RICE LEAF DISEASE DETECTION SYSTEM USING CNN

A PROJECT REPORT

Submitted by

DHARSHANA S	**422620104012**
PRAVEEN V	**422620104030**
VANITHA V	**422620104047**
VIJAYANANTH S	**422620104049**

in partial fulfillment for the award of the degree

of

BACHELOR OF ENGINEERING

in

COMPUTER SCIENCE AND ENGINEERING

UNIVERSITY COLLEGE OF ENGINEERING, PANRUTI

ANNA UNIVERSITY: CHENNAI 600 025

MAY 2024

ANNA UNIVERSITY: CHENNAI 600 025

BONAFIDE CERTIFICATE

Certified that this project report **“RICE LEAF DISEASE DETECTION SYSTEM USING CNN”** is the Bonafide work of **“DHARSHANA S (422620104012), PRAVEEN V (422620104030), VANITHA V (422620104047), VIJAYANANTH S (422620104049)”** who carried out the project work under my supervision.

SIGNATURE	**SIGNATURE**
Dr.D.Muruganantham.M.Tech.,Ph.D	Dr.A.Ramachandran ME.,MBA.,Ph.,D
HEAD OF DEPARTMENT	**SUPERVISOR**
Computer Science and Engineering, Panruti	Computer Science and Engineering, Panruti
University College of Engineering, Panruti	University College of Engineering, Panruti
Panruti-607106	Panruti-607106

Examination held on …………………………

INTERNAL EXAMINER **EXTERNAL EXAMINER**

ACKNOWLEDGEMENT

We would like to thank our dean Dr.S. Muthukumaran, M.E., Ph.D., for providing infrastructure facilities and hole hearted encouragement for completing our project successfully.

For mostly, we pay our grateful acknowledgement and extend our sincere gratitude to Dr.D.Muruganantham.M.Tech., Ph.D., Head of the Department Computer Science and Engineering, University College of Engineering panruti, Anna University, Chennai, for extending the facilities of the department towards our project and for his unstinting support.

We take this opportunity to also thank all our lectures who have directly or indirectly helped our project. we pay our respect and love to our parents and all other family members and friends for their love and encouragement throughout our career.

ABSTRACT

Rice is one of the most important staple crops in the world and a major source of food for millions of people. However, rice plants are susceptible to various diseases that can cause significant losses in yield and quality. Among these diseases, leaf blast, bacterial blight, and brown spot are the major attacking diseases that can cause devastating damage to rice crops. Effective disease management strategies are crucial for controlling these diseases and reducing their impact on rice production. To address this issue, the development of an efficient and accurate automated disease detection system is crucial. In recent years, the use of machine learning algorithms for plant disease diagnosis and classification has gained significant attention. With the availability of large datasets of plant images and the advancements in deep learning algorithms, it has become possible to accurately classify plant diseases based on their visual symptoms. This has the potential to improve disease management strategies and reduce the impact of plant diseases on crop production. In this project, we aim to develop a machine learning model that can accurately classify the three major attacking diseases of rice plants based on leaf images. By using advanced machine learning techniques, our model will be able to provide fast and accurate detection of diseases, enabling farmers to take prompt and effective measures to control the spread of the diseases and minimize crop losses. This project has the potential to revolutionize rice crop management and contribute to the sustainable development of agriculture.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF ABBREVATIONS

CNN	-	CONVOLUTIONAL NEURAL NETWORKS
VGG	-	VISUAL GEOMETRY GROUP AT THE UNIVERSITY OF OXFORD
ResNET	-	RESIDUAL NETWORK

CHAPTER-1

INTRODUCTION

1.1 DEEP LEARNING

The term Deep Learning, nowadays often an overused buzzword, is a subfield of machine learning. Deep learning focuses on training algorithms known as neural networks to learn and make predictions from data. Unlike traditional machine learning algorithms that require manual feature extraction, deep learning models can automatically learn hierarchical representations of data. This ability makes deep learning particularly effective for tasks such as image and speech recognition, natural language processing, and other complex pattern recognition tasks. The key characteristic of Deep Learning is the use of deep neural networks, which have multiple layers of interconnected nodes. These networks can learn features in the data. Deep Learning algorithms can automatically learn and improve from data without the need for manual feature engineering.

1.2 CONVOLUTIONAL NEURAL NETWORKS

A Convolutional Neural Network (CNN) also known as ConvNet, is the extended version of artificial neural network (ANN) which is predominantly used to extract the feature from the grid-like matrix dataset. And it is a specialized type of deep learning algorithm mainly designed for tasks that necessitate object recognition, including image classification, detection, and segmentation. It used to recognize the pattern from structured arrays.

1.2.1 CNN ARCHITECTURE

Convolutional Neural Network consists of multiple layers like the input layer, Convolutional layer, Pooling layer, and fully connected layers. The Convolutional layer applies filters to the input image to extract features, the Pooling layer down samples the image to reduce computation, and the fully connected layer makes the final prediction.

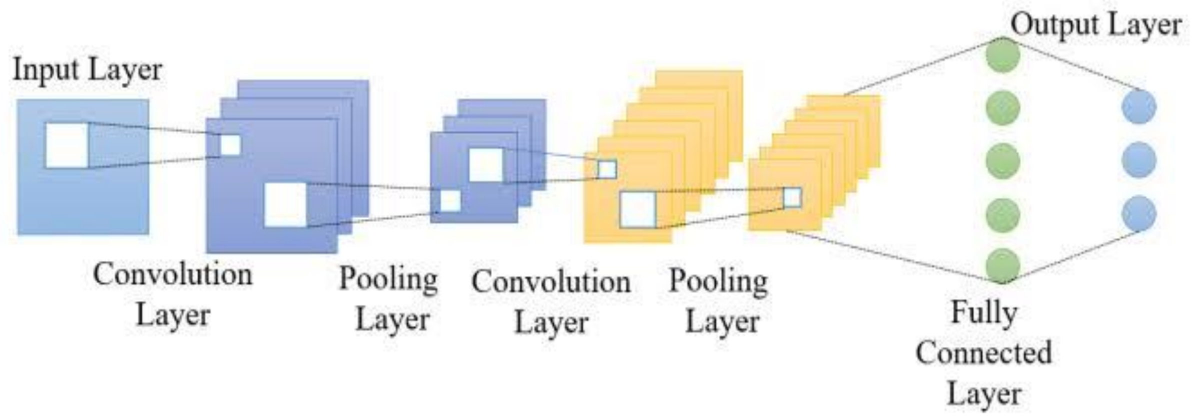


FIGURE 1.2.1 CNN ARCHITECTURE

1.2.2 TRANSFER LEARNING

Transfer learning is a technique in deep learning where a pre-trained neural network model is used as a starting point for a new task instead of training a new model from scratch. This approach has gained popularity in recent years due to the high cost of training large neural networks on big datasets. Transfer learning can significantly reduce the time and computational resources needed to build accurate deep learning models, especially when the new task has a small amount of data.

1. **VGG16**

   VGG16 is a deep convolutional neural network architecture proposed by the Visual Geometry Group at the University of Oxford. It was introduced in the 2014 ImageNet Large Scale Visual Recognition Challenge, where it achieved state-of-the-art performance on object recognition tasks. The architecture consists of 16 layers, including 13 convolutional layers and 3 fully connected layers, and has been widely used as a base model for transfer learning in various computer vision tasks.

2. **ResNET**

   ResNet, short for Residual Network, is a deep neural network architecture that was introduced by Kaiming He et al. in 2015. ResNet's unique feature is the use of residual connections, which allow for the training of much deeper neural networks without suffering from the problem of vanishing gradients. This makes ResNet especially well-suited for image recognition tasks.

3. **INCEPTIONV3**

   InceptionV3 is a deep convolutional neural network architecture that was developed by Google. It is notable for its efficient use of computational resources and its ability to achieve high accuracy on image classification tasks. It has been widely used in various computer vision applications, including image recognition, object detection, and image segmentation.

4. XCEPTION

Xception is a deep neural network architecture proposed by Google researchers in 2016. It is an extension of the Inception architecture that replaces the standard Inception modules with depthwise separable convolutions, which are computationally efficient and lead to improved performance. Xception has achieved state-of-the-art results on various computer vision tasks and is commonly used in transfer learning applications.

CHAPTER-2

LITERATURE SURVEY

2.1 RICE LEAF DISEASE CLASSIFICATION USING CNN

AUTHOR: Priyanshi Singh, Monica Ramchandani

YEAR OF PUBLICATION:2022

In this paper we are working on an automated system to detect the fungal disease of rice plants which is a major cause for a loss of a rice plant. This type of disease may occur and spread due to climate change and moisture on the leaves. The first step is acquiring the image. The act of obtaining a picture from a source is known as image acquisition. The input can be taken from different resources especially hardware such as sensors or cameras etc. This is a crucial step in the entire process because processing is impossible without an image. This step is always the initial phase in the process. The second step is dataset collection. The dataset in our method consists of a number of images of the four diseases of rice leaf which we are predicting. We train the dataset using a wide variety of images to get better accuracy. The next step is preprocessing of data. Collected data should be preprocessed involving data cleaning and removing all the inconsistencies from the data. We do this process to obtain a clean dataset for achieving better accuracy. Then we perform feature extraction which involves extraction of only those features which are important and which are most required. Then we use various DL methods which we are employing in our method and train and test the dataset accordingly. Finally, the images are classified according to their respective diseases.

ADVANTAGES

- Achieved high accuracy and efficiency in rice leaf disease detection.
- CNNs demonstrated the ability to learn complex patterns and variations in leaf diseases, leading to robust classification performance.

DISADVANTAGES

- Limited discussion on model interpretability and the rationale behind architecture selection.
- Dependency on annotated datasets, which may be limited in size and diversity.

2.2 ENHANCING RICE LEAF DISEASE DETECTION USING TRANSFER LEARNING

Author: patel et al.

Year of publication: 2020

This research aimed to improve rice leaf disease detection accuracy by leveraging transfer learning techniques with pre-trained CNN models. The study addressed the challenge of limited annotated data by transferring knowledge from large-scale datasets to the target domain. The techniques used in this project is Transfer learning with pre-trained CNN models, such as InceptionV3 and MobileNet, was employed to extract relevant features from rice leaf images. Fine-tuning and retraining were performed on the pre-trained models using the available annotated dataset.

ADVANTAGES

- Leveraged knowledge from pre-trained models to enhance generalization to unseen data.
- Addressed data scarcity issues by effectively utilizing information from related domains.

DISADVANTAGES

- Fine-tuning process may require careful hyperparameter tuning and computational resources.
- Performance heavily relies on the similarity between the source and target domains, which may not always align perfectly.

2.3 ROBUST RICE LEAF DISEASE DETECTION USING DATA AUGMENTATION

Author: Lee et al.

Year of Publication: 2019

This study proposed a methodology for robust rice leaf disease detection by incorporating data augmentation techniques into the training pipeline. The research aimed to increase the diversity and size of the training dataset to improve model generalization and resilience to variations in disease patterns. Techniques used in this project is CNN architecture combined with data augmentation strategies, including rotation, flipping, and zooming, was employed to augment the training dataset. The study investigated the impact of different augmentation techniques on model performance.

ADVANTAGES

- Robustness and generalization to unseen data by increasing dataset diversity.
- Mitigated overfitting and enhanced model performance on real-world datasets with varying disease manifestations.

DISADVANTAGE

- Augmentation strategies may introduce synthetic artifacts and distortions that do not accurately represent real-world data.
- Optimal selection and tuning of augmentation parameters require careful experimentation and domain expertise.

2.4 RICE LEAF DISEASES DETECTION BASED ON DEEP CONVOLUTIONAL NEURAL NETWORKS

Author: Li et al.

Year of Publication: 2018

This project Rice leaf diseases detection based on deep convolutional neural networks is aimed to develop an automated system for detecting rice leaf diseases using deep convolutional neural networks. The research focused on exploring the effectiveness of CNNs in identifying various types of rice leaf diseases to aid in timely disease management. The Techniques used in this project is CNN architectures, including AlexNet and GoogLeNet, were employed for feature extraction and classification. The study utilized a dataset of labeled rice leaf images containing multiple disease classes for model training and evaluation.

ADVANTAGES

- Demonstrated the feasibility of using deep CNNs for accurate and efficient rice leaf disease detection.
- CNNs showed robustness to variations in disease patterns and leaf conditions, enabling reliable disease diagnosis.

DISADVANTAGES

- Limited discussion on model optimization and hyperparameter tuning, which may impact model performance.
- Evaluation primarily focused on classification accuracy, with limited analysis of false positives and false negatives.

CHAPTER-3

SYSTEM ANALYSIS

3.1 EXSISTING SYSTEM

RiceLeafNet is an automated system designed for the detection and classification of rice leaf diseases using deep learning techniques, particularly CNNs. The system aims to assist farmers and agricultural experts in identifying diseases early to prevent yield loss and optimize crop management practices. The users can capture images of rice leaves using a smartphone or a digital camera. The captured images undergo preprocessing steps such as resizing, normalization, and noise reduction to enhance image quality and facilitate feature extraction.A CNN architecture, such as VGG-16 or ResNet, is employed for feature extraction from preprocessed rice leaf images. The CNN model is trained on a large dataset of labelled rice leaf images containing various disease classes. Extracted features are fed into the trained CNN model for disease classification. The system is capable of distinguishing between different types of rice leaf diseases, including bacterial blight, blast, sheath blight, and brown spot. RiceLeafNet features a user-friendly interface accessible via web or mobile application. Users can upload rice leaf images, initiate disease detection, and view the classification results in real-time.

3.1.1 DISADVANTAGES OF EXISTING SYSTEM

- Dependency on Data Quality
- Training and deploying CNN models require significant computational resources, which may pose challenges in resource-constrained environments.

3.2 PROPOSED SYSTEM

Transfer learning is a technique in deep learning where a pre-trained neural network model is used as a starting point for a new task instead of training a new model from scratch. This approach has gained popularity in recent years due to the high cost of training large neural networks on big datasets. Transfer learning can significantly reduce the time and computational resources needed to build accurate deep learning models, especially when the new task has a small amount of data.

In our rice leaf disease Detection project, we can leverage transfer learning by using pre-trained models that have been trained on large image datasets, such as ImageNet, and fine-tune them on our rice leaf disease dataset. This can help our model learn features and patterns that are common across different image recognition tasks, and improve its accuracy and generalization capabilities.

Some of the popular pre-trained models used in transfer learning include VGG, ResNet, Inception, and Xception. These models have achieved state-of-the-art performance on various image classification tasks and are available in popular deep learning frameworks like TensorFlow and Keras. Transfer Learning: Leveraging InceptionV3 as a pre-trained model allows for efficient transfer learning. Transfer learning involves using the knowledge gained from training on a source task (ImageNet classification) and applying it to a target task (rice leaf disease detection). By utilizing the pre-trained weights of InceptionV3, we can initialize the network with parameters that have already learned generic image features, which can significantly speed up the training process and improve the model's performance, especially when dealing with a limited amount of labelled data.

3.2.1 ADVANTAGES OF PROPOSED SYSTEM

HIGH ACCURACY: CNNs are known for their ability to learn intricate patterns and features from images, making them highly accurate in image classification tasks. Transfer learning further enhances accuracy by leveraging pre-trained models like InceptionV3, which have already learned a rich set of features from a vast dataset . This enables the model to achieve high accuracy even with limited labeled data for the target task of rice leaf disease detection.

TIME AND RESOURCE EFFICIENCY: Transfer learning with InceptionV3 significantly reduces the time and computational resources required for training compared to training a CNN from scratch.

AUTOMATION AND EFFICIENCY: By automating the process of disease detection, CNN-based systems reduce the reliance on manual inspection and human expertise. This leads to more efficient and consistent diagnosis, allowing for early detection and timely intervention to prevent the spread of diseases and minimize crop losses.

3.2.2 MAJOR DISEASE

1. Bacterial leaf blight, caused by the bacterium Xanthomonas oryzae pv. oryzae, is a serious disease that can cause extensive damage to rice plants. The symptoms of bacterial leaf blight include water-soaked lesions on the leaves, which later turn brown and dry up. In severe

cases, the disease can cause wilting and death of the plant, leading to a significant reduction in crop yield.

2. Brown spot, caused by the fungus Cochliobolus miyabeanus, is another major rice disease that can cause significant yield losses. The disease is characterized by small, oval to elliptical spots on the leaves, which turn brown with a yellow halo. In severe cases, the spots can coalesce and cause the leaves to wither and die. Brown spot can also affect the panicles, leading to a reduction in grain quality and yield.

3. Leaf smut, caused by the fungus Entyloma oryzae, is a relatively less common rice disease. The symptoms of leaf smut include the formation of small, round, and reddish-brown spots on the leaves. These spots later turn black and produce powdery spores. Although leaf smut does not usually cause significant yield losses, it can affect the quality of rice grains by reducing their weight and size.

CHAPTER – 4

SYSTEM SPECIFICATION

4.1 HARDWARE REQUIREMENTS

• Processor	: Intel processor 2.6.0 GHZ
• RAM	: 1GB
• Hard disk	: 160 GB
• Compact Disk	: 650 Mb
• Keyboard	: Standard keyboard
• Monitor	: 15inch color monitor

4.2 SOFTWARE REQUIREMENTS

• Operating System	: Window 10
• Front End	: PYTHON
• Back End	: GOOGLE COLAB

CHAPTER-5

SYSTEM DESIGN

5.1 SYSTEM ARCHITECTURE

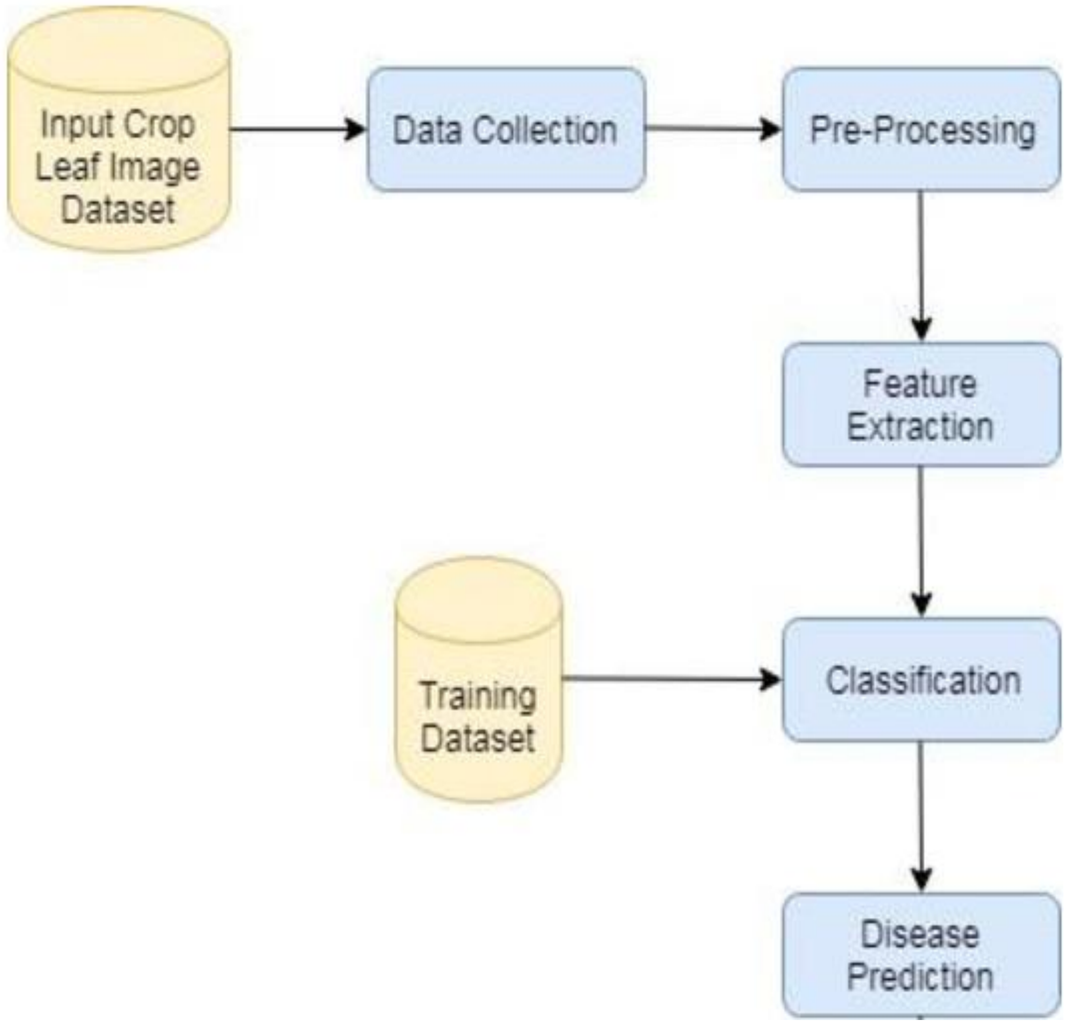


Fig.5.1 SYSTEM ARCHITECTURE

CHAPTER-6

MODULE DESCRIPTION

6.1 LIST OF MODULES

- Data Collection
- Preprocessing
- Data Augmentation
- Feature Extraction
- Classification

6.2 MODULE DESCRIPTIONS

6.2.1 DATA COLLECTION

- Gather a dataset of rice leaf images, including healthy leaves and leaves affected by various diseases prevalent in rice plants.
- Ensure the dataset is diverse, representative, and sufficiently large to train a robust CNN model.

6.2.2 PREPROCESSING

- Normalization of image data is an important step in the pre-processing. It is used to train a neural network.
- Normalization can help to reduce the effects of lighting conditions, noise and other factors that can cause variation in the input data.
- Overall, normalization is an important step that can improve the performance and accuracy of a neural network by ensuring that the input data is in a consistent and standardized format.

- This step involves data cleaning, data transformation, and data reduction (data compression) Data cleaning involves cleaning the data. It removes the noise present in the data.
- Data transformation is transforming high-level data into low-level data for easier calculations.
- Data reduction involves reducing the data dimensions so that the data is not high dimensional but the quality of data remains the same. Data cube aggregation which is summarizing the data.

6.2.3 DATA AUGMENTATION

- Data augmentation is a technique used to increase the size and diversity of a dataset by applying various transformations to the existing data.

- This technique has become an essential tool in computer vision and image processing tasks, such as object recognition and classification, due to its ability to enhance the generalization ability of machine learning models and prevent overfitting.

- In the context of our rice leaf disease image classification project, data augmentation can play a crucial role in improving the performance and robustness of our model.

- By generating new images with different variations such as rotations, flips, zooms, and other transformations, we can increase the diversity of our dataset and provide our model with more examples to learn from, which can lead to better classification accuracy and robustness to variations in the real-world data.

6.2.4 FEATURE EXTRACTION

- Extract relevant features from the preprocessed images.
- This step may involve techniques such as edge detection, texture analysis, and color feature extraction to capture distinguishing characteristics of diseased and healthy leaves.

6.2.5 CLASSIFICATION

- Choose an appropriate classification model for the task. Convolutional Neural Networks (CNNs) are commonly used for image classification tasks due to their ability to automatically learn discriminative features from data.
- Here, we applied a Convolutional neural network (CNN) based approach which is a method of DL that takes input as an image and gives importance to many other objects in the image, as well as differentiates between them. The amount of pre-processing needed by a CNN is substantially less than that required by other classification methods.
- Our architecture mainly contains the following layers:
    1. Convolution layer
    2. Pooling layer
    3. Fully connected layer
- Input Layer: The input layer of CNN consists of the dataset. The input data will be represented as a 3X3 matrix.
- Convolution Layer: A layer that uses filters to learn from smaller sections of input data to obtain features from an image.

- Pooling Layer: This layer is used to shrink the image's dimensionality, lowering the processing power required for subsequent layers. There are two variations of pooling. They are,
- Max pooling: The pixel with the maximum value as input is selected and transferred to the output while parsing input. It is the most used approach compared to average pooling.
- Fully Connected Layer (Dense): This is one of CNN's last layers, and it can
- recognize features that are significantly linked with the output class. The result is a one-dimensional vector created by flattening the pooling layer results.
- Dropout Layer: Used to reduce model overfitting problem by removing a random set of neurons in that layer. It is connected with the FC layer.
- SoftMax Layer: This is the network's last layer that assists in classifying individual input images of the dataset into several classes depending on the learned properties from the network.
- Output Layer: The output layer holds the final classification result.

CHAPTER – 7

SYSTEM IMPLEMENTATION

7.1 ALGORITHM

7.1.1 VGG16

VGG16 is a deep convolutional neural network architecture that was proposed by the Visual Geometry Group (VGG) at the University of Oxford. It is widely used for various computer vision tasks, including image classification, object detection, and segmentation.

We have used Visual Geometry Group (VGG) 16 layers model. The input image size has set to 224 × 224 × 3, and it is connected to next convolution layer Conv1. Then combination of convolutional and max pooling layers is applied for feature extraction and feature reduction respectively.

In a few settings, it includes 11 convolutional filters which are considered to extract the features from input image and send these features to subsequent layers for further processing.To extract the features from corners of the image we used “similar padding” of one row and column each side and stride is set to “1”. Spatial pooling is done via five max pooling layers that follow part of the Conv Layers.

7.1.2 INCEPTIONV3

The Inception architecture, also known as GoogLeNet, is another deep convolutional neural network architecture designed by Google. It is well-known for its computational efficiency and ability to capture complex features from images while maintaining a relatively low number of parameters compared to other architectures like VGG or ResNet.

- Inception Modules: The key innovation in the Inception architecture is the use of inception modules, which are composed of multiple convolutional filters of different sizes (1x1, 3x3, 5x5) applied in parallel to the input. This allows the network to capture features at different spatial scales efficiently. Additionally, max-pooling layers are used to downsample the spatial dimensions and reduce the computational burden.
- Feature Extraction: Similar to VGG16, the early layers of the Inception network serve as feature extractors, learning to detect low-level features such as edges, textures, and patterns from input rice leaf images. The use of multiple convolutional filters of different sizes in the inception modules enables the network to capture features at various scales and resolutions, making it particularly effective for tasks like object detection and image classification.
- Transfer Learning: As with VGG16, you can leverage transfer learning by initializing the Inception network with weights pre-trained on a large dataset such as ImageNet. This allows the network to benefit from the learned representations of generic features while adapting them to the specific task of rice leaf disease detection.
- Fine-Tuning: After initializing the Inception network with pre-trained weights, you can fine-tune it on the dataset of rice leaf images. During fine-tuning, the weights of the network are updated using backpropagation to better represent the features present in the rice leaf images, while the final classification layers are trained from scratch to classify the images into healthy or diseased categories.

- Training and Evaluation: Train the fine-tuned Inception network on the dataset of rice leaf images, monitoring its performance on separate validation set to avoid overfitting. Once training is complete, evaluate the performance of the model on a test set using metrics such as accuracy, precision, recall, and F1-score to assess its ability to classify healthy and diseased rice leaves.

7.1.3 RESNET

- ResNet, short for Residual Network, is a deep neural network architecture that was introduced by Kaiming He et al. in 2015. ResNet's unique feature is the use of residual connections, which allow for the training of much deeper neural networks without suffering from the problem of vanishing gradients. This makes ResNet especially well-suited for image recognition tasks.
- The ResNet architecture typically consists of multiple layers, including convolutional layers, pooling layers, and fully connected layers. However, the core building blocks of ResNet are residual blocks, which contain skip connections. These connections enable the network to learn residual functions with reference to the layer inputs, making it easier to optimize and train deeper networks.
- The ResNet model is trained on the labeled dataset using a supervised learning approach. During training, the model learns to classify images into different categories (e.g., healthy or diseased) by adjusting the parameters of the neural network to minimize a predefined loss function, such as cross-entropy loss.
- Depending on the specific application and dataset, fine-tuning may be performed to improve the performance of the ResNet model. This

involves adjusting hyperparameters, such as learning rate, batch size, and regularization techniques, as well as potentially fine-tuning the pre-trained ResNet model on the specific rice leaf disease detection dataset.

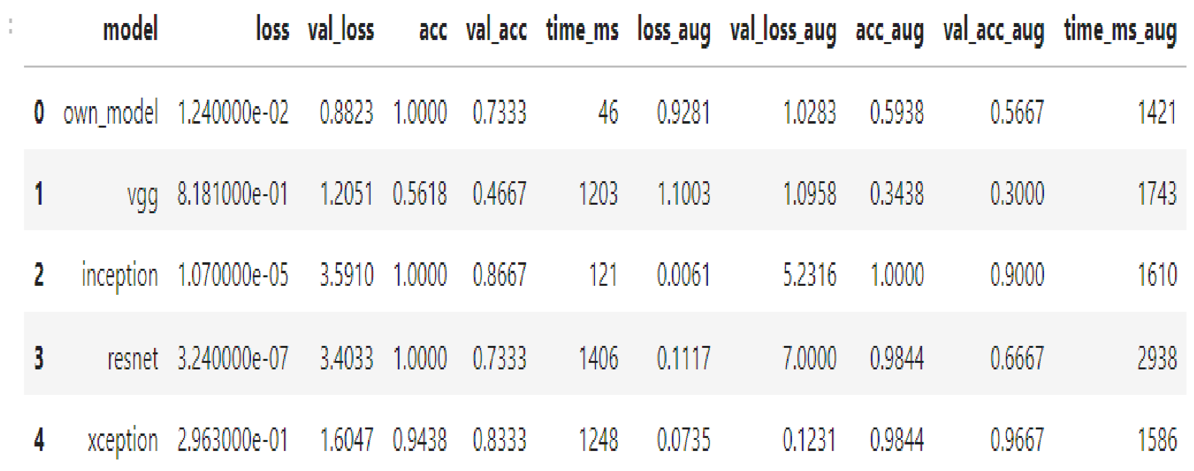
- Once the ResNet model is trained and fine-tuned, it can be used to classify new, unseen rice leaf images as healthy or diseased.

7.1.4 XCEPTION

- Xception is a deep neural network architecture proposed by Google researchers in 2016. It is an extension of the Inception architecture that replaces the standard Inception modules with depthwise separable convolutions, which are computationally efficient and lead to improved performance.
- Xception has achieved state-of-the-art results on various computer vision tasks and is commonly used in transfer learning applications.
- Xception is a convolutional neural network with just convolution layers according to their depth. The Xception architecture's feature extraction building is constructed up of 36 convolutional layers
- . In our experimental evaluation, we will just look at picture classification, therefore our results will be constrained. Just after the convolutional base layer, regression will be utilized. Fully-connected layers, as illustrated in the example, can be introduced before the logistic regression layer.

- The experimental assessment portion. Six categories are used to categorize the 36 convolutional layers. There are a total of 14 modules, each having its own linear residual connection. Everything else is constructed around them, with the exception of the first and last modules.

7.1.5 MODEL COMPARISON AND ANALYSIS



	model	loss	val_loss	acc	val_acc	time_ms	loss_aug	val_loss_aug	acc_aug	val_acc_aug	time_ms_aug
0	own_model	1.240000e-02	0.8823	1.0000	0.7333	46	0.9281	1.0283	0.5938	0.5667	1421
1	vgg	8.181000e-01	1.2051	0.5618	0.4667	1203	1.1003	1.0958	0.3438	0.3000	1743
2	inception	1.070000e-05	3.5910	1.0000	0.8667	121	0.0061	5.2316	1.0000	0.9000	1610
3	resnet	3.240000e-07	3.4033	1.0000	0.7333	1406	0.1117	7.0000	0.9844	0.6667	2938
4	xception	2.963000e-01	1.6047	0.9438	0.8333	1248	0.0735	0.1231	0.9844	0.9667	1586

Fig.7.1.5 COMPARISON AND ANALYSIS

Based on the provided data, it appears that the Xception model trained on augmented data has the highest validation accuracy among the models evaluated, with a validation accuracy of 0.9667. Additionally, the Xception model has a relatively low validation loss, indicating that it is effectively minimizing the difference between predicted and actual labels. Furthermore, the Xception model appears to be relatively efficient, with a time per epoch of 1586 ms.

Overall, the Xception model appears to have the best balance of accuracy and efficiency among the five models evaluated, which is why we will select it as the final model.

CHAPTER-8

CONCLUSION

8.1 CONCLUSION

In the project involved building a deep learning model to classify images of rice leaf disease. The dataset contained images from 3 type of rice leaf disease namely bacterial blight, brown spot, and leaf smut. The project was divided into several steps, including data exploration, data preprocessing, building and training deep learning models, and evaluating model performance.During data exploration, we analysed the dataset and visualized the images to get a better understanding of the data. We observed that the dataset was balanced, with an equal number of images for rice leaf diseases. We also noticed that the images were of different sizes and needed to be resized to a uniform size before being used for training. We have rescaled them to uniform size of 224 X 224.For data preprocessing, We have normalized the training dataset and also encoded the labels. We used Keras Image Data Generator to generate augmented images to increase the size of the dataset and reduce overfitting. We also resized the images and divided them into training, validation, and testing sets.

We built several deep learning models by applying transfer learning technique, including a custom model, VGG, Inception, ResNet, and Xception. We trained these models on both the original and augmented datasets and evaluated their performance using accuracy, loss, and time taken per epoch. We also plotted the training and validation curves to analyse the models' behaviour during training.

We observed that the custom model performed well on the original dataset, but its performance improved on the augmented dataset. Xception, in particular, performed the best, achieving the highest accuracy and the lowest loss on both datasets. Based on these observations, we selected Xception as the final model and used it to predict the class of new images.

Overall, the project was successful in building a deep learning model to classify images of rice leaf diseases. We explored the dataset, pre-processed the data, built and trained deep learning models, and evaluated their performance. We also gained insights into how different models behave when trained on normal vs. augmented data, what makes a good model in terms of accuracy and loss, and how to analyse models with respect to their time taken per epoch. Finally, we selected Xception as the best model and achieved good accuracy on new images.

8.2 FEATURE ENHANCEMENTS

Enhancing a rice leaf disease detection system involves improving its accuracy, efficiency, and usability. Here are some potential feature enhancements:

1. **Advanced Image Processing Techniques:** Incorporate advanced image processing techniques like convolutional neural networks (CNNs) or deep learning models to accurately detect and classify rice

leaf diseases from images. These techniques can help in better feature extraction and classification, leading to higher accuracy.

2. **Multi-Spectral Imaging:** Integrate multi-spectral imaging technology to capture images of rice leaves at different wavelengths. This can provide additional information about the health status of the plants, enabling more precise disease detection and early diagnosis.
3. **User-Friendly Interface:** Design an intuitive and user-friendly interface for the system, keeping in mind the technical proficiency and accessibility needs of the target users. Provide clear instructions and guidance to ensure easy adoption and usage.

CHAPTER- 9

APPENDIX

9.1 SOURCE CODE

```
import os

import sys

from tempfile import NamedTemporaryFile

from urllib.request import urlopen

from urllib.parse import unquote, urlparse

from urllib.error import HTTPError

from zipfile import ZipFile

import tarfile

import shutil

CHUNK_SIZE = 40960

DATA_SOURCE_MAPPING = 'rice-leaf-
diseases:https%3A%2F%2Fstorage.googleapis.com%2Fkaggle-data-
sets%2F522658%2F959791%2Fbundle%2Farchive.zi

KAGGLE_INPUT_PATH='/kaggle/input'

KAGGLE_WORKING_PATH='/kaggle/working'

KAGGLE_SYMLINK='kaggle'

!umount /kaggle/input/ 2> /dev/null

shutil.rmtree('/kaggle/input', ignore_errors=True)
```

```
os.makedirs(KAGGLE_INPUT_PATH, 0o777, exist_ok=True)

os.makedirs(KAGGLE_WORKING_PATH, 0o777, exist_ok=True)

try:

os.symlink(KAGGLE_INPUT_PATH, os.path.join("..", 'input'),
target_is_directory=True)

except FileExistsError:

pass

try:

os.symlink(KAGGLE_WORKING_PATH, os.path.join("..", 'working'),
target_is_directory=True)

except FileExistsError:

pass

for data_source_mapping in DATA_SOURCE_MAPPING.split(','):

directory, download_url_encoded = data_source_mapping.split(':')

download_url = unquote(download_url_encoded)

filename = urlparse(download_url).path

destination_path = os.path.join(KAGGLE_INPUT_PATH, directory)

try:

with urlopen(download_url) as fileres, NamedTemporaryFile() as tfile:

total_length = fileres.headers['content-length']

print(f'Downloading {directory}, {total_length} bytes compressed')
```

```
dl = 0
data = fileres.read(CHUNK_SIZE)
while len(data) > 0:
dl += len(data)
tfile.write(data)
done = int(50 * dl / int(total_length))
sys.stdout.write(f"\r[{'=' * done}{' ' * (50-done)}] {dl} bytes
downloaded")
sys.stdout.flush()
data = fileres.read(CHUNK_SIZE)
if filename.endswith('.zip'):
with ZipFile(tfile) as zfile:
zfile.extractall(destination_path)
else:
with tarfile.open(tfile.name) as tarfile:
tarfile.extractall(destination_path)
print(f'\nDownloaded and uncompressed: {directory}')
except HTTPError as e:
print(f'Failed to load (likely expired) {download_url} to path
{destination_path}')
continue
```

```
except OSError as e:

print(f'Failed to load {download_url} to path {destination_path}')

continue

print('Data source import complete.')

#Checking Memory & Requirements

!cat /proc/cpuinfo

!cat /proc/meminfo
```

Installing Modules

```
!pip install keras -tuner

!pip install keras_tuner
```

Importing Libraries

```
#basics

import pandas as pd

import numpy as np

import tensorflow as tf

import keras

# visualisation

import matplotlib.pyplot as plt

import seaborn as sns

#utility & processing

from tensorflow.keras import utils
from tensorflow.keras.utils import to_categorical

from sklearn.model_selection import train_test_split

#cnn architecture

from tensorflow.keras.layers import Dense, Conv2D, MaxPooling2D,
Flatten

from tensorflow.keras.models import Sequential

#callbacks

from tensorflow.keras.callbacks import ModelCheckpoint, EarlyStopping

#data augmentation

from tensorflow.keras.preprocessing.image import ImageDataGenerator

#keras tuner

import keras_tuner

from keras_tuner import RandomSearch

from keras_tuner.engine.hyperparameters import HyperParameters

#transfer learning

from keras.applications.vgg16 import VGG16

from keras.applications.inception_v3 import InceptionV3

from keras.applications.resnet_v2 import ResNet152V2

from keras.applications.xception import Xception

#warnings

import warnings
```

```
warnings.filterwarnings('ignore')

print('we are currently using tensorflow version', tf._version_)
```

Setting up an environment

```
input_shape_2D=(224,224)

input_shape_3D= (224,224,3)

seed=1

batch_size=32

epochs=30

stopper=EarlyStopping(patience=5)
```

Uploading image dataset

```
#loading image data

data_ds= tf.keras.utils.image_dataset_from_directory(

directory='/kaggle/input/rice-leaf-diseases/rice_leaf_diseases',

labels="inferred",

label_mode="int",

class_names=None,

color_mode="rgb",

batch_size=None,

image_size=input_shape_2D,

seed=seed

)
```

```
#class labels

class_names = data_ds.class_names

print("Class Names:", class_names)

def show_images(images, labels):

fig, axes = plt.subplots(8, 4, figsize=(10, 10))

axes = axes.flatten()

for i, (img, label) in enumerate(zip(images, labels)):

axes[i].imshow(img.numpy().astype("uint8"))

axes[i].set_title(f"Label: {label}")

axes[i].axis("off")

plt.tight_layout()

plt.show()

#displaying image batch of 32

batched_ds = data_ds.batch(batch_size)

batch = next(iter(batched_ds))

images, labels = batch

images, labels = next(iter(batched_ds))

show_images(images, labels)
```

Data Processing

```
X=[]

y= []
```

```
for image, label in data_ds.batch(119):

for img, lab in zip(image, label):

X.append(img)

y.append(lab.numpy())

# converting to numpy array

X= np.array(X)

y= np.array(y)

# train test split

X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=seed)

keyboard_arrow_down Normalisation

X_train = X_train/255

X_test = X_test/255

print("shape of X_train_scaled:", X_train.shape)

print("shape of y_train:", y_train.shape)

print("shape of X_test_scaled:", X_test.shape)

print("shape of y_test:", y_test.shape)

y_train = to_categorical(y_train, num_classes=len(class_names))

y_test = to_categorical(y_test, num_classes=len(class_names))
```

CNN Architecture

```
own_model=Sequential()

# adding first convloutional layer
```

```
own_model.add(Conv2D(16, kernel_size=(3,3), padding='same',
activation='relu', input_shape=input_shape_3D))

own_model.add(MaxPooling2D())

# adding second convloutional layer

own_model.add(Conv2D(32, kernel_size=(3,3), padding='same',
activation='relu'))

own_model.add(MaxPooling2D())

# adding third convloutional layer

own_model.add(Conv2D(64, kernel_size=(3,3), padding='same',
activation='relu'))

own_model.add(MaxPooling2D())

# adding flatten layer

own_model.add(Flatten())

# adding fully connected layers

own_model.add(Dense(200, activation='relu'))

#adding output layer : number of classes are 3

own_model.add(Dense(3, activation='softmax'))

# looking at the architecture summary

own_model.summary()

# compiling model

own_model.compile(optimizer='adam', loss='categorical_crossentropy',
metrics=['accuracy'])
```

```
#training model

checkpointer_own =
ModelCheckpoint(filepath='own_model.weights.best.hdf5', verbose=3,
save_best_only=True)

history=own_model.fit(X_train, y_train, validation_data= (X_test, y_test),
epochs=epochs, callbacks=[checkpointer_own, stopper])

def plot_performance(model_name, acc, val_acc, loss, val_loss):

plt.figure(figsize=(10,5))

plt.subplot(1,2,1)

plt.plot(loss)

plt.plot(val_loss)

plt.title('{} : Loss'.format(model_name), fontsize=12, fontweight='bold')

plt.ylabel('Loss')

plt.xlabel('Epoch')

plt.legend(['train', 'test'], loc='upper right')

plt.subplot(1,2,2)

plt.plot(acc)

plt.plot(val_acc)

plt.title('{} : Accuracy'.format(model_name), fontsize=12,
fontweight='bold')

plt.ylabel('Accuracy')

plt.xlabel('Epoch')
plt.legend(['train', 'test'], loc='upper right')

plt.tight_layout()

plot_performance(model_name= 'Own Model',

acc=history.history['accuracy'],

val_acc=history.history['val_accuracy'],

loss=history.history['loss'],

val_loss=history.history['val_loss'])
```

Data Augmentation

```
#creating image generator object

datagen = ImageDataGenerator(

rotation_range=40,

width_shift_range=0.3,

height_shift_range=0.3,

horizontal_flip=True,

vertical_flip=True)

#training model

checkpointer_own_aug =
ModelCheckpoint(filepath='own_model_aug.weights.best.hdf5',
verbose=3, save_best_only=True)

aug_history= own_model.fit(

datagen.flow(X_train, y_train, batch_size=batch_size),
validation_data=(X_test, y_test),
steps_per_epoch=len(X_train)//batch_size,
epochs=epochs,
callbacks=[stopper, checkpointer_own_aug]
)
#visualising model performance
plot_performance(model_name='Own_Aug',
acc=aug_history.history['accuracy'],
val_acc=aug_history.history['val_accuracy'],
loss=aug_history.history['loss'],
val_loss=aug_history.history['val_loss'])
def build_model(hp):
new_model=Sequential()
# adding first convloutional layer
new_model.add(Conv2D(
filters= hp.Int('conv_1_filter', min_value=8, max_value=32, step=8),
kernel_size=hp.Choice('conv_1_kernel', values= [2,3]),
padding=hp.Choice('conv_1_pad', values=['same', 'valid']),
activation='relu', input_shape=input_shape_3D))
new_model.add(MaxPooling2D())
# adding second convloutional layer
```

```
new_model.add(Conv2D(
filters= hp.Int('conv_2_filter', min_value=16, max_value=64, step=16),
kernel_size=hp.Choice('conv_2_kernel', values= [2,3]),
padding=hp.Choice('conv_2_pad', values=['same', 'valid']),
activation='relu'))
new_model.add(MaxPooling2D())
# adding third convloutional layer
new_model.add(Conv2D(
filters= hp.Int('conv_3_filter', min_value=32, max_value=64, step=16),
kernel_size=hp.Choice('conv_3_kernel', values= [2,3]),
padding=hp.Choice('conv_3_pad', values=['same', 'valid']),
activation='relu'))
new_model.add(MaxPooling2D())
# adding flatten layer
new_model.add(Flatten())
# adding fully connected layers
new_model.add(Dense(
units=hp.Int('dense_1_units', min_value=50, max_value=250, step=50),
activation='relu'
))
#adding output layer : number of classes are 3
```

```
new_model.add(Dense(3, activation='softmax'))
new_model.compile(
optimizer=keras.optimizers.Adam(hp.Choice('learning_rate', values=[1e-1,
1e-2, 1e-3])),
loss='categorical_crossentropy',
metrics=['accuracy']
)
tuner_search=RandomSearch(build_model, objective='val_accuracy',
max_trials=3, directory='output', project_name='RiceLeafDisease')
tuner_search.search(
datagen.flow(X_train, y_train),
validation_data=(X_test, y_test),
epochs=10
)
tuned_model=tuner_search.get_best_models(num_models=1)[0]
tuned_model.summary()
#training model
checkpointer_own_tuned =
ModelCheckpoint(filepath='own_model_tuned.weights.best.hdf5',
verbose=3, save_best_only=True)
tuned_history= tuned_model.fit(
datagen.flow(X_train, y_train, batch_size=batch_size),
validation_data=(X_test, y_test),
steps_per_epoch=len(X_train)//batch_size,
initial_epoch=10,
epochs=epochs,
callbacks=[stopper, checkpointer_own_tuned]
)
plot_performance(model_name='Tuned',
acc=tuned_history.history['accuracy'],
val_acc=tuned_history.history['val_accuracy'],
loss=tuned_history.history['loss'],
val_loss=tuned_history.history['val_loss'])
#Transfer Learning
```

InceptionV3

```
inception_base=tf.keras.applications.InceptionV3(
include_top=False,
weights="imagenet",
input_shape=input_shape_3D,
)
#unfreezing the convloution layers
inception_base.trainable=True
set_trainable=False
```

```
for layer in inception_base.layers:

if layer.name=='mixed9_0':

set_trainable=True

if set_trainable==True:

layer.trainable=True

else:

layer.trainable=False

# building fully connected layers

inception_model= Sequential()

inception_model.add(inception_base)

inception_model.add(Flatten())

inception_model.add(Dense(128, activation='relu'))

inception_model.add(Dense(3, activation='softmax'))

#compiling

inception_model.compile(optimizer='adam',
loss='categorical_crossentropy', metrics=['accuracy'])

tf.keras.utils.plot_model(

inception_model,

show_shapes=True,

show_layer_names=True,

show_layer_activations=True
```

```
)
#training model
checkpointer_inception =
ModelCheckpoint(filepath='inception.weights.best.hdf5', verbose=3,
save_best_only=True)
inception_history= inception_model.fit(
X_train,
y_train,
validation_data=(X_test, y_test),
epochs=epochs,
callbacks=[stopper, checkpointer_inception]
)
plot_performance(model_name='Inception',
acc=inception_history.history['accuracy'],
val_acc=inception_history.history['val_accuracy'],
loss=inception_history.history['loss'],
val_loss=inception_history.history['val_loss'])
#training model
checkpointer_inception_aug =
ModelCheckpoint(filepath='inception_aug.weights.best.hdf5', verbose=3,
save_best_only=True)
inception_history_aug= inception_model.fit(
datagen.flow(X_train,y_train, batch_size=batch_size),
validation_data=(X_test, y_test),
steps_per_epoch=len(X_train)//batch_size,
epochs=epochs,
callbacks=[stopper, checkpointer_inception_aug]
)
plot_performance(model_name='Inception_Aug',
acc=inception_history_aug.history['accuracy'],
val_acc=inception_history_aug.history['val_accuracy'],
loss=inception_history_aug.history['loss'],
val_loss=inception_history_aug.history['val_loss'])
```

Xception

```
xception_base= Xception(
include_top=False,
weights='imagenet',
input_shape=input_shape_3D
)
xception_base.trainable=True
set_trainable=False
for layer in xception_base.layers:
if layer.name=='add_8':
```

```
set_trainable=True

if set_trainable==True:

layer.trainable=True

else:

layer.trainable=False

#building fully connected layers

xception_model=Sequential()

xception_model.add(xception_base)

xception_model.add(Flatten())

xception_model.add(Dense(128, activation='relu'))

xception_model.add(Dense(3, activation='softmax'))

xception_model.compile(optimizer='adam',
loss='categorical_crossentropy', metrics=['accuracy'])

tf.keras.utils.plot_model(

xception_model,

show_shapes=True,

show_dtype=False,

show_layer_names=True,

rankdir="TB",

expand_nested=False,

dpi=96,
layer_range=None,
show_layer_activations=True
)
#training model
checkpointer_xception =
ModelCheckpoint(filepath='xception.weights.best.hdf5', verbose=3,
save_best_only=True)
xception_history=xception_model.fit(
X_train,
y_train,
validation_data=(X_test, y_test),
epochs=epochs,
callbacks=[stopper, checkpointer_xception]
)
plot_performance(model_name='Xception',
acc=xception_history.history['accuracy'],
val_acc=xception_history.history['val_accuracy'],
loss=xception_history.history['loss'],
val_loss=xception_history.history['val_loss'])
checkpointer_xception_aug =
ModelCheckpoint(filepath='xception_aug.weights.best.hdf5', verbose=3,
save_best_only=True)
```

```
xception_history_aug=xception_model.fit(
datagen.flow(X_train, y_train, batch_size=batch_size),
validation_data=(X_test, y_test),
steps_per_epoch=len(X_train)//batch_size,
epochs=epochs,
callbacks=[stopper, checkpointer_xception_aug]
)
plot_performance(model_name='Xception_Aug',
acc=xception_history_aug.history['accuracy'],
val_acc=xception_history_aug.history['val_accuracy'],
loss=xception_history_aug.history['loss'],
val_loss=xception_history_aug.history['val_loss'])
```

LOADING BEST MODEL

```
xception_model.load_weights('xception_aug.weights.best.hdf5')
```

Visualising Kernels

```
fig, ax= plt.subplots(3,4, figsize=(8,8))
for i, ax in zip(range(12), ax.flatten()):
conv1=xception_model.layers[0]
weights1 = conv1.get_weights()
kernels1 = weights1[0]
kernel1_1 = kernels1[:,:,0,i]
```

```
ax.imshow(kernel1_1)

ax.set_xticks([])

ax.set_yticks([])

plt.tight_layout()

plt.axis('off');

image = tf.keras.utils.load_img('/kaggle/input/rice-leaf-
diseases/rice_leaf_diseases/Brown spot/DSC_0112.jpg',
target_size=input_shape_2

input_arr = tf.keras.utils.img_to_array(image)/255

image = input_arr[:, :, 1]

plt.imshow(image)

plt.axis('off');

import numpy as np

def convolution(image, kernel):

# Get the size of the kernel

kernel_size = kernel.shape[0]

# Calculate the size of the output image

output_size = image.shape[0] - kernel_size + 1

# Initialize the output image

output_image = np.zeros((output_size, output_size))

# Pad the image with zeros
```

```
padded_image = np.pad(image, ((kernel_size-1)//2, (kernel_size-1)//2),
'constant')

# Apply the convolution operation

for ii in range(output_size):

for jj in range(output_size):

window = padded_image[ii:ii+kernel_size, jj:jj+kernel_size]

output_image[ii, jj] = np.sum(window * kernel)

return output_image

fig, ax= plt.subplots(3,4, figsize=(8,8))

for i, ax in zip(range(12), ax.flatten()):

conv1=xception_model.layers[0]

weights1 = conv1.get_weights()

kernels1 = weights1[0]

kernel1_1 = kernels1[:,:,0,i]

output=convolution(image, kernel1_1)

ax.imshow(output)

ax.set_xticks([])

ax.set_yticks([])

plt.tight_layout()

plt.axis('off');
```

Visualising Prediction

```
# get predictions on the test set
y_hat = xception_model.predict(X_test)
# define text labels
labels = data_ds.class_names
# plot a random sample of test images, their predicted labels, and ground
truth
fig = plt.figure(figsize=(20, 8))
for i, idx in enumerate(np.random.choice(X_test.shape[0], size=20,
replace=False)):
ax = fig.add_subplot(5, 4, i + 1, xticks=[], yticks=[])
ax.imshow(np.squeeze(X_test[idx]))
pred_idx = np.argmax(y_hat[idx])
true_idx = np.argmax(y_test[idx])
ax.set_title("{} ({})".format(labels[pred_idx], labels[true_idx]),
color=("blue" if pred_idx == true_idx else "red"))
plt.tight_layout()
```

9.2 SCREENSHOTS

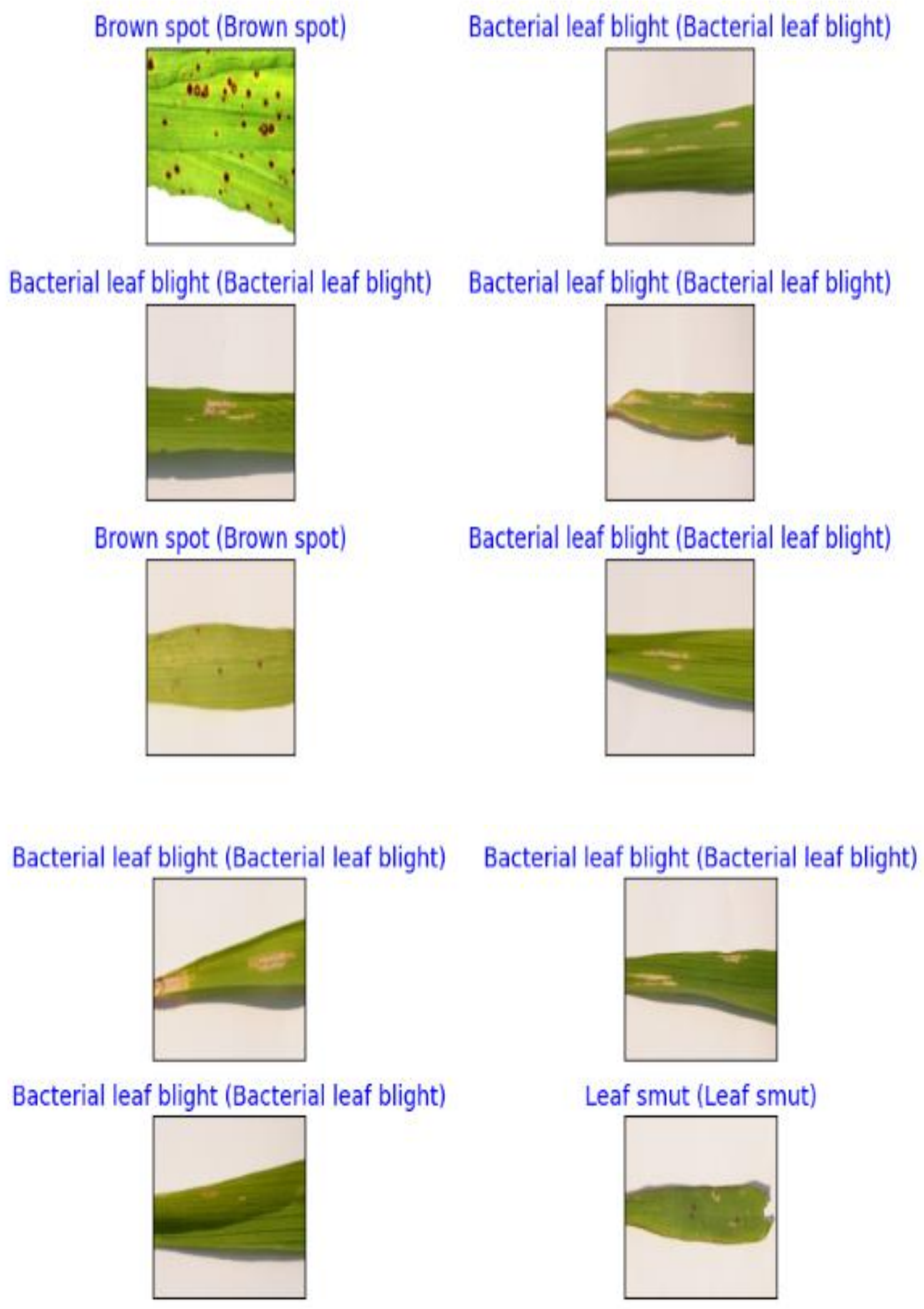

Brown spot (Brown spot)
Bacterial leaf blight (Bacterial leaf blight)
Bacterial leaf blight (Bacterial leaf blight)
Bacterial leaf blight (Bacterial leaf blight)
Brown spot (Brown spot)
Bacterial leaf blight (Bacterial leaf blight)
Bacterial leaf blight (Bacterial leaf blight)
Bacterial leaf blight (Bacterial leaf blight)
Bacterial leaf blight (Bacterial leaf blight)
Leaf smut (Leaf smut)

Fig.9.2.1 DISEASE PREDICTION

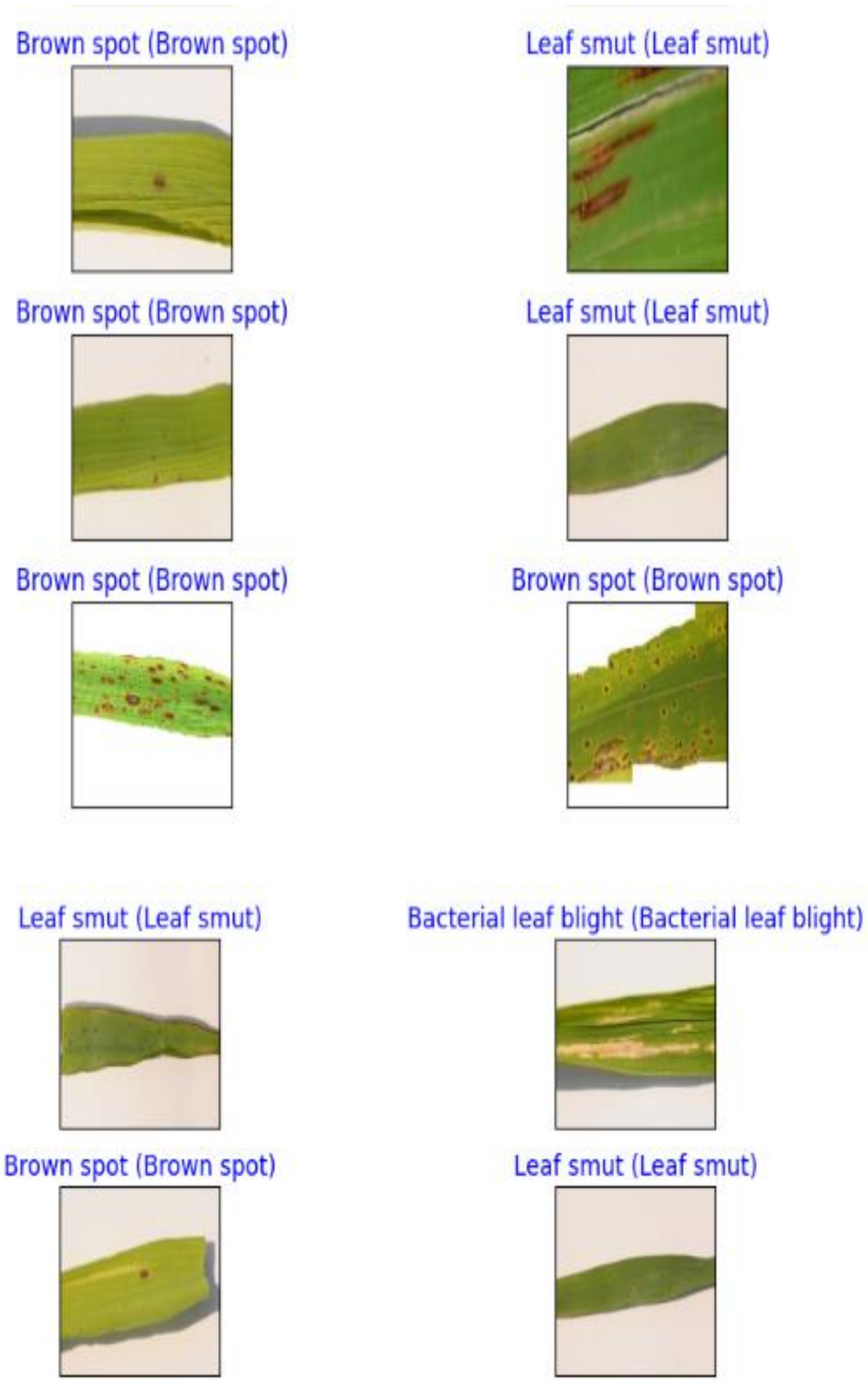


Fig.9.2.2 DISEASE PREDICTION

MODEL COMPARISON

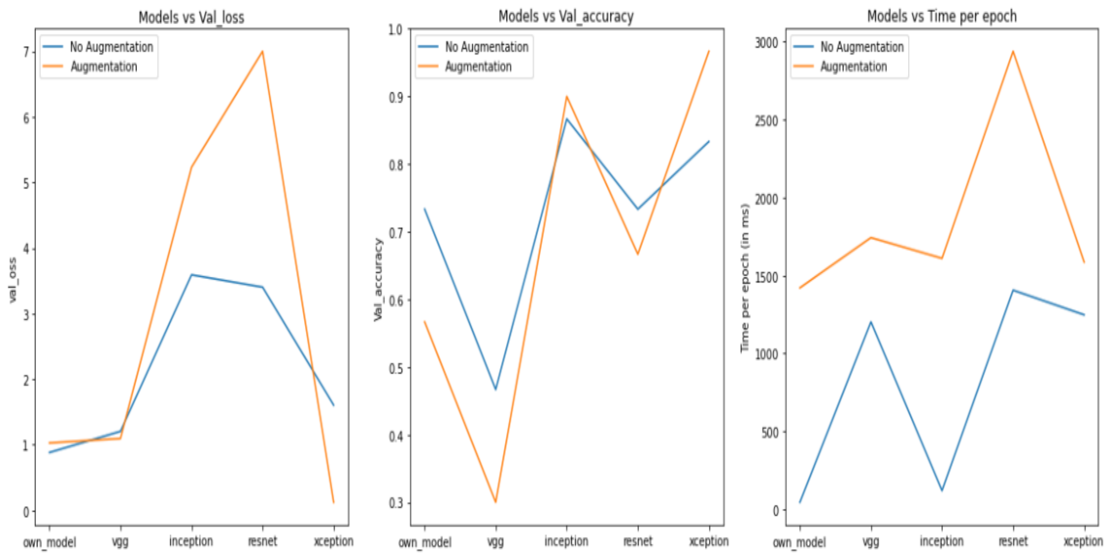
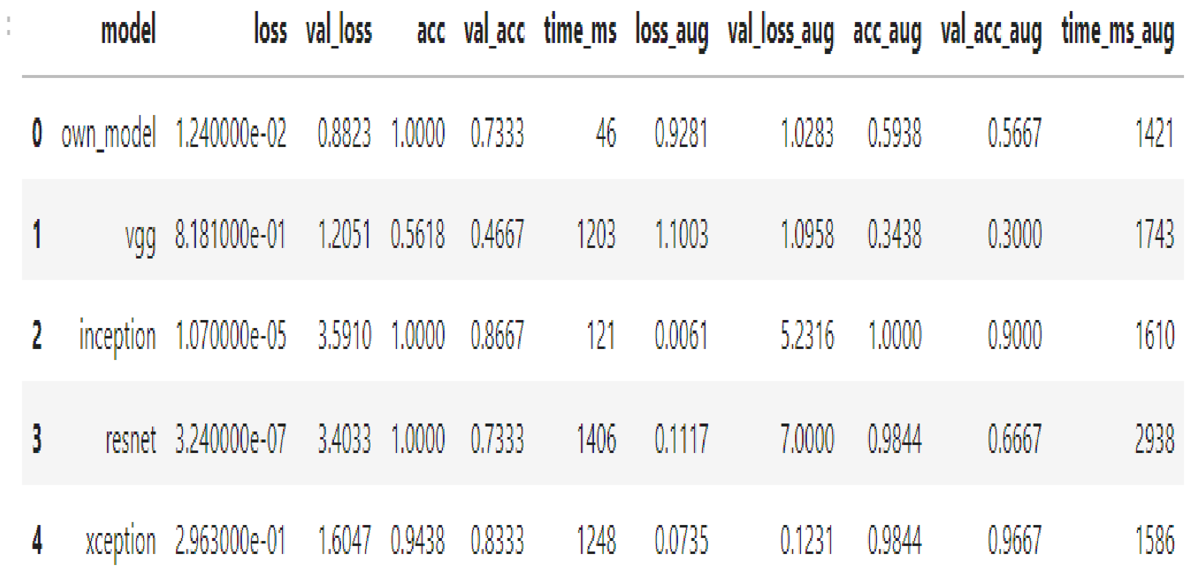


Fig.9.2.3 MODEL COMPARISON

COMPARISON TABLE



	model	loss	val_loss	acc	val_acc	time_ms	loss_aug	val_loss_aug	acc_aug	val_acc_aug	time_ms_aug
0	own_model	1.240000e-02	0.8823	1.0000	0.7333	46	0.9281	1.0283	0.5938	0.5667	1421
1	vgg	8.181000e-01	1.2051	0.5618	0.4667	1203	1.1003	1.0958	0.3438	0.3000	1743
2	inception	1.070000e-05	3.5910	1.0000	0.8667	121	0.0061	5.2316	1.0000	0.9000	1610
3	resnet	3.240000e-07	3.4033	1.0000	0.7333	1406	0.1117	7.0000	0.9844	0.6667	2938
4	xception	2.963000e-01	1.6047	0.9438	0.8333	1248	0.0735	0.1231	0.9844	0.9667	1586

Fig.9.2.4 COMPARISON TABLE

- The own model, inception and resnet models have higher validation loss on Augmented data than normal data.
- The Inception and Xception models haver higher validation accuracy on augmented data than on normal data.
- The "own_model" has the lowest loss and whereas "xception" model has the lowest validation loss and the highest validation accuracy on augmented data.
- Xception' is best performing model whereas 'VGG' is worst performing model on this dataset.

CHAPTER- 9
REFERENCES

[1] Rahman, C. R., Arko, P. S., Ali, M. E., Khan, M. A. I., Apon, S. H., Nowrin, F., & Wasif, A. (2020). “Identification and recognition of rice diseases and pests using convolutional neural networks”. Biosystems Engineering, 194, 112-120.

[2] Kaiming He, Xiangyu Zhang, Shaoqing Ren, Jian Sun. "Deep Residual Learning for Image Recognition." Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2016.

[3] Szegedy, C., Vanhoucke, V., Ioffe, S., Shlens, J., & Wojna, Z. (2018). Rethinking the inception architecture for computer vision. In Proceedings of the IEEE conference on computer vision and pattern recognition (pp. 2818-2826)

[4] Ahmed, K., Shahidi, T. R., Alam, S. M. I., &Momen, S. (2019, December). Rice leaf disease detection using machine learning techniques. In 2019 International Conference on Sustainable Technologies for Industry 4.0 (STI) (pp. 1-5). IEEE.

[5] Ramesh, S., &Vydeki, D. (2018, September). Rice blast disease detection and classification using machine learning algorithm. In 2018 2nd International Conference on Micro-Electronics and Telecommunication Engineering (ICMETE) (pp. 255-259). IEEE

[6] Singh, U.S., Singh, D.P., and Chaube, H.S. (2015). Rice brown spot disease: A review. Journal of Plant Pathology and Microbiology, 6(4), 1-8.

[7] sladojevic, S., Arsenovic, M., Anderla, A., Culibrk, D., Stefanovic, D., and Crnojevic, V. (2016). Deep neural network based recognition of plant diseases by leaf image classification. Computational Intelligence and Neuroscience, 2016, 1-11.

[8] Singh, U.S., Singh, D.P., and Chaube, H.S. (2015). Rice brown spot disease: A review. Journal of Plant Pathology and Microbiology, 6(4), 1-8.

[9] Acedo, A.L., and Daquioag, R.D. (2013). Leaf smut disease of rice. Philippine Agricultural Scientist, 96(4), 354-364.

[10] Vaishnnave, M. P., Devi, K. S., Srinivasan, P., & Jothi, G. A. P. (2019, March). Detection and classification of groundnut leaf diseases using KNN classifier. In 2019 IEEE International Conference on System, Computation, Automation and Networking (ICSCAN) (pp. 1-5). IEEE