\_reduce = Conv2D(filters[1], (1, 1), padding='same', activation='relu')(prev\_layer)

tower\_3x3 = Conv2D(filters[2], (3, 3), padding='same', activation='relu')(tower\_3x3\_reduce)

tower\_5x5\_reduce = Conv2D(filters[3], (1, 1), padding='same', activation='relu')(prev\_layer)

tower\_5x5 = Conv2D(filters[4], (5, 5), padding='same', activation='relu')(tower\_5x5\_reduce)

tower\_pool = MaxPooling2D((3, 3), strides=(1, 1), padding='same')(prev\_layer)

tower\_pool\_1x1 = Conv2D(filters[5], (1, 1), padding='same', activation='relu')(tower\_pool)

return Concatenate(axis=-1)([tower\_1x1, tower\_3x3, tower\_5x5, tower\_pool\_1x1])

# Input layer

input\_layer = Input(shape=(227, 227, 3))

# Convolution and Pooling layers

x = Conv2D(64, (7, 7), strides=(2, 2), padding='same', activation='relu')(input\_layer)

x = MaxPooling2D((3, 3), strides=(2, 2), padding='same')(x)

# Inception modules

x = inception\_module(x, [64, 128, 128, 32, 32, 32])

x = inception\_module(x, [128, 192, 96, 64, 64, 64])

x = MaxPooling2D((3, 3), strides=(2, 2), padding='same')(x)

x = inception\_module(x, [192, 208, 48, 64, 64, 64])

x = inception\_module(x, [160, 224, 64, 64, 64, 128])

x = MaxPooling2D((3, 3), strides=(2, 2), padding='same')(x)

x = inception\_module(x, [128, 256, 64, 64, 64, 128])

x = inception\_module(x, [112, 288, 64, 64, 64, 128])

x = inception\_module(x, [256, 320, 128, 128, 128, 128])

x = MaxPooling2D((3, 3), strides=(2, 2), padding='same')(x)

# Fully connected layers

x = AveragePooling2D(pool\_size=(4, 4))(x)

x = Flatten()(x)

x = Dense(1024, activation='relu')(x)

x = Dropout(0.4)(x)

output\_layer = Dense(14, activation='softmax')(x) # Assuming 4 classes for the classification task

# Create the model

GoogleNet = Model(inputs=input\_layer, outputs=output\_layer)

# Compile the model

from tensorflow.keras.optimizers import Adam

optimizer = Adam(learning\_rate=0.0001)

GoogleNet.compile(optimizer=optimizer, loss='categorical\_crossentropy', metrics=['accuracy'])

early\_stopping = EarlyStopping(monitor='val\_loss', patience=3)

# Print the model summary

GoogleNet.summary()

# Train the model

googlenet\_history = GoogleNet.fit(X\_train, Y\_train, validation\_data=(X\_test, Y\_test), epochs=20, batch\_size=32, verbose=1)

train\_loss =googlenet\_history.history['loss']

train\_accuracy = googlenet\_history.history['accuracy']

test\_loss = googlenet\_history.history['val\_loss']

test\_accuracy = googlenet\_history.history['val\_accuracy']

epochs = range(1, len(train\_loss)+1)

plt.plot(epochs, train\_accuracy, 'b', label='Training Accuracy')

plt.plot(epochs, test\_accuracy, 'r', label='Testing Accuracy')

plt.title('Training and Testing Accuracy')

plt.xlabel('Epochs')

plt.ylabel('Accuracy')

plt.ylim(0,1.2)

plt.legend()

plt.grid(True)

plt.show()

plt.figure(figsize=(8, 4))

plt.plot(epochs, train\_loss, 'b', label='Training Loss')

plt.plot(epochs, test\_loss, 'r', label='Testing Loss')

plt.title('Training and Testing Loss')

plt.xlabel('Epochs')

plt.ylabel('Loss')

plt.ylim(0,35)

plt.legend()

plt.grid(True)

plt.show()

from sklearn.metrics import accuracy\_score

from sklearn.metrics import precision\_score

from sklearn.metrics import recall\_score

from sklearn.metrics import f1\_score

from sklearn.metrics import confusion\_matrix

from sklearn.metrics import classification\_report

y\_pred\_test = GoogleNet.predict(X\_test)

# Convert predictions and true labels from one-hot encoding to class labels

y\_pred\_test\_labels = np.argmax(y\_pred\_test, axis=1)

y\_true\_test\_labels = np.argmax(Y\_test, axis=1)

# Compute validation accuracy

validation\_accuracy = accuracy\_score(y\_true\_test\_labels, y\_pred\_test\_labels)

# Compute precision, recall, and F1-score

validation\_precision = precision\_score(y\_true\_test\_labels, y\_pred\_test\_labels, average='weighted')

validation\_recall = recall\_score(y\_true\_test\_labels, y\_pred\_test\_labels, average='weighted')

validation\_f1 = f1\_score(y\_true\_test\_labels, y\_pred\_test\_labels, average='weighted')

# Print validation metrics

print(f"Validation Accuracy: {validation\_accuracy:.4f}")

print(f"Validation Precision: {validation\_precision:.4f}")

print(f"Validation Recall: {validation\_recall:.4f}")

print(f"Validation F1 Score: {validation\_f1:.4f}")

# Display classification report

print("\nClassification Report:")

print(classification\_report(y\_true\_test\_labels, y\_pred\_test\_labels, target\_names=labels))

y\_pred =GoogleNet.predict(X\_train)

# Convert predictions from one-hot encoding to class labels

y\_pred\_labels = np.argmax(y\_pred, axis=1)

y\_true\_labels = np.argmax(Y\_train, axis=1)

# Create the confusion matrix

cm = confusion\_matrix(y\_true\_labels, y\_pred\_labels)

# Display the confusion matrix as a heatmap

plt.figure(figsize=(10, 8))

plt.imshow(cm, interpolation='nearest', cmap=plt.cm.Blues)

plt.title('Confusion Matrix')

plt.colorbar()

tick\_marks = np.arange(len(labels))

plt.xticks(tick\_marks, labels, rotation=45)

plt.yticks(tick\_marks, labels)

plt.xlabel('Predicted Label')

plt.ylabel('True Label')

# Add text annotations for each cell in the heatmap

thresh = cm.max() / 2.

for i in range(cm.shape[0]):

for j in range(cm.shape[1]):

plt.text(j, i, format(cm[i, j], 'd'), ha="center", va="center", color="white" if cm[i, j] > thresh else "black")

plt.tight\_layout()

plt.show()

**MobileNet Code:**

import os

folder\_path = '/content/dataset/Updated'

subfolders = [f for f in os.listdir(folder\_path) if os.path.isdir(os.path.join(folder\_path, f))]

print(subfolders)

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

import os

import glob

import seaborn as sns

import cv2

import io

import tensorflow as tf

from sklearn.model\_selection import train\_test\_split

from tqdm import tqdm

import keras

from keras.preprocessing import image

#from keras.preprocessing.image import ImageDataGenerator

from PIL import Image

from sklearn.utils import shuffle

from keras import layers,models,optimizers

import ipywidgets as widgets

from keras.utils import to\_categorical

from keras.models import Sequential

from keras.layers import Dense, Activation, Dropout, Flatten, Conv2D, MaxPooling2D,Dropout,BatchNormalization,Concatenate,AveragePooling2D

from sklearn.metrics import classification\_report, confusion\_matrix

from keras.models import Sequential

from keras.regularizers import l2

from keras.callbacks import EarlyStopping

import glob

import cv2

X=[]

Y=[]

image\_size=(227,227)

for i in subfolders:

path="/content/dataset/Updated/"+i+"/"

print(path)

fileRead=glob.glob(path+"\*")

print(len(fileRead))

for j in fileRead:

image=cv2.imread(j)

image=cv2.resize(image,image\_size)

X.append(image)

Y.append(i)

X = np.array(X)

Y = np.array(Y)

labels=subfolders

import numpy as np

Temp\_y = []

for i in Y:

Temp\_y.append(labels.index(i))

Y = np.array(Temp\_y)

Y = to\_categorical(Y)

X\_train,X\_test,Y\_train,Y\_test=train\_test\_split(X,Y,test\_size=0.2,random\_state=42)

from tensorflow.keras.applications import MobileNet

# Load MobileNet with pre-trained weights

mobile\_base = MobileNet(weights='imagenet', include\_top=False, input\_shape=(227, 227, 3))

# Freeze pre-trained layers (optional)

mobile\_base.trainable = False

# Add custom layers for classification

model = Sequential()

model.add(mobile\_base)

model.add(Flatten())

model.add(Dense(1024, activation='relu'))

model.add(Dropout(0.5))

model.add(Dense(len(labels), activation='softmax'))

# Compile the model

optimizer = optimizers.Adam(learning\_rate=0.0001)

model.compile(loss='categorical\_crossentropy', optimizer=optimizer, metrics=['accuracy'])

early\_stopping = EarlyStopping(monitor='val\_loss', patience=3)

# Train the model

model.summary()

mobile\_history = model.fit(X\_train, Y\_train, validation\_data=(X\_test, Y\_test), epochs=20, batch\_size=32, verbose=1)

train\_loss = mobile\_history.history['loss']

train\_accuracy = mobile\_history.history['accuracy']

test\_loss = mobile\_history.history['val\_loss']

test\_accuracy = mobile\_history.history['val\_accuracy']

epochs = range(1, len(train\_loss)+1)

plt.plot(epochs, train\_accuracy, 'b', label='Training Accuracy')

plt.plot(epochs, test\_accuracy, 'r', label='Testing Accuracy')

plt.title('Training and Testing Accuracy')

plt.xlabel('Epochs')

plt.ylabel('Accuracy')

plt.ylim(0,1.2)

plt.legend()

plt.grid(True)

plt.show()

from sklearn.metrics import accuracy\_score

from sklearn.metrics import precision\_score

from sklearn.metrics import recall\_score

from sklearn.metrics import f1\_score

from sklearn.metrics import confusion\_matrix

from sklearn.metrics import classification\_report

y\_pred\_test = model.predict(X\_test)

# Convert predictions and true labels from one-hot encoding to class labels

y\_pred\_test\_labels = np.argmax(y\_pred\_test, axis=1)

y\_true\_test\_labels = np.argmax(Y\_test, axis=1)

# Compute validation accuracy

validation\_accuracy = accuracy\_score(y\_true\_test\_labels, y\_pred\_test\_labels)

# Compute precision, recall, and F1-score

validation\_precision = precision\_score(y\_true\_test\_labels, y\_pred\_test\_labels, average='weighted')

validation\_recall = recall\_score(y\_true\_test\_labels, y\_pred\_test\_labels, average='weighted')

validation\_f1 = f1\_score(y\_true\_test\_labels, y\_pred\_test\_labels, average='weighted')

# Print validation metrics

print(f"Validation Accuracy: {validation\_accuracy:.4f}")

print(f"Validation Precision: {validation\_precision:.4f}")

print(f"Validation Recall: {validation\_recall:.4f}")

print(f"Validation F1 Score: {validation\_f1:.4f}")

# Display classification report

print("\nClassification Report:")

print(classification\_report(y\_true\_test\_labels, y\_pred\_test\_labels, target\_names=labels))

y\_pred = model.predict(X\_train)

# Convert predictions from one-hot encoding to class labels

y\_pred\_labels = np.argmax(y\_pred, axis=1)

y\_true\_labels = np.argmax(Y\_train, axis=1)

# Create the confusion matrix

cm = confusion\_matrix(y\_true\_labels, y\_pred\_labels)

# Display the confusion matrix as a heatmap

plt.figure(figsize=(10, 8))

plt.imshow(cm, interpolation='nearest', cmap=plt.cm.Blues)

plt.title('Confusion Matrix')

plt.colorbar()

tick\_marks = np.arange(len(labels))

plt.xticks(tick\_marks, labels, rotation=45)

plt.yticks(tick\_marks, labels)

plt.xlabel('Predicted Label')

plt.ylabel('True Label')

# Add text annotations for each cell in the heatmap

thresh = cm.max() / 2.

for i in range(cm.shape[0]):

for j in range(cm.shape[1]):

plt.text(j, i, format(cm[i, j], 'd'), ha="center", va="center", color="white" if cm[i, j] > thresh else "black")

plt.tight\_layout()

plt.show()

**EfficientNet Code:**

import os

folder\_path = '/Users/sasankdachepalli/Capstone New Dataset/Updated'

subfolders = [f for f in os.listdir(folder\_path) if os.path.isdir(os.path.join(folder\_path, f))]

print(subfolders)

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

import os

import glob

import seaborn as sns

import cv2

import io

import tensorflow as tf

from sklearn.model\_selection import train\_test\_split

from tqdm import tqdm

import keras

from keras.preprocessing import image

#from keras.preprocessing.image import ImageDataGenerator

from PIL import Image

from sklearn.utils import shuffle

from keras import layers,models,optimizers

import ipywidgets as widgets

from keras.utils import to\_categorical

from keras.models import Sequential

from keras.layers import Dense, Activation, Dropout, Flatten, Conv2D, MaxPooling2D,Dropout,BatchNormalization,Concatenate,AveragePooling2D

from sklearn.metrics import classification\_report, confusion\_matrix

from keras.models import Sequential

from keras.regularizers import l2

from keras.callbacks import EarlyStopping

import glob

import cv2

X=[]

Y=[]

image\_size=(227,227)

for i in subfolders:

path="/Users/sasankdachepalli/Capstone New Dataset/Updated/"+i+"/"

print(path)

fileRead=glob.glob(path+"\*")

print(len(fileRead))

for j in fileRead:

image=cv2.imread(j)

image=cv2.resize(image,image\_size)

X.append(image)

Y.append(i)

X = np.array(X)

Y = np.array(Y)

labels=subfolders

import numpy as np

Temp\_y = []

for i in Y:

Temp\_y.append(labels.index(i))

Y = np.array(Temp\_y)

Y = to\_categorical(Y)

#EfficientNet

from tensorflow.keras.applications import EfficientNetB0 # Choose the desired EfficientNet variant (B0, B1, B2, etc.)

from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import Flatten, Dense, Dropout

from tensorflow.keras.optimizers import Adam

from tensorflow.keras.callbacks import EarlyStopping

# Load EfficientNet with pre-trained weights

efficientnet\_base = EfficientNetB0(weights='imagenet', include\_top=False, input\_shape=(227, 227, 3))

# Freeze pre-trained layers (optional)

efficientnet\_base.trainable = False

# Create the model

model = Sequential()

model.add(efficientnet\_base)

model.add(Flatten())

model.add(Dense(1024, activation='relu'))

model.add(Dropout(0.5))

model.add(Dense(len(labels), activation='softmax'))

model.summary()

# Compile the model

optimizer = optimizers.Adam(learning\_rate=0.0001)

model.compile(loss='categorical\_crossentropy', optimizer=optimizer, metrics=['accuracy'])

early\_stopping = EarlyStopping(monitor='val\_loss', patience=3)

#Train the model

efficientnet\_history = model.fit(X\_train, Y\_train, validation\_data=(X\_test, Y\_test), epochs=20, batch\_size=32, verbose=1, callbacks=[early\_stopping])

train\_loss = efficientnet\_history.history['loss']

train\_accuracy = efficientnet\_history.history['accuracy']

test\_loss = efficientnet\_history.history['val\_loss']

test\_accuracy = efficientnet\_history.history['val\_accuracy']

epochs = range(1, len(train\_loss)+1)

plt.plot(epochs, train\_accuracy, 'b', label='Training Accuracy')

plt.plot(epochs, test\_accuracy, 'r', label='Testing Accuracy')

plt.title('Training and Testing Accuracy')

plt.xlabel('Epochs')

plt.ylabel('Accuracy')

plt.ylim(0,1.2)

plt.legend()

plt.grid(True)

plt.show()

from sklearn.metrics import accuracy\_score

from sklearn.metrics import precision\_score

from sklearn.metrics import recall\_score

from sklearn.metrics import f1\_score

from sklearn.metrics import confusion\_matrix

from sklearn.metrics import classification\_report

y\_pred\_test = model.predict(X\_test)

# Convert predictions and true labels from one-hot encoding to class labels

y\_pred\_test\_labels = np.argmax(y\_pred\_test, axis=1)

y\_true\_test\_labels = np.argmax(Y\_test, axis=1)

# Compute validation accuracy

validation\_accuracy = accuracy\_score(y\_true\_test\_labels, y\_pred\_test\_labels)

# Compute precision, recall, and F1-score

validation\_precision = precision\_score(y\_true\_test\_labels, y\_pred\_test\_labels, average='weighted')

validation\_recall = recall\_score(y\_true\_test\_labels, y\_pred\_test\_labels, average='weighted')

validation\_f1 = f1\_score(y\_true\_test\_labels, y\_pred\_test\_labels, average='weighted')

# Print validation metrics

print(f"Validation Accuracy: {validation\_accuracy:.4f}")

print(f"Validation Precision: {validation\_precision:.4f}")

print(f"Validation Recall: {validation\_recall:.4f}")

print(f"Validation F1 Score: {validation\_f1:.4f}")

# Display classification report

print("\nClassification Report:")

print(classification\_report(y\_true\_test\_labels, y\_pred\_test\_labels, target\_names=labels))

y\_pred = model.predict(X\_train)

# Convert predictions from one-hot encoding to class labels

y\_pred\_labels = np.argmax(y\_pred, axis=1)

y\_true\_labels = np.argmax(Y\_train, axis=1)

# Create the confusion matrix

cm = confusion\_matrix(y\_true\_labels, y\_pred\_labels)

# Display the confusion matrix as a heatmap

plt.figure(figsize=(10, 8))

plt.imshow(cm, interpolation='nearest', cmap=plt.cm.Blues)

plt.title('Confusion Matrix')

plt.colorbar()

tick\_marks = np.arange(len(labels))

plt.xticks(tick\_marks, labels, rotation=45)

plt.yticks(tick\_marks, labels)

plt.xlabel('Predicted Label')

plt.ylabel('True Label')

# Add text annotations for each cell in the heatmap

thresh = cm.max() / 2.

for i in range(cm.shape[0]):

for j in range(cm.shape[1]):

plt.text(j, i, format(cm[i, j], 'd'), ha="center", va="center", color="white" if cm[i, j] > thresh else "black")

plt.tight\_layout()

plt.show()

**DenseNet Code:**

import os

folder\_path = '/content/UpdatedDataset/Updated'

subfolders = [f for f in os.listdir(folder\_path) if os.path.isdir(os.path.join(folder\_path, f))]

print(subfolders)

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

import os

import glob

import seaborn as sns

import cv2

import io

import tensorflow as tf

from sklearn.model\_selection import train\_test\_split

from tqdm import tqdm

import keras

from keras.preprocessing import image

#from keras.preprocessing.image import ImageDataGenerator

from PIL import Image

from sklearn.utils import shuffle

from keras import layers,models,optimizers

import ipywidgets as widgets

from keras.utils import to\_categorical

from keras.models import Sequential

from keras.layers import Dense, Activation, Dropout, Flatten, Conv2D, MaxPooling2D,Dropout,BatchNormalization,Concatenate,AveragePooling2D

from sklearn.metrics import classification\_report, confusion\_matrix

from keras.models import Sequential

from keras.regularizers import l2

from keras.callbacks import EarlyStopping

import glob

import cv2

X=[]

Y=[]

image\_size=(227,227)

for i in subfolders:

path="/content/UpdatedDataset/Updated/"+i+"/"

print(path)

fileRead=glob.glob(path+"\*")

print(len(fileRead))

for j in fileRead:

image=cv2.imread(j)

image=cv2.resize(image,image\_size)

X.append(image)

Y.append(i)

X = np.array(X)

Y = np.array(Y)

labels=subfolders

import numpy as np

Temp\_y = []

for i in Y:

Temp\_y.append(labels.index(i))

Y = np.array(Temp\_y)

Y = to\_categorical(Y)

X\_train,X\_test,Y\_train,Y\_test=train\_test\_split(X,Y,test\_size=0.2,random\_state=42)

from tensorflow.keras.applications import DenseNet121

# Load DenseNet121 with pre-trained weights

densenet\_base = DenseNet121(weights='imagenet', include\_top=False, input\_shape=(227, 227, 3))

# Freeze pre-trained layers (optional)

densenet\_base.trainable = False

# Add custom layers for classification

model = Sequential()

model.add(densenet\_base)

model.add(Flatten())

model.add(Dense(1024, activation='relu'))

model.add(Dropout(0.5))

model.add(Dense(len(labels), activation='softmax'))

# Compile the model

optimizer = optimizers.Adam(learning\_rate=0.0001)

model.compile(loss='categorical\_crossentropy', optimizer=optimizer, metrics=['accuracy'])

early\_stopping = EarlyStopping(monitor='val\_loss', patience=3)

model.summary()

# Train the model

densenet\_history = model.fit(X\_train, Y\_train, validation\_data=(X\_test, Y\_test), epochs=20, batch\_size=32, verbose=1)

train\_loss = densenet\_history.history['loss']

train\_accuracy = densenet\_history.history['accuracy']

test\_loss = densenet\_history.history['val\_loss']

test\_accuracy = densenet\_history.history['val\_accuracy']

epochs = range(1, len(train\_loss)+1)

plt.plot(epochs, train\_accuracy, 'b', label='Training Accuracy')

plt.plot(epochs, test\_accuracy, 'r', label='Testing Accuracy')

plt.title('Training and Testing Accuracy')

plt.xlabel('Epochs')

plt.ylabel('Accuracy')

plt.ylim(0,1.2)

plt.legend()

plt.grid(True)

plt.show()

from sklearn.metrics import accuracy\_score

from sklearn.metrics import precision\_score

from sklearn.metrics import recall\_score

from sklearn.metrics import f1\_score

from sklearn.metrics import confusion\_matrix

from sklearn.metrics import classification\_report

y\_pred\_test = model.predict(X\_test)

# Convert predictions and true labels from one-hot encoding to class labels

y\_pred\_test\_labels = np.argmax(y\_pred\_test, axis=1)

y\_true\_test\_labels = np.argmax(Y\_test, axis=1)

# Compute validation accuracy

validation\_accuracy = accuracy\_score(y\_true\_test\_labels, y\_pred\_test\_labels)

# Compute precision, recall, and F1-score

validation\_precision = precision\_score(y\_true\_test\_labels, y\_pred\_test\_labels, average='weighted')

validation\_recall = recall\_score(y\_true\_test\_labels, y\_pred\_test\_labels, average='weighted')

validation\_f1 = f1\_score(y\_true\_test\_labels, y\_pred\_test\_labels, average='weighted')

# Print validation metrics

print(f"Validation Accuracy: {validation\_accuracy:.4f}")

print(f"Validation Precision: {validation\_precision:.4f}")

print(f"Validation Recall: {validation\_recall:.4f}")

print(f"Validation F1 Score: {validation\_f1:.4f}")

# Display classification report

print("\nClassification Report:")

print(classification\_report(y\_true\_test\_labels, y\_pred\_test\_labels, target\_names=labels))

y\_pred = model.predict(X\_train)

# Convert predictions from one-hot encoding to class labels

y\_pred\_labels = np.argmax(y\_pred, axis=1)

y\_true\_labels = np.argmax(Y\_train, axis=1)

# Create the confusion matrix

cm = confusion\_matrix(y\_true\_labels, y\_pred\_labels)

# Display the confusion matrix as a heatmap

plt.figure(figsize=(10, 8))

plt.imshow(cm, interpolation='nearest', cmap=plt.cm.Blues)

plt.title('Confusion Matrix')

plt.colorbar()

tick\_marks = np.arange(len(labels))

plt.xticks(tick\_marks, labels, rotation=45)

plt.yticks(tick\_marks, labels)

plt.xlabel('Predicted Label')

plt.ylabel('True Label')

# Add text annotations for each cell in the heatmap

thresh = cm.max() / 2.

for i in range(cm.shape[0]):

for j in range(cm.shape[1]):

plt.text(j, i, format(cm[i, j], 'd'), ha="center", va="center", color="white" if cm[i, j] > thresh else "black")

plt.tight\_layout()

plt.show()

**AlexNet Code:**

import os

folder\_path = '/content/dataset\_folder/Updated'

subfolders = [f for f in os.listdir(folder\_path) if os.path.isdir(os.path.join(folder\_path, f))]

print(subfolders)

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

import os

import glob

import seaborn as sns

import cv2

import io

import tensorflow as tf

from sklearn.model\_selection import train\_test\_split

from tqdm import tqdm

import keras

from keras.preprocessing import image

#from keras.preprocessing.image import ImageDataGenerator

from PIL import Image

from sklearn.utils import shuffle

from keras import layers,models,optimizers

import ipywidgets as widgets

from keras.utils import to\_categorical

from keras.models import Sequential

from keras.layers import Dense, Activation, Dropout, Flatten, Conv2D, MaxPooling2D,Dropout,BatchNormalization,Concatenate,AveragePooling2D

from sklearn.metrics import classification\_report, confusion\_matrix

from keras.models import Sequential

from keras.regularizers import l2

from keras.callbacks import EarlyStopping

import glob

import cv2

X=[]

Y=[]

image\_size=(227,227)

for i in subfolders:

path="/content/dataset\_folder/Updated/"+i+"/"

print(path)

fileRead=glob.glob(path+"\*")

print(len(fileRead))

for j in fileRead:

image=cv2.imread(j)

image=cv2.resize(image,image\_size)

X.append(image)

Y.append(i)

X = np.array(X)

Y = np.array(Y)

labels=subfolders

labels

import numpy as np

Temp\_y = []

for i in Y:

Temp\_y.append(labels.index(i))

Y = np.array(Temp\_y)

Y = to\_categorical(Y)

AlexNet = Sequential()

#1st Conv2D Layer

AlexNet.add(Conv2D(96, kernel\_size = (11, 11), strides = (4, 4),

padding = "valid", activation = 'relu', input\_shape = (227, 227, 3)))

AlexNet.add(MaxPooling2D(pool\_size = (3, 3),

strides = (2, 2), padding = "valid",

data\_format = None))

#2nd Conv2D Layer

AlexNet.add(Conv2D(256, kernel\_size = (5, 5), strides = 1,

padding = "same", activation = 'relu'))

AlexNet.add(MaxPooling2D(pool\_size = (3, 3),

strides = (2, 2), padding = "valid",#"same"

data\_format = None))

#3rd Conv2D Layer

AlexNet.add(Conv2D(384, kernel\_size = (3, 3), strides = 1,

padding = "same", activation = 'relu'))

#4th Conv2D Layer

AlexNet.add(Conv2D(384, kernel\_size = (3, 3), strides = 1,

padding = "same", activation = 'relu'))

#5th Conv2D Layer

AlexNet.add(Conv2D(256, kernel\_size = (3, 3), strides = 1,

padding = "same", activation = 'relu'))

AlexNet.add(MaxPooling2D(pool\_size = (3, 3),

strides = (2, 2), padding = "valid",#"same"

data\_format = None))

# Flatten Layer

AlexNet.add(Flatten())

AlexNet.add(Dense(4096, activation = 'relu'))

AlexNet.add(Dense(4096, activation = 'relu'))

#AlexNet.add(Dense(1000, activation = 'relu'))

AlexNet.add(Dense(14, activation = 'softmax'))

optimizer = optimizers.Adam(learning\_rate=0.0001)

AlexNet.summary()

AlexNet.compile(loss='categorical\_crossentropy', optimizer=optimizer, metrics=['accuracy'])

early\_stopping = EarlyStopping(monitor='val\_loss', patience=3)

alex\_history=AlexNet.fit(X\_train,Y\_train, validation\_data=(X\_test,Y\_test),epochs=20,batch\_size=32,verbose=1)

train\_loss = alex\_history.history['loss']

train\_accuracy = alex\_history.history['accuracy']

test\_loss = alex\_history.history['val\_loss']

test\_accuracy = alex\_history.history['val\_accuracy']

epochs = range(1, len(train\_loss)+1)

plt.figure(figsize=(8, 4))

plt.plot(epochs, train\_loss, 'b', label='Training Loss')

plt.plot(epochs, test\_loss, 'r', label='Testing Loss')

plt.title('Training and Testing Loss')

plt.xlabel('Epochs')

plt.ylabel('Loss')

plt.ylim(0,35)

plt.legend()

plt.grid(True)

plt.show()

plt.plot(epochs, train\_accuracy, 'b', label='Training Accuracy')

plt.plot(epochs, test\_accuracy, 'r', label='Testing Accuracy')

plt.title('Training and Testing Accuracy')

plt.xlabel('Epochs')

plt.ylabel('Accuracy')

plt.ylim(0,1.2)

plt.legend()

plt.grid(True)

plt.show()

y\_pred = AlexNet.predict(X\_train)

# Convert predictions from one-hot encoding to class labels

y\_pred\_labels = np.argmax(y\_pred, axis=1)

y\_true\_labels = np.argmax(Y\_train, axis=1)

# Create the confusion matrix

cm = confusion\_matrix(y\_true\_labels, y\_pred\_labels)

# Display the confusion matrix as a heatmap

plt.figure(figsize=(10, 8))

plt.imshow(cm, interpolation='nearest', cmap=plt.cm.Blues)

plt.title('Confusion Matrix')

plt.colorbar()

tick\_marks = np.arange(len(labels))

plt.xticks(tick\_marks, labels, rotation=45)

plt.yticks(tick\_marks, labels)

plt.xlabel('Predicted Label')

plt.ylabel('True Label')

# Add text annotations for each cell in the heatmap

thresh = cm.max() / 2.

for i in range(cm.shape[0]):

for j in range(cm.shape[1]):

plt.text(j, i, format(cm[i, j], 'd'), ha="center", va="center", color="white" if cm[i, j] > thresh else "black")

plt.tight\_layout()

plt.show()

from sklearn.metrics import accuracy\_score

from sklearn.metrics import precision\_score

from sklearn.metrics import recall\_score

from sklearn.metrics import f1\_score

from sklearn.metrics import confusion\_matrix

from sklearn.metrics import classification\_report

y\_pred\_test = AlexNet.predict(X\_test)

# Convert predictions and true labels from one-hot encoding to class labels

y\_pred\_test\_labels = np.argmax(y\_pred\_test, axis=1)

y\_true\_test\_labels = np.argmax(Y\_test, axis=1)

# Compute validation accuracy

validation\_accuracy = accuracy\_score(y\_true\_test\_labels, y\_pred\_test\_labels)

# Compute precision, recall, and F1-score

validation\_precision = precision\_score(y\_true\_test\_labels, y\_pred\_test\_labels, average='weighted')

validation\_recall = recall\_score(y\_true\_test\_labels, y\_pred\_test\_labels, average='weighted')

validation\_f1 = f1\_score(y\_true\_test\_labels, y\_pred\_test\_labels, average='weighted')

# Print validation metrics

print(f"Validation Accuracy: {validation\_accuracy:.4f}")

print(f"Validation Precision: {validation\_precision:.4f}")

print(f"Validation Recall: {validation\_recall:.4f}")

print(f"Validation F1 Score: {validation\_f1:.4f}")

# Display classification report

print("\nClassification Report:")

print(classification\_report(y\_true\_test\_labels, y\_pred\_test\_labels, target\_names=labels))

**ResNet Code:**

import os

folder\_path = '/content/dataset/Updated'

subfolders = [f for f in os.listdir(folder\_path) if os.path.isdir(os.path.join(folder\_path, f))]

print(subfolders)

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

import os

import glob

import seaborn as sns

import cv2

import io

import tensorflow as tf

from sklearn.model\_selection import train\_test\_split

from tqdm import tqdm

import keras

from keras.preprocessing import image

#from keras.preprocessing.image import ImageDataGenerator

from PIL import Image

from sklearn.utils import shuffle

from keras import layers,models,optimizers

import ipywidgets as widgets

from keras.utils import to\_categorical

from keras.models import Sequential

from keras.layers import Dense, Activation, Dropout, Flatten, Conv2D, MaxPooling2D,Dropout,BatchNormalization,Concatenate,AveragePooling2D

from sklearn.metrics import classification\_report, confusion\_matrix

from keras.models import Sequential

from keras.regularizers import l2

from keras.callbacks import EarlyStopping

import glob

import cv2

X=[]

Y=[]

image\_size=(227,227)

for i in subfolders:

path="/content/dataset/Updated/"+i+"/"

print(path)

fileRead=glob.glob(path+"\*")

print(len(fileRead))

for j in fileRead:

image=cv2.imread(j)

image=cv2.resize(image,image\_size)

X.append(image)

Y.append(i)

X = np.array(X)

Y = np.array(Y)

labels=subfolders

import numpy as np

Temp\_y = []

for i in Y:

Temp\_y.append(labels.index(i))

Y = np.array(Temp\_y)

Y = to\_categorical(Y)

X\_train,X\_test,Y\_train,Y\_test=train\_test\_split(X,Y,test\_size=0.2,random\_state=42)

from tensorflow.keras.applications import ResNet50

# Load ResNet50 with pre-trained weights

resnet\_base = ResNet50(weights='imagenet', include\_top=False, input\_shape=(227, 227, 3))

# Freeze pre-trained layers (optional)

resnet\_base.trainable = False

# Add custom layers for classification

model = Sequential()

model.add(resnet\_base)

model.add(Flatten())

model.add(Dense(1024, activation='relu'))

model.add(Dropout(0.5))

model.add(Dense(len(labels), activation='softmax'))

# Compile the model

optimizer = optimizers.Adam(learning\_rate=0.0001)

model.compile(loss='categorical\_crossentropy', optimizer=optimizer, metrics=['accuracy'])

early\_stopping = EarlyStopping(monitor='val\_loss', patience=3)

model.summary()

# Train the model

resnet\_history = model.fit(X\_train, Y\_train, validation\_data=(X\_test, Y\_test), epochs=20, batch\_size=32, verbose=1)

train\_loss = resnet\_history.history['loss']

train\_accuracy = resnet\_history.history['accuracy']

test\_loss = resnet\_history.history['val\_loss']

test\_accuracy = resnet\_history.history['val\_accuracy']

epochs = range(1, len(train\_loss)+1)

plt.plot(epochs, train\_accuracy, 'b', label='Training Accuracy')

plt.plot(epochs, test\_accuracy, 'r', label='Testing Accuracy')

plt.title('Training and Testing Accuracy')

plt.xlabel('Epochs')

plt.ylabel('Accuracy')

plt.ylim(0,1.2)

plt.legend()

plt.grid(True)

plt.show()

from sklearn.metrics import accuracy\_score

from sklearn.metrics import precision\_score

from sklearn.metrics import recall\_score

from sklearn.metrics import f1\_score

from sklearn.metrics import confusion\_matrix

from sklearn.metrics import classification\_report

y\_pred\_test = model.predict(X\_test)

# Convert predictions and true labels from one-hot encoding to class labels

y\_pred\_test\_labels = np.argmax(y\_pred\_test, axis=1)

y\_true\_test\_labels = np.argmax(Y\_test, axis=1)

# Compute validation accuracy

validation\_accuracy = accuracy\_score(y\_true\_test\_labels, y\_pred\_test\_labels)

# Compute precision, recall, and F1-score

validation\_precision = precision\_score(y\_true\_test\_labels, y\_pred\_test\_labels, average='weighted')

validation\_recall = recall\_score(y\_true\_test\_labels, y\_pred\_test\_labels, average='weighted')

validation\_f1 = f1\_score(y\_true\_test\_labels, y\_pred\_test\_labels, average='weighted')

# Print validation metrics

print(f"Validation Accuracy: {validation\_accuracy:.4f}")

print(f"Validation Precision: {validation\_precision:.4f}")

print(f"Validation Recall: {validation\_recall:.4f}")

print(f"Validation F1 Score: {validation\_f1:.4f}")

# Display classification report

print("\nClassification Report:")

print(classification\_report(y\_true\_test\_labels, y\_pred\_test\_labels, target\_names=labels))

y\_pred = model.predict(X\_train)

# Convert predictions from one-hot encoding to class labels

y\_pred\_labels = np.argmax(y\_pred, axis=1)

y\_true\_labels = np.argmax(Y\_train, axis=1)

# Create the confusion matrix

cm = confusion\_matrix(y\_true\_labels, y\_pred\_labels)

# Display the confusion matrix as a heatmap

plt.figure(figsize=(10, 8))

plt.imshow(cm, interpolation='nearest', cmap=plt.cm.Blues)

plt.title('Confusion Matrix')

plt.colorbar()

tick\_marks = np.arange(len(labels))

plt.xticks(tick\_marks, labels, rotation=45)

plt.yticks(tick\_marks, labels)

plt.xlabel('Predicted Label')

plt.ylabel('True Label')

# Add text annotations for each cell in the heatmap

thresh = cm.max() / 2.

for i in range(cm.shape[0]):

for j in range(cm.shape[1]):

plt.text(j, i, format(cm[i, j], 'd'), ha="center", va="center", color="white" if cm[i, j] > thresh else "black")

plt.tight\_layout()

plt.show()

**Xception Code:**

import os

folder\_path = '/Users/sasankdachepalli/Capstone New Dataset/Updated'

subfolders = [f for f in os.listdir(folder\_path) if os.path.isdir(os.path.join(folder\_path, f))]

print(subfolders)

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

import os

import glob

import seaborn as sns

import cv2

import io

import tensorflow as tf

from sklearn.model\_selection import train\_test\_split

from tqdm import tqdm

import keras

from keras.preprocessing import image

#from keras.preprocessing.image import ImageDataGenerator

from PIL import Image

from sklearn.utils import shuffle

from keras import layers,models,optimizers

import ipywidgets as widgets

from keras.utils import to\_categorical

from keras.models import Sequential

from keras.layers import Dense, Activation, Dropout, Flatten, Conv2D, MaxPooling2D,Dropout,BatchNormalization,Concatenate,AveragePooling2D

from sklearn.metrics import classification\_report, confusion\_matrix

from keras.models import Sequential

from keras.regularizers import l2

from keras.callbacks import EarlyStopping

import glob

import cv2

X=[]

Y=[]

image\_size=(227,227)

for i in subfolders:

path="/Users/sasankdachepalli/Capstone New Dataset/Updated/"+i+"/"

print(path)

fileRead=glob.glob(path+"\*")

print(len(fileRead))

for j in fileRead:

image=cv2.imread(j)

image=cv2.resize(image,image\_size)

X.append(image)

Y.append(i)

X = np.array(X)

Y = np.array(Y)

labels=subfolders

import numpy as np

Temp\_y = []

for i in Y:

Temp\_y.append(labels.index(i))

Y = np.array(Temp\_y)

Y = to\_categorical(Y)

X\_train,X\_test,Y\_train,Y\_test=train\_test\_split(X,Y,test\_size=0.2,random\_state=42)

#Xception

from tensorflow.keras.applications import Xception

from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import Flatten, Dense, Dropout

from tensorflow.keras.optimizers import Adam

from tensorflow.keras.callbacks import EarlyStopping

# Load Xception with pre-trained weights

xception\_base = Xception(weights='imagenet', include\_top=False, input\_shape=(227, 227, 3))

# Freeze pre-trained layers (optional)

xception\_base.trainable = False

# Create the model

model = Sequential()

model.add(xception\_base)

model.add(Flatten())

model.add(Dense(1024, activation='relu'))

model.add(Dropout(0.5))

model.add(Dense(len(labels), activation='softmax'))

# Compile the model

optimizer = Adam(learning\_rate=0.0001)

model.compile(loss='categorical\_crossentropy', optimizer=optimizer, metrics=['accuracy'])

early\_stopping = EarlyStopping(monitor='val\_loss', patience=3)

# Train the model

model.summary()

# Compile the model

optimizer = Adam(learning\_rate=0.0001)

model.compile(loss='categorical\_crossentropy', optimizer=optimizer, metrics=['accuracy'])

early\_stopping = EarlyStopping(monitor='val\_loss', patience=3)

model.summary()

# Train the model

xception\_history = model.fit(X\_train, Y\_train, validation\_data=(X\_test, Y\_test), epochs=20, batch\_size=32, verbose=1, callbacks=[early\_stopping])

train\_loss = xception\_history.history['loss']

train\_accuracy = xception\_history.history['accuracy']

test\_loss = xception\_history.history['val\_loss']

test\_accuracy = xception\_history.history['val\_accuracy']

epochs = range(1, len(train\_loss)+1)

plt.plot(epochs, train\_accuracy, 'b', label='Training Accuracy')

plt.plot(epochs, test\_accuracy, 'r', label='Testing Accuracy')

plt.title('Training and Testing Accuracy')

plt.xlabel('Epochs')

plt.ylabel('Accuracy')

plt.ylim(0,1.2)

plt.legend()

plt.grid(True)

plt.show()

from sklearn.metrics import accuracy\_score

from sklearn.metrics import precision\_score

from sklearn.metrics import recall\_score

from sklearn.metrics import f1\_score

from sklearn.metrics import confusion\_matrix

from sklearn.metrics import classification\_report

y\_pred\_test = model.predict(X\_test)

# Convert predictions and true labels from one-hot encoding to class labels

y\_pred\_test\_labels = np.argmax(y\_pred\_test, axis=1)

y\_true\_test\_labels = np.argmax(Y\_test, axis=1)

# Compute validation accuracy

validation\_accuracy = accuracy\_score(y\_true\_test\_labels, y\_pred\_test\_labels)

# Compute precision, recall, and F1-score

validation\_precision = precision\_score(y\_true\_test\_labels, y\_pred\_test\_labels, average='weighted')

validation\_recall = recall\_score(y\_true\_test\_labels, y\_pred\_test\_labels, average='weighted')

validation\_f1 = f1\_score(y\_true\_test\_labels, y\_pred\_test\_labels, average='weighted')

# Print validation metrics

print(f"Validation Accuracy: {validation\_accuracy:.4f}")

print(f"Validation Precision: {validation\_precision:.4f}")

print(f"Validation Recall: {validation\_recall:.4f}")

print(f"Validation F1 Score: {validation\_f1:.4f}")

# Display classification report

print("\nClassification Report:")

print(classification\_report(y\_true\_test\_labels, y\_pred\_test\_labels, target\_names=labels))

y\_pred = model.predict(X\_train)

# Convert predictions from one-hot encoding to class labels

y\_pred\_labels = np.argmax(y\_pred, axis=1)

y\_true\_labels = np.argmax(Y\_train, axis=1)

# Create the confusion matrix

cm = confusion\_matrix(y\_true\_labels, y\_pred\_labels)

# Display the confusion matrix as a heatmap

plt.figure(figsize=(10, 8))

plt.imshow(cm, interpolation='nearest', cmap=plt.cm.Blues)

plt.title('Confusion Matrix')

plt.colorbar()

tick\_marks = np.arange(len(labels))

plt.xticks(tick\_marks, labels, rotation=45)

plt.yticks(tick\_marks, labels)

plt.xlabel('Predicted Label')

plt.ylabel('True Label')

# Add text annotations for each cell in the heatmap

thresh = cm.max() / 2.

for i in range(cm.shape[0]):

for j in range(cm.shape[1]):

plt.text(j, i, format(cm[i, j], 'd'), ha="center", va="center", color="white" if cm[i, j] > thresh else "black")

plt.tight\_layout()

plt.show()

**CHAPTER 6**

**REFERENCES**

1. Tm, P., Pranathi, A., SaiAshritha, K., Chittaragi, N. B., & Koolagudi, S. G. (2018, August). Tomato leaf disease detection using convolutional neural networks. In *2018 eleventh international conference on contemporary computing (IC3)* (pp. 1-5). IEEE.
2. Dube, S. S., Chiwera, E., Nleya, S. M., Ndlovu, S., Zengeya, T., Marabada, N., ... & Mapenduka, W. (2023, November). Tomato Leaf Disease Detection System Using Convolutional Neural Networks. In *2023 2nd Zimbabwe Conference of Information and Communication Technologies (ZCICT)* (pp. 1-4). IEEE.
3. Gargade, A., & Khandekar, S. A. (2019, March). A review: custard apple leaf parameter analysis and leaf disease detection using digital image processing. In *2019 3rd International Conference on Computing Methodologies and Communication (ICCMC)* (pp. 267-271). IEEE.
4. B. U. Rani, K. Pavani, S. Bhavani and G. Alapati, "A Systematic Analysis of Deep Learning and Machine Learning Methods for Identifying Apple Leaf Disease," 2023 4th International Conference on Electronics and Sustainable Communication Systems (ICESC), Coimbatore, India, 2023, pp. 761-765, doi: 10.1109/ICESC57686.2023.10192948.
5. Ahil, M. N., Vanitha, V., & Rajathi, N. (2021, October). Apple and grape leaf disease classification using MLP and CNN. In *2021 International Conference on Advancements in Electrical, Electronics, Communication, Computing and Automation (ICAECA)* (pp. 1-4). IEEE.
6. Sebastian, M., Suchithra, M. S., & Antony, C. M. (2023, December). Apple Leaf Disease Detection: Machine Learning & Deep Learning Techniques. In *2023 Intelligent Computing and Control for Engineering and Business Systems (ICCEBS)* (pp. 1-5). IEEE.
7. Yang, J. (2024, September). Deep Learning for Tomato Leaf Disease Recognition in Balanced Data Environment. In *2024 4th International Conference on Computer Science and Blockchain (CCSB)* (pp. 74-78). IEEE.
8. Brucal, S. G. E., de Jesus, L. C. M., Peruda, S. R., Samaniego, L. A., & Yong, E. D. (2023, October). Development of Tomato Leaf Disease Detection using YoloV8 Model via RoboFlow 2.0. In *2023 IEEE 12th Global Conference on Consumer Electronics (GCCE)* (pp. 692-694). IEEE.
9. Kaur, A., Kukreja, V., Kumar, M., Choudhary, A., & Sharma, R. (2024, April). Multiclass Classification of Tomato Leaf Diseases using a Hybrid of LSTM and CNN Model. In *2024 IEEE 9th International Conference for Convergence in Technology (I2CT)* (pp. 1-5). IEEE.
10. Chakraborty, S., Paul, S., & Rahat-uz-Zaman, M. (2021, January). Prediction of apple leaf diseases using multiclass support vector machine. In *2021 2Nd international conference on robotics, electrical and signal processing techniques (ICREST)* (pp. 147-151). IEEE.

**BIODATA**

|  |  |
| --- | --- |
| Profile Pic | Personal Information |
|  | Name: Dachepalli Sasank  Mobile Number: 8125934715  E-mail: sasank.21bce9133@vitapstudent.ac.in  Permanent Address: 6-3-25, Allaka Street, near Pen School, Vizanagaram, 535001 |
|  | Name: Ryali Purna Sri Sai Sumanth  Mobile Number: 7660901949  E-mail: sumanth.21bce9227@vitapstudent.ac.in  Permanent Address: 5-52, ramalayam street, chikkala, 534305 |
|  | Name: chennupati venkata Lakshman  Mobile Number: 8886951645  E-mail: chennupati2504@gmail.com  Permanent Address: 001, main bazaar,kolalapudi village,martur mandal,bapatla district,andhra Pradesh – 523261 |
|  | Name: Venkata Ram Chennupati  Mobile Number: 8886788098  E-mail: ramchennupati2504@gmail.com  Permanent Address: 001, main bazaar,kolalapudi village,martur mandal,bapatla district,andhra Pradesh - 523261 |