International Journal of Scientific Research in Engineering and Management (IJSREM)
Volume: 09 Issue: 05 | May - 2025 SJIF Rating: 8.586 ISSN: 2582-3930

Agri Nama: An Intelligent, Integrated Platform for Digital Governance and Agricultural Empowerment

Shubham Kapase[1], Prabuddha Sonalkar[2], Karan Patil [3] ,Sujal Vadgave[4], Chaitanya Chougale[5], Prof B.D Jitkar

Department of Computer Science and Engineering

D.Y. Patil Collage of Engineering and Technology Kolhapur, Maharashtra, India.

Abstract:

Even though agriculture is a pillar of many economies, farmers nevertheless confront difficulties because of crop diseases, erratic weather patterns, a lack of reliable information, and poor government-to-government communication. We have created a comprehensive mobile application called AgriNama to help farmers with technology-driven solutions in order to address these problems. Multiple modules, including weather forecasting, yield prediction, crop disease detection, an intelligent agricultural chatbot, a farmer-government communication interface, and a wealth of information about crops and animals, are integrated into this all-in-one platform.

Early intervention and decreased crop loss are made possible by the crop disease detection module's high-accuracy disease identification based on deep learning analysis of plant photos. To assist farmers in making well-informed planning decisions, the yield prediction module makes use of machine learning algorithms based on historical and environmental data. Users may get ready for weather conditions that could impact their fields with the help of real-time weather predictions. Expert assistance is always available thanks to the AI-powered chatbot's user-friendly responses to agricultural questions. Furthermore, by enabling users to voice concerns, access polls, and receive information straight from authorities, AgriNama improves contact between farmers and government representatives. With comprehensive information about crops, animals, and best practices, the app also functions as a teaching tool.AgriNama wants to transform the agricultural ecosystem and improve the farming community by fusing artificial intelligence, real-time data, and user-centric design.

1. Introduction:

Many economies still rely heavily on agriculture, particularly in rural areas where millions of people depend on it for their living. However, farmers frequently encounter difficulties including erratic weather patterns, a lack of timely information, ineffective government communication, and a lack of technical assistance for yield predictions and disease detection. We introduce Agri Nama, a web-based and mobile application that aims to improve and modernize the agricultural ecosystem in order to address these problems.

Weather Forecasting, Chatbot Support, Live Query, Crops and Cattles Information, GovLinks, AgriSurvey, Crop Disease Detection, and Yield Prediction are the eight main components that Agri Nama integrates into a single platform. The Weather Forecasting module helps farmers plan their agricultural operations by providing real-time weather data. Chatbot Assistance provides round-the-clock assistance for often requested questions, and LiveQuery allows farmers and agricultural officers to communicate directly. For improved decision-making, the Crops and Cattles Information module provides carefully selected information on cattle breeds and crop variations.

Additionally, GovLinks makes it possible for farmers to effectively access government programs and services. By enabling authorities to collect organized data from the field, AgriSurvey enhances resource allocation and policy choices. While the Yield Prediction module assists farmers in estimating production based on historical and environmental data,

International Journal of Scientific Research in Engineering and Management (IJSREM)
Volume: 09 Issue: 05 | May - 2025 SJIF Rating: 8.586 ISSN: 2582-3930

Crop Disease Detection uses state-of-the-art AI to assess plant health and provide treatment choices. These elements work together to create Agri Nama, a comprehensive tool that improves production, guarantees data-driven decision-making, and fortifies ties between farmers and the government.

2. Objectives:

Agri Nama's main goal is to provide a complete digital platform that uses technology-driven solutions to empower farmers and other agricultural stakeholders. The program seeks to offer 24/7 AI-powered chatbot help for prompt assistance, real-time weather forecasts for well-informed agricultural planning, and the ability to communicate live with experts via the LiveQuery feature. Additionally, it aims to provide comprehensive information on cattle and crops, facilitate field data gathering through AgriSurvey, and enhance access to government programs through GovLinks. Agri Nama also uses AI for precise yield forecasting and early agricultural disease identification, which eventually increases output, lowers risks, and promotes improved communication between farmers and government officials.

3. Problem Description:

Today's farmers deal with a variety of issues that impede the sustainability and productivity of agriculture. These include unpredictable weather patterns, delayed access to professional guidance, a lack of understanding about agricultural diseases, and a lack of familiarity with high-yield crop and animal breeds. Furthermore, there is a substantial communication breakdown between government agencies and farmers, which leads to delayed responses to field problems and underutilization of advantageous programs. Government officials' traditional approaches of gathering data are likewise laborious, prone to mistakes, and ineffective, which results in subpar policy choices. Farmers find it challenging to implement contemporary agricultural methods because they lack an integrated digital solution that integrates weather forecasts, expert engagement, disease detection, yield prediction, and access to government services. This creates a pressing need for a unified, intelligent platform that can address these issues holistically.

4. System Architecture Diagram:

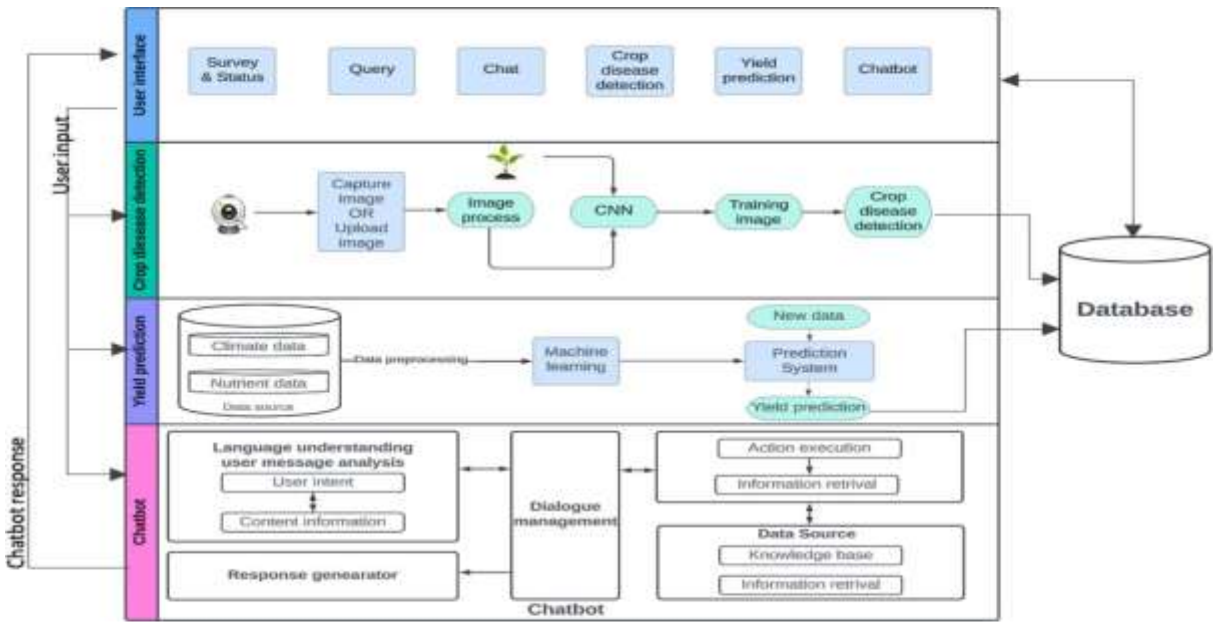


Fig. System Architecture Diagram

The system architecture follows a modular design to support crop disease detection, yield prediction, and user interaction. The **User Interface** provides access to key modules: Survey & Status, Query, Chat, Crop Disease Detection, Yield Prediction, and Chatbot. Users can upload crop images for **disease detection**, where a CNN model analyses them, and results are stored in a **central database**. The **yield prediction** module processes climate and nutrient data using machine learning models to forecast crop output. The **chatbot** interprets user intent, retrieves information, and provides automated responses. A unified **database** stores all inputs, results, and logs, enabling smooth data flow and system integration for efficient agricultural management.

5. Class diagram:

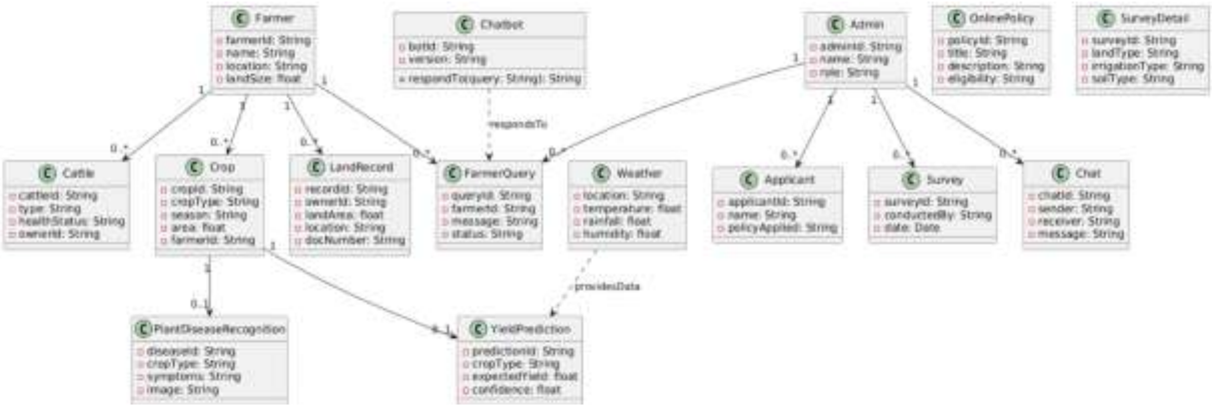


Fig-2: Class Diagram

At its core is the **Farmer** entity, connected to records of **Crops**, **Cattle**, and **Land**. Farmers can raise queries answered by a **Chatbot** and receive crop disease detection through **Plant Disease Recognition**. **Weather** data supports **Yield Prediction**. An **Admin** manages **Surveys**, **Applicants**, and **Online Policies**, with additional communication handled via the **Chat** entity. This system aims to streamline farming, policy access, and agricultural decision-making through smart integration.

6. Yield prediction (Random Forest):

The random forest algorithm is suitable for environmental factors like soil, climate, photoperiod, fertilization data, and water, as well as global and regional crop yields like potatoes, maize, and wheat. The outcomes were assessed using root mean square, Nash-Sutcliffe model efficiency, index of agreement, and observed vs. predicted plots, and they were compared with multiple linear regressions. In the US, the information is taken from a variety of sources. The findings demonstrated that Random Forest is a dynamic and successful technique for predicting agricultural output that is highly accurate, precise, user-friendly, and suitable for data processing. Using a variety of classification techniques, including support vector machines, Naive Bayes, and novel ensemble techniques like AdaSVM and AdaNaive, Narayanan Balakrishnan1 et al. [1] suggested an ensemble model to yield crops over a given time period. The suggested model was assessed based on accuracy and classification error, and AdaNaive was found to produce the best results for rice crops with an accuracy of 96.52. In order to map soil parameters and maize production at a small scale, Sami Khanal et al. [2] suggested an approach based on remotely sensed data and machine learning methods as Random Forest, Neural Network, Support Vector Machine, Gradient Boosting Model, and Cubist. Root mean square and accuracy are used to evaluate the model and remotely sense the data. Instead of using MLR (multiple linear regression) and RF (Random Forest) models to predict coffee yield for small farms, Louis Kouadioa et al. [3] proposed an artificial intelligence-based ELM model. Different machine learning models were compared to the ELM models. The author asserted that when it comes to feature extraction, ELM models outperform RF and MLR models.

Crop yield can be predicted using a variety of supervised machine learning algorithms, including support vector machines, random forests, ecision trees, polynomial regression, and linear regression.

6.1 Dataset: Data plays an important role in Machine Learning. To design and perform crop yield prediction system data is taken from various cities of Maharastra state. The collected data are previous data which is taken while survey in market. In that have different parameters including area of farm, city, which quality of pesticide used, what is best weather to work on farming, climate of area ,etc this are the data collected in survey

6.2 Proposed System: Random forest is a technique for classification and regression that is essentially supervised learning. The Random Forest method builds decision trees using several data samples, predicts the data from each subset, and then uses voting to choose the best system solution. To train the data, Random Forest employed the bagging technique. The bagging approach is essentially a combination of researching several models and improving the system's end output. In order to achieve high accuracy, we employed the Random Forest technique, which provides accuracy based on the model and the dataset's actual prediction outcome. In a random forest, a decision tree is created from a sample of data, and each tree provides a forecast from each family. The best option is chosen by voting, improving the model's accuracy. It provides the system with its best outcomes.

7. Weather Forecasting:

In the agricultural industry, where farming operations are heavily reliant on climatic conditions, weather forecasting is essential. Farmers may get real-time weather information directly on their web or mobile platforms by incorporating weather forecast APIs into agricultural applications.[4] This enhances productivity and reduces crop loss by enabling them to make well-informed decisions about irrigation, fertilization, insect management, and harvesting. An application and an outside weather service provider are connected by means of an API (Application Programming Interface)[5]. The application sends a request with parameters like location to the weather provider's server when an API call is made[6]. The server replies with current, precise meteorological information in a structured format, such as JSON[7]. Important meteorological information including temperature, humidity, wind speed, likelihood of rainfall, and general weather conditions are included in this data[8].

For instance, by retrieving data from a weather API such as OpenWeatherMap, the application may show the weather for a farmer's location when they launch it. The farmer may use this information to determine if it will rain soon, how hot or cold it will be, and whether it is safe to water the crops or apply pesticides[9]. With these characteristics, farmers may more effectively organize their operations and prevent needless losses brought on by unfavourable weather circumstances.

All things considered, utilizing weather prediction APIs in agriculture improves decision-making, encourages environmentally friendly farming methods, and provides farmers with the up-to-date knowledge they require for effective crop management.

How to utilize AgriNama : This is the detailed procedure for integrating weather forecasting via an API into your agricultural application:

1. Select a Weather API Supplier
For instance, WeatherStack, WeatherAPI, or OpenWeatherMap.
2. Obtain an API Key:
To authenticate your app, you must first register and then acquire a unique API key.
3. Call an API :
Provide the farm or village's location in a request to the API.
4. Get a JSON response
Weather data is sent back by the server.
5. Show the User the Forecast :

"Light Rain expected today in Nagpur, Temperature: 31.5°C, Humidity: 55%" is what your app can display.

8. Chatbot:

AI-powered chatbots and other digital technologies are helping to improve communication between farmers and technology in contemporary agriculture. Botpress, an open-source conversational AI platform, is one such potent tool that enables programmers to create intelligent, interactive chatbots that can help users through natural language interactions.

A Botpress chatbot may be included into a web or mobile application in the agriculture industry to respond to frequently requested queries by farmers.[10] These might include questions concerning market prices, government programs, fertilizer use, crop choices, disease detection, weather updates, and more. The objective is to develop a round-the-clock virtual assistant that gives farmers timely, accurate, and understandable information in the language of their choice.

Steps to work :

1. Configuring Bot press

Initially, Bot press is set up and installed. It can run in a cloud environment or on your own server. It offers a graphical user interface for organizing content and creating conversational flows.

2. Intent & Entity Recognition

When a farmer types or speaks a question like “Which fertilizer is best for wheat?” or “What’s the weather forecast for today?”, the Botpress Natural Language Understanding (NLU) engine processes the input. It detects the intent (e.g., ask_fertilizer_info, ask_weather) and extracts key entities (like crop name, date, location).

3. Personalized Flows and Q&A

You may address commonly requested questions using the QnA module or create your own discussion flows. For instance: The bot may provide symptoms, causes, and treatments in response to a user inquiry concerning "crop diseases. "When a user inquiry about the "PM Kisan Yojana," information on the program and how to apply may be displayed.

4. Integration of APIs

Additionally, Bot press is able to link to external APIs. For instance: Live weather updates may be obtained by calling a weather forecast API (such as OpenWeatherMap). Using crop names or symptoms, a crop disease detection API may get disease information[11].

5. Support for Multiple Languages

Farmers may communicate in Hindi, Marathi, or any other regional language thanks to Botpress's multilingual capability. For dynamic answers, you may either set up translations or utilize an external translation API.

6. Interface for Users

You may include the chatbot into your website or agricultural app. Farmers may quickly obtain responses by typing or clicking on specified alternatives. For improved accessibility, the interface may be altered to incorporate buttons, pictures, carousels, or even voice input.

An intelligent chatbot that serves as a virtual farming assistant may be provided by agricultural apps through the use of Botpress. It can provide prompt answers to questions about weather, illnesses, agriculture, and government assistance programs[12]. It becomes much more potent when combined with external APIs, providing farmers with real-time information in their own tongue. This increases farming's efficiency, accessibility, and data-drivenness.

9. Disease Detection:

Intro: The Gramineae family, which includes maize, ranks third in terms of overall production and cultivated area, after rice and wheat. Maize is a great feed for animals as well as for human use. Furthermore, it is a crucial raw element for the medical and light industries. Diseases are the main calamity impacting maize output, accounting for 6–10% of the yearly loss. Statistics show that there are around 80 maize diseases throughout the globe[13]. Sheath blight, rust, northern leaf blight, curcuma leaf spot, stem base rot, head smut, and other diseases are currently common and have detrimental effects.

The first stage in automatically identifying maize leaf diseases is the precise detection of lesions on the leaves. Nevertheless, it is challenging to detect maize leaf diseases using machine vision technologies. Because maize cultivars and growth phases differ greatly in terms of the form, size, texture, and posture of the leaves. The growth margins of maize leaves are quite asymmetrical, and the stem's hue is comparable to that of the leaves. In the real field setting, various maize organs and plants obstruct one another. Accurate automated identification of maize leaf diseases is made more difficult by the irregular and ever-changing nature of natural light. For improved generalization in various contexts, models that detect illnesses of the maize leaves must be created[14].

9.1 Dataset:

The Science Park on the west campus of China Agriculture University and the Vocational and Technical College of Inner Mongolia Agricultural University provided the data set used in this study. Figure 1 displays a total of 4428 photos, comprising 2735 normal images, 521 sheath blight images, 459 rust images, and 713 northern leaf blight images [15]. The photographs were taken at varied distances, in a variety of settings, and in a range of light and weather situations.

9.2 Analysis of Datasets:

The pre-processing of data presents a number of challenges, which also make it challenging to use picture recognition technologies for crop phenotypic analysis:

Some of the crops in the data set had many illnesses; the picture features of maize leaf diseases change with the severity of the disease; the shot will be fuzzy under windy conditions; and there are frequently overlapping plants in the image of maize in the densely planted area.
the distribution of the number of lesion characteristics of the three disease pictures in the dataset sample, according to additional statistical analysis of the dataset. Only a small percentage of each illness picture lacked clear focus characteristics, whereas around half had them.

9.3 Data Augmentation:

The data augmentation method is usually applied in the case of insufficient training samples. If the sample size of the training set is too small, the training of the network model will be insufficient, or the model will be overfitting. The data amplification method used in this paper includes two parts, simple amplification, and experimental amplification [16].

1. Simple amplification. We use the traditional image geometry transform, including image translation, rotation, cutting, and other operations. In this study, the method proposed by Alex et al. was explicitly adopted. First, images

were cut, the original image was cut into five subgraphs, and then the five subgraphs were flipped horizontally and vertically. Outsourcing frames counted the trimmed training set image to prevent the part of outsourcing frames from being cut out. In this way, each original image will eventually generate 15 extended images and the procedure of data augmentation is illustrated.

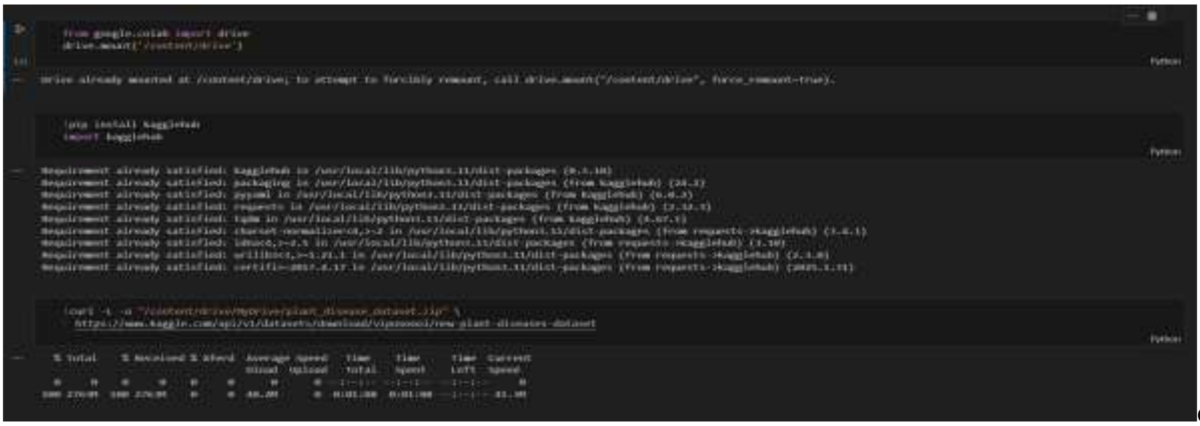
2. Image greying. In order to perform subsequent higher-level operations like picture segmentation, image recognition, and image analysis, grayscale processing is an essential preprocessing step. The photos used in this work are in RGB color mode, and each of the three RGB components is treated independently during the image processing procedure. However, RGB is only able to combine colors based on optical principles; it is unable to identify the morphological characteristics of the pictures used for illness identification. Grayscale processing can preserve the disease's visual characteristics, reducing the model's parameter count and thus speeding up the training and inferencing process[17]. In particular, the first phase involved graying the RGB three-channel pictures. was effectively cut down to a third of the original model. As a consequence, the model's training time was shortened.

3. Interfering leaf details are eliminated. Since numerous features in the maize leaf pictures will cause the model to malfunction due to the properties of the dataset utilized in this work, the data was preprocessed using erosion and dilation . The erosion operation is carried out first. Equation (1) illustrates the process of logical operation. It is possible to eliminate the leaf features by erosion, although this process would alter the lesion's properties. Consequently, the dilation procedure was required, and Equation (2) illustrates the logical operation process.

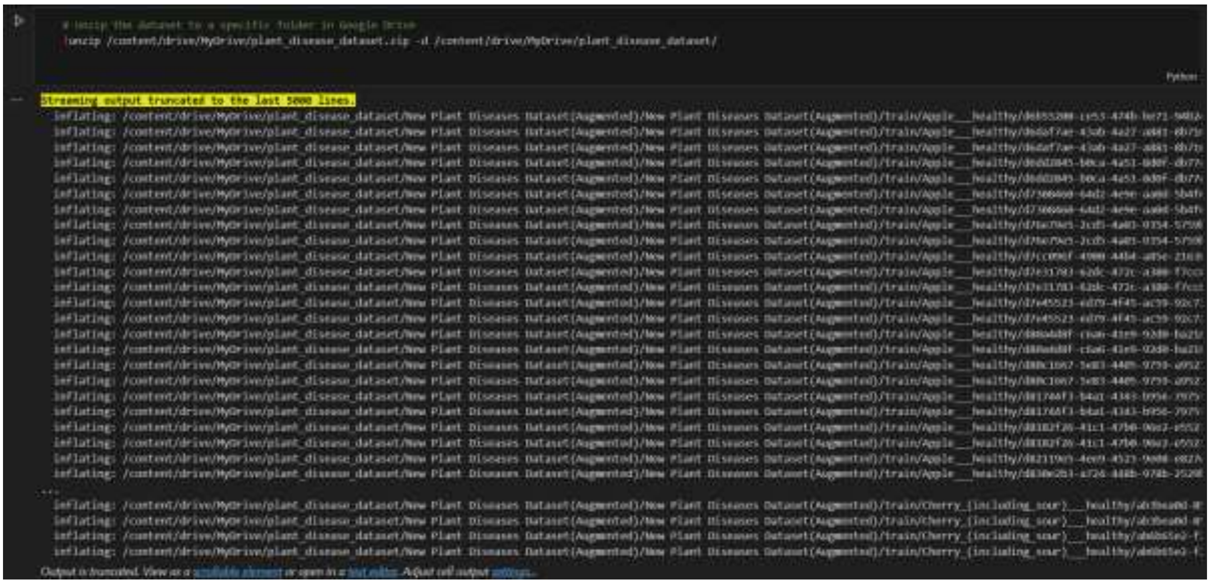
4. Mosaic and Snapmix. Currently, Snap mix and Mosaic are widely used data amplification techniques in deep learning research. These two techniques were applied in this work to further amplify the data using 59,778 training samples. The comparative experimental findings were evaluated using several amplification techniques. The categorization label stays the same when the Snapmix approach randomly removes portions of the sample and replaces them with a specific patch from other photos. Multiple images might be used simultaneously with the mosaic approach, and its most important.

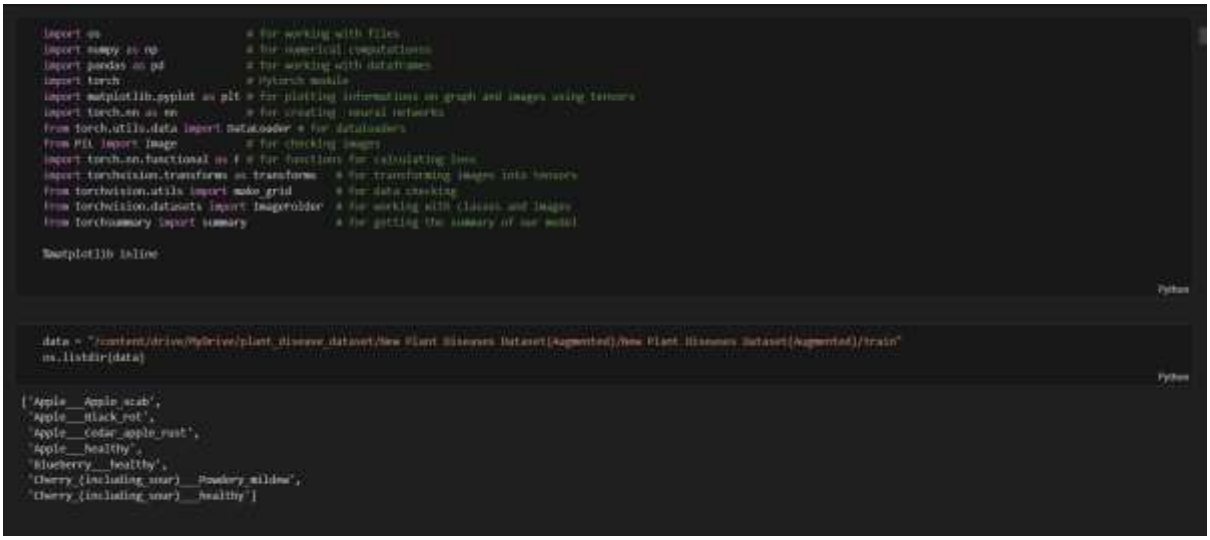
5. The creation of synthetic data is essential to model training in this research. Numerous solutions have been put up to address the issues of missing data. Assume that the training data is limited. In that instance, it is essential to provide three different types of data, such as three disease pictures of maize leaves afflicted with northern leaf blight, rust, and sheath blight. To create imagers based on the provided photos, a sampling technique based on Gaussian will be used[18]. The mean and standard deviation are the two necessary parameters.

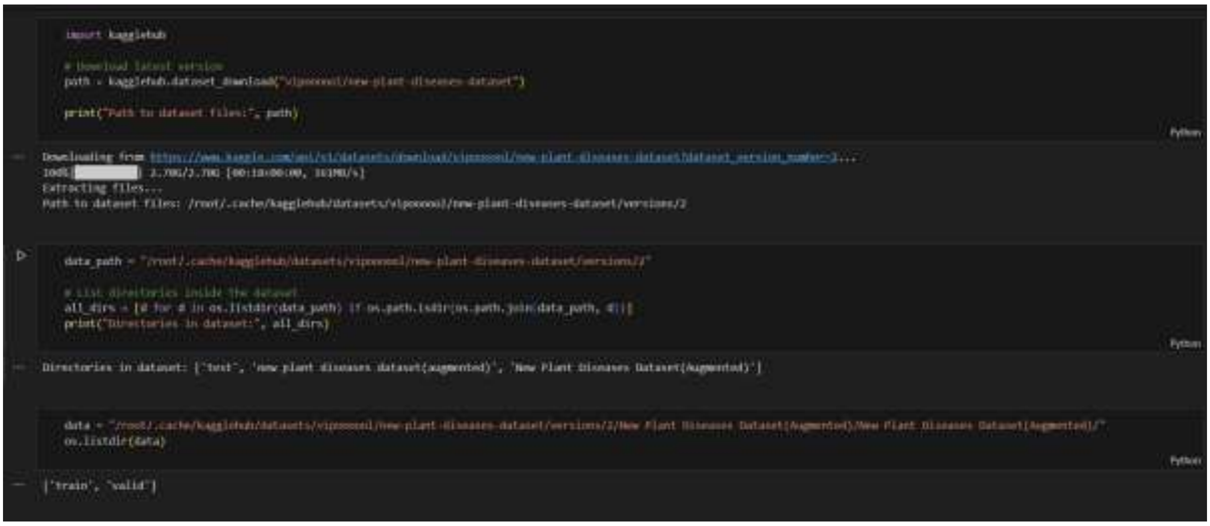
9.4 Plant Disease Detections Cod



e:


Unzip the dataset to a specific folder in Google Drive
!unzip /content/drive/MyDrive/plant_disease_dataset.zip -d /content/drive/MyDrive/plant_disease_dataset/
Streaming output truncated to the last 5000 lines.
inflating: /content/drive/MyDrive/plant_disease_dataset/New Plant Diseases Dataset(Augmented)/New Plant Diseases Dataset(Augmented)/train/Apple___healthy/
inflating: /content/drive/MyDrive/plant_disease_dataset/New Plant Diseases Dataset(Augmented)/New Plant Diseases Dataset(Augmented)/train/Cherry_(including_sour)___healthy/
Output is truncated. View as a scrollable element or open in a text editor. Adjust cell output settings...


import os # for working with files
import numpy as np # for numerical computations
import pandas as pd # for working with dataframes
import torch # Pytorch module
import matplotlib.pyplot as plt # for plotting informations on graph and images using tensors
import torch.nn as nn # for creating neural networks
from torch.utils.data import DataLoader # for dataloaders
from PIL import Image # for checking images
import torch.nn.functional as F # for functions for calculating loss
import torchvision.transforms as transforms # for transforming images into tensors
from torchvision.utils import make_grid # for data checking
from torchvision.datasets import ImageFolder # for working with classes and images
from torchsummary import summary # for getting the summary of our model
%matplotlib inline
data = "/content/drive/MyDrive/plant_disease_dataset/New Plant Diseases Dataset(Augmented)/New Plant Diseases Dataset(Augmented)/train"
os.listdir(data)
['Apple___Apple_scab',
'Apple___Black_rot',
'Apple___Cedar_apple_rust',
'Apple___healthy',
'Blueberry___healthy',
'Cherry_(including_sour)___Powdery_mildew',
'Cherry_(including_sour)___healthy']


import kagglehub
# Download latest version
path = kagglehub.dataset_download("vipoooool/new-plant-diseases-dataset")
print("Path to dataset files:", path)
Downloading from https://www.kaggle.com/api/v1/datasets/download/vipoooool/new-plant-diseases-dataset?dataset_version_number=2...
Extracting files...
Path to dataset files: /root/.cache/kagglehub/datasets/vipoooool/new-plant-diseases-dataset/versions/2
data_path = "/root/.cache/kagglehub/datasets/vipoooool/new-plant-diseases-dataset/versions/2"
# List directories inside the dataset
all_dirs = [d for d in os.listdir(data_path) if os.path.isdir(os.path.join(data_path, d))]
print("Directories in dataset:", all_dirs)
Directories in dataset: ['test', 'new plant diseases dataset(augmented)', 'New Plant Diseases Dataset(Augmented)']
data = "/root/.cache/kagglehub/datasets/vipoooool/new-plant-diseases-dataset/versions/2/New Plant Diseases Dataset(Augmented)/New Plant Diseases Dataset(Augmented)/"
os.listdir(data)
['train', 'valid']

International Journal of Scientific Research in Engineering and Management (IJSREM)
Volume: 09 Issue: 05 | May - 2025 SJIF Rating: 8.586 ISSN: 2582-3930

```python
train_dir = data + '/train/'
valid_dir = data + '/valid/'
test_dir = '/root/.cache/kagglehub/datasets/vipoooool/new-plant-diseases-dataset/versions/2' + '/test/'
diseases = os.listdir(train_dir)
```

```python
plants = []
NumberOfDiseases = 0
for plant in diseases:
    if plant.split('___')[0] not in plants:
        plants.append(plant.split('___')[0])
    if plant.split('___')[1] != 'healthy':
        NumberOfDiseases += 1
```

```python
# unique plants in the dataset
print(f"Unique Plants are: \n{plants}")
```

```
Unique Plants are:
['Potato', 'Raspberry', 'Soybean', 'Strawberry', 'Apple', 'Tomato', 'Cherry_(including_sour)', 'Peach', 'Corn_(maize)', 'Squash', 'Grape', 'Pepper,_bell', 'Blueberry', 'Orange']
```

```python
print("Number of plants: {}".format(len(plants)))
```

```
Number of plants: 14
```

```python
print("Number of diseases: {}".format(NumberOfDiseases))
```

```python
train_df = {}
for dis in diseases:
    train_df[dis] = len(os.listdir(train_dir + '/' + dis))
# train_df = pd.DataFrame(list(train_df.items()), columns=['Disease', 'Count'])
train_df
```

```
{'Potato___healthy': 1824,
 'Raspberry___healthy': 1781,
 'Soybean___healthy': 2022,
 'Potato___Late_blight': 1939,
 'Strawberry___Leaf_scorch': 1774,
 'Apple___Cedar_apple_rust': 1760,
 'Potato___Early_blight': 1939,
 'Tomato___Leaf_Mold': 1882,
 'Cherry_(including_sour)___Powdery_mildew': 1683,
 'Peach___Bacterial_spot': 1838,
 'Tomato___Tomato_mosaic_virus': 1790,
 'Cherry_(including_sour)___healthy': 1826,
 'Peach___healthy': 1728,
 'Tomato___Spider_mites Two-spotted_spider_mite': 1741,
 'Apple___Black_rot': 1987,
 'Corn_(maize)___Common_rust_': 1907,
 'Apple___Apple_scab': 2016,
 'Corn_(maize)___healthy': 1859,
 'Squash___Powdery_mildew': 1736,
 'Grape___Leaf_blight_(Isariopsis_Leaf_Spot)': 1722,
 'Corn_(maize)___Northern_Leaf_Blight': 1908,
 'Tomato___Septoria_leaf_spot': 1745,
 'Grape___healthy': 1692,
 'Pepper,_bell___Bacterial_spot': 1913,
 'Corn_(maize)___Cercospora_leaf_spot Gray_leaf_spot': 1642,
...
 'Grape___Esca_(Black_Measles)': 1920,
 'Strawberry___healthy': 1824,
 'Orange___Haunglongbing_(Citrus_greening)': 2010,
```

```python
total = 0
for _,value in train_df.items():
  total +=value
total
```

```
70295
```

```python
train = ImageFolder(train_dir, transform=transforms.ToTensor())
valid = ImageFolder(valid_dir, transform=transforms.ToTensor())
```

```python
img, label = train[0]
img.shape, label
```

```
(torch.Size([3, 256, 256]), 0)
```

```python
len(train.classes),len(valid.classes)
```

```
(38, 38)
```

```python
def imshow(image):
    img, label = image

    # Convert tensor image to numpy array
    img = img.permute(1, 2, 0).numpy()  # Change dimensions from (C, H, W) to (H, W, C)

    # Unnormalize the image if normalization was applied
    mean = np.array([0.485, 0.456, 0.406])
    std = np.array([0.229, 0.224, 0.225])
    img = img * std + mean
    img = np.clip(img, 0, 1)

    # Display the image
    plt.imshow(img)
    plt.title(train.classes[label])
    plt.axis('off')
    plt.show()

# Show the first image from the dataset
imshow(train[0])
```


```python
#setting the batch size
batch_size = 32
```

```python
# DataLoaders for training and validation
train_dl = DataLoader(train, batch_size, shuffle=True, num_workers=2, pin_memory=True)
valid_dl = DataLoader(valid, batch_size, num_workers=2, pin_memory=True)
```

```python
# helper function to show a batch of training instances
def show_batch(data):
    for images, labels in data:
        fig, ax = plt.subplots(figsize=(30, 30))
        ax.set_xticks([]); ax.set_yticks([])
        ax.imshow(make_grid(images, nrow=8).permute(1, 2, 0))
        break
show_batch(train_dl)
```

```python
# for moving data into GPU (if available)
def get_default_device():
    """Pick GPU if available, else CPU"""
    if torch.cuda.is_available:
        return torch.device("cuda")
    else:
        return torch.device("cpu")

# for moving data to device (CPU or GPU)
def to_device(data, device):
    """Move tensor(s) to chosen device"""
    if isinstance(data, (list,tuple)):
        return [to_device(x, device) for x in data]
    return data.to(device, non_blocking=True)

# for loading in the device (GPU if available else CPU)
class DeviceDataLoader():
    """Wrap a dataloader to move data to a device"""
    def __init__(self, dl, device):
        self.dl = dl
        self.device = device

    def __iter__(self):
        """Yield a batch of data after moving it to device"""
        for b in self.dl:
            yield to_device(b, self.device)

    def __len__(self):
        """Number of batches"""
        return len(self.dl)
```

```python
device = get_default_device()
```

International Journal of Scientific Research in Engineering and Management (IJSREM)
Volume: 09 Issue: 05 | May - 2025 SJIF Rating: 8.586 ISSN: 2582-3930

```
device(type='cuda')
```

```python
# Moving data into GPU
train_dl = DeviceDataLoader(train_dl, device)
valid_dl = DeviceDataLoader(valid_dl, device)
```

Python

```python
class SimpleResidualBlock(nn.Module):
    def __init__(self):
        super().__init__()
        self.conv1 = nn.Conv2d(in_channels=3, out_channels=3, kernel_size=3, stride=1, padding=1)
        self.relu1 = nn.ReLU()
        self.conv2 = nn.Conv2d(in_channels=3, out_channels=3, kernel_size=3, stride=1, padding=1)
        self.relu2 = nn.ReLU()

    def forward(self, x):
        out = self.conv1(x)
        out = self.relu1(out)
        out = self.conv2(out)
        return self.relu2(out) + x # ReLU can be applied before or after adding the input
```

Python

```python
# for calculating the accuracy
def accuracy(outputs, labels):
    _, preds = torch.max(outputs, dim=1)
    return torch.tensor(torch.sum(preds == labels).item() / len(preds))


# base class for the model
class ImageClassificationBase(nn.Module):

    def training_step(self, batch):
        images, labels = batch
        out = self(images)                  # Generate predictions
        loss = F.cross_entropy(out, labels) # Calculate loss
        return loss

    def validation_step(self, batch):
        images, labels = batch
        out = self(images)                   # Generate prediction
        loss = F.cross_entropy(out, labels)  # Calculate loss
        acc = accuracy(out, labels)          # Calculate accuracy
        return {"val_loss": loss.detach(), "val_accuracy": acc}

    def validation_epoch_end(self, outputs):
        batch_losses = [x["val_loss"] for x in outputs]
        batch_accuracy = [x["val_accuracy"] for x in outputs]
        epoch_loss = torch.stack(batch_losses).mean()       # Combine loss
        epoch_accuracy = torch.stack(batch_accuracy).mean()
        return {"val_loss": epoch_loss, "val_accuracy": epoch_accuracy} # Combine accuracies

    def epoch_end(self, epoch, result):
        print("Epoch [{}], last_lr: {:.5f}, train_loss: {:.4f}, val_loss: {:.4f}, val_acc: {:.4f}".format(
            epoch, result['lrs'][-1], result['train_loss'], result['val_loss'], result['val_accuracy']))
```

```python
# convolution block with BatchNormalization
def ConvBlock(in_channels, out_channels, pool=False):
    layers = [nn.Conv2d(in_channels, out_channels, kernel_size=3, padding=1),
             nn.BatchNorm2d(out_channels),
             nn.ReLU(inplace=True)]
    if pool:
        layers.append(nn.MaxPool2d(4))
    return nn.Sequential(*layers)


# resnet architecture
class ResNet9(ImageClassificationBase):
    def __init__(self, in_channels, num_diseases):
        super().__init__()

        self.conv1 = ConvBlock(in_channels, 64)
        self.conv2 = ConvBlock(64, 128, pool=True) # out_dim : 128 x 64 x 64
        self.res1 = nn.Sequential(ConvBlock(128, 128), ConvBlock(128, 128))

        self.conv3 = ConvBlock(128, 256, pool=True) # out_dim : 256 x 16 x 16
        self.conv4 = ConvBlock(256, 512, pool=True) # out_dim : 512 x 4 x 44
        self.res2 = nn.Sequential(ConvBlock(512, 512), ConvBlock(512, 512))

        self.classifier = nn.Sequential(nn.MaxPool2d(4),
                                       nn.Flatten(),
                                       nn.Linear(512, num_diseases))

    def forward(self, xb): # xb is the loaded batch
        out = self.conv1(xb)
        out = self.conv2(out)
        out = self.res1(out) + out
        out = self.conv3(out)
        out = self.conv4(out)
        out = self.res2(out) + out
        out = self.classifier(out)
        return out
```

```python
model = to_device(ResNet9(3, len(train.classes)), device)
model
```

Python

```
ResNet9(
  (conv1): Sequential(
    (0): Conv2d(3, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (2): ReLU(inplace=True)
  )
  (conv2): Sequential(
    (0): Conv2d(64, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (2): ReLU(inplace=True)
    (3): MaxPool2d(kernel_size=4, stride=4, padding=0, dilation=1, ceil_mode=False)
  )
  (res1): Sequential(
    (0): Sequential(
      (0): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
      (1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (2): ReLU(inplace=True)
    )
    (1): Sequential(
      (0): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
      (1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (2): ReLU(inplace=True)
    )
  )
  (conv3): Sequential(
...
    (0): MaxPool2d(kernel_size=4, stride=4, padding=0, dilation=1, ceil_mode=False)
    (1): Flatten(start_dim=1, end_dim=-1)
    (2): Linear(in_features=512, out_features=38, bias=True)
  )
)
```

Output is truncated. View as a scrollable element or open in a text editor. Adjust cell output settings...

```python
# getting summary of the model
INPUT_SHAPE = (3, 256, 256)
print(summary(model.cuda(), (INPUT_SHAPE)))
```

Python

```
----------------------------------------------------------------
        Layer (type)               Output Shape         Param #
================================================================
            Conv2d-1         [-1, 64, 256, 256]           1,792
       BatchNorm2d-2         [-1, 64, 256, 256]             128
              ReLU-3         [-1, 64, 256, 256]               0
            Conv2d-4        [-1, 128, 256, 256]          73,856
       BatchNorm2d-5        [-1, 128, 256, 256]             256
              ReLU-6        [-1, 128, 256, 256]               0
         MaxPool2d-7          [-1, 128, 64, 64]               0
            Conv2d-8          [-1, 128, 64, 64]         147,584
       BatchNorm2d-9          [-1, 128, 64, 64]             256
             ReLU-10          [-1, 128, 64, 64]               0
           Conv2d-11          [-1, 128, 64, 64]         147,584
      BatchNorm2d-12          [-1, 128, 64, 64]             256
             ReLU-13          [-1, 128, 64, 64]               0
           Conv2d-14          [-1, 256, 64, 64]         295,168
      BatchNorm2d-15          [-1, 256, 64, 64]             512
             ReLU-16          [-1, 256, 64, 64]               0
        MaxPool2d-17          [-1, 256, 16, 16]               0
           Conv2d-18          [-1, 512, 16, 16]       1,180,160
      BatchNorm2d-19          [-1, 512, 16, 16]           1,024
             ReLU-20          [-1, 512, 16, 16]               0
        MaxPool2d-21            [-1, 512, 4, 4]               0
           Conv2d-22            [-1, 512, 4, 4]       2,359,808
...
Params size (MB): 25.18
Estimated Total Size (MB): 360.83
----------------------------------------------------------------
None
```

```python
@torch.no_grad()
def evaluate(model, val_loader):
    model.eval()
    outputs = [model.validation_step(batch) for batch in val_loader]
    return model.validation_epoch_end(outputs)


def get_lr(optimizer):
    for param_group in optimizer.param_groups:
        return param_group['lr']


def fit_OneCycle(epochs, max_lr, model, train_loader, val_loader, weight_decay=0,
                grad_clip=None, opt_func=torch.optim.SGD):
    torch.cuda.empty_cache()
    history = []

    optimizer = opt_func(model.parameters(), max_lr, weight_decay=weight_decay)
    # scheduler for one cycle learning rate
    sched = torch.optim.lr_scheduler.OneCycleLR(optimizer, max_lr, epochs=epochs, steps_per_epoch=len(train_loader))


    for epoch in range(epochs):
        # Training
        model.train()
        train_losses = []
        lrs = []
        for batch in train_loader:
            loss = model.training_step(batch)
            train_losses.append(loss)
            loss.backward()

            # gradient clipping
            if grad_clip:
                nn.utils.clip_grad_value_(model.parameters(), grad_clip)
```

International Journal of Scientific Research in Engineering and Management (IJSREM)
Volume: 09 Issue: 05 | May - 2025 SJIF Rating: 8.586 ISSN: 2582-3930

```python
            optimizer.step()
            optimizer.zero_grad()

            # recording and updating learning rates
            lrs.append(get_lr(optimizer))
            sched.step()

        # validation
        result = evaluate(model, val_loader)
        result['train_loss'] = torch.stack(train_losses).mean().item()
        result['lrs'] = lrs
        model.epoch_end(epoch, result)
        history.append(result)

    return history
```

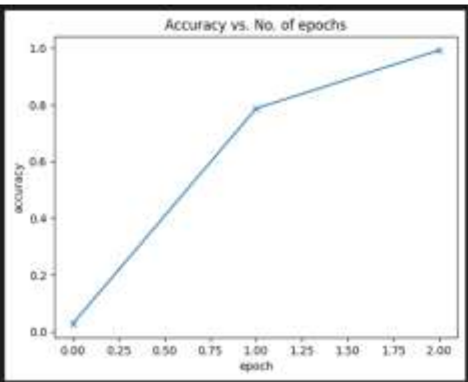
```python
%%time
history = [evaluate(model, valid_dl)]
history
```

```
CPU times: user 59 s, sys: 5.76 s, total: 1min 4s
Wall time: 1min 10s

[{'val_loss': tensor(3.6378, device='cuda:0'), 'val_accuracy': tensor(0.0302)}]
```

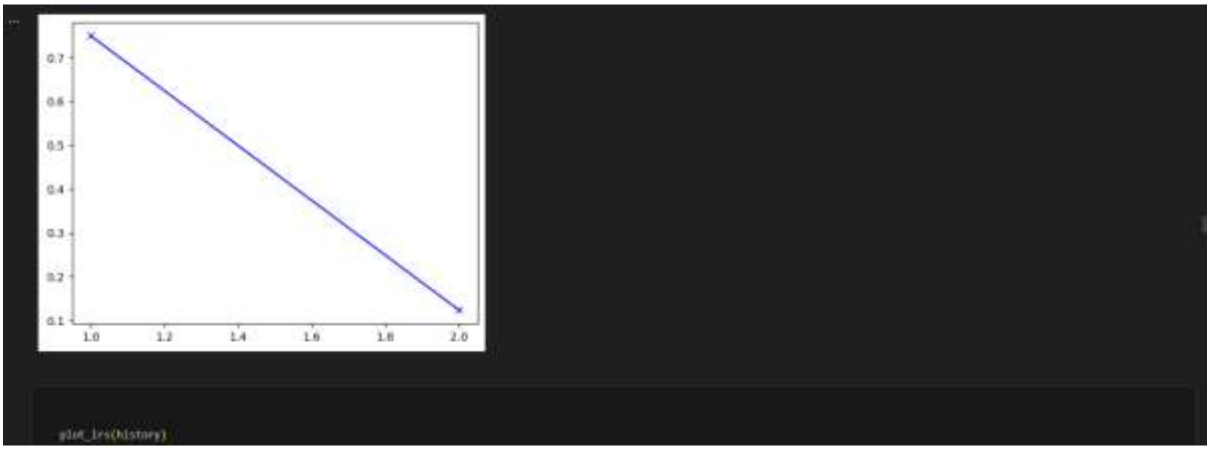
```python
epochs = 2
max_lr = 0.01
grad_clip = 0.1
weight_decay = 1e-4
opt_func = torch.optim.Adam
```

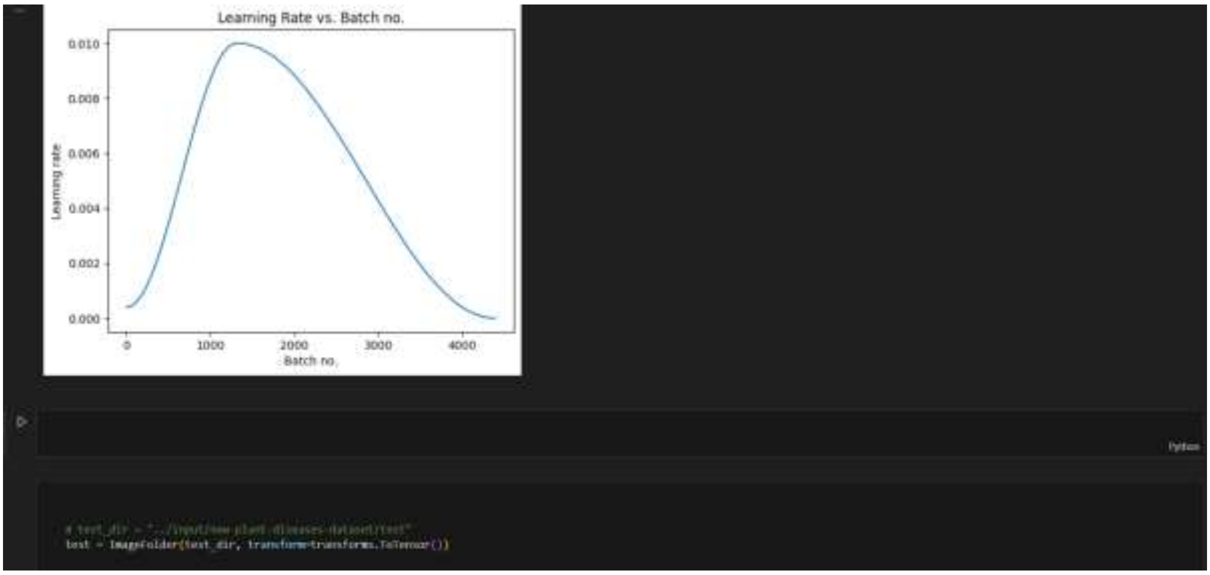
```python
def plot_accuracies(history):
    accuracies = [x['val_accuracy'] for x in history]
    plt.plot(accuracies, '-x')
    plt.xlabel('epoch')
    plt.ylabel('accuracy')
    plt.title('Accuracy vs. No. of epochs');

def plot_losses(history):
    train_losses = [x.get('train_loss') for x in history]
    val_losses = [x['val_loss'] for x in history]
    plt.plot(train_losses, '-bx')
    plt.plot(val_losses, '-rx')
    plt.xlabel('epoch')
    plt.ylabel('loss')
    plt.legend(['Training', 'Validation'])
    plt.title('Loss vs. No. of epochs');

def plot_lrs(history):
    lrs = np.concatenate([x.get('lrs', []) for x in history])
    plt.plot(lrs)
    plt.xlabel('Batch no.')
    plt.ylabel('Learning rate')
    plt.title('Learning Rate vs. Batch no.');
```

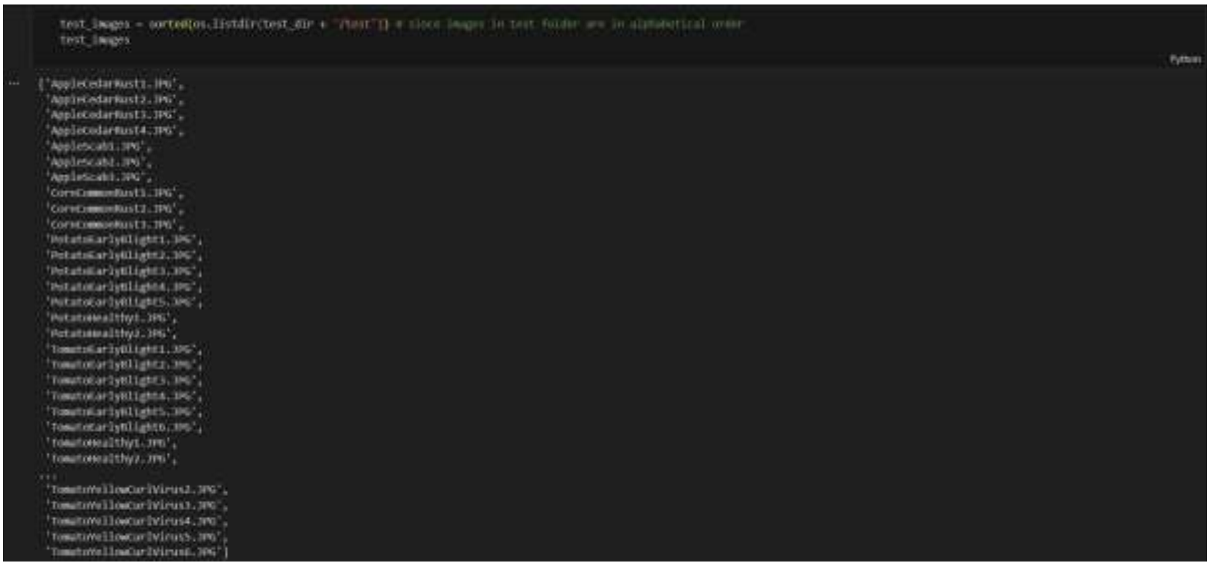
```python
plot_accuracies(history)
```




```python
plot_losses(history)
```


plot_lrs(history)


Learning Rate vs. Batch no.
Learning rate
Batch no.
Python


Python

Volume: 09 Issue: 05 | May - 2025 SJIF Rating: 8.586 ISSN: 2582-3930

```python
def predict_image(img, model):
    """Converts image to array and return the predicted class
       with highest probability"""
    # Convert to a batch of 1
    xb = to_device(img.unsqueeze(0), device)
    # Get predictions from model
    yb = model(xb)
    # Pick index with highest probability
    _, preds  = torch.max(yb, dim=1)
    # Retrieve the class label

    return train.classes[preds[0].item()]
```

Python

```python
# getting all predictions (actual label vs predicted)
for i, (img, label) in enumerate(test):
  print('Label:', test_images[i], ', Predicted:', predict_image(img, model))
```

Python

```python
# saving to the kaggle working directory
PATH = '/content/drive/MyDrive/plant_disease_resnet9.pth'
torch.save(model.state_dict(), PATH)
```

Python

```python
import torch
model_path = '/content/drive/MyDrive/plant_disease_resnet9.pth'
```

Python

```python
model = ResNet9(3, 38)
```

Python

```python
model.load_state_dict(torch.load(model_path, map_location=torch.device('cpu')))
```

Python

```
<All keys matched successfully>
```

```python
def predict_image(image_path, model, device='cpu'):
    # Define the same transform used during training/validation
    transform = transforms.Compose([
        # transforms.Resize((256, 256)),  # Resize to 256x256
        transforms.ToTensor()
    ])

    # Load the image and apply the transformation
    image = Image.open(image_path).convert('RGB')
    image_tensor = transform(image).unsqueeze(0)  # Add batch dimension

    # Move the model and tensor to the correct device (CPU or GPU)
    # model.to(device)
    # image_tensor = image_tensor.to(device)

    # Set the model to evaluation mode
    # model.eval()

    # Run inference without gradient tracking
    with torch.no_grad():
        output = model(image_tensor)

    # Get predicted class (if classification task)
    _, predicted_class = torch.max(output, 1)

    return predicted_class.item()
```

Python

```python
class_names[predict_image(image_path,model)]
```

Python

```
'Apple___Cedar_apple_rust'
```

International Journal of Scientific Research in Engineering and Management (IJSREM)
Volume: 09 Issue: 05 | May - 2025 SJIF Rating: 8.586 ISSN: 2582-3930

```python
image_path = '/content/healthy_tomato.jpeg'
```

```python
image_path = '/content/drive/MyDrive/plant_disease_dataset/New Plant Diseases Dataset(Augmented)/New Plant Diseases Dataset(Augmented)/train/Apple___Black_rot/
```

```python
model = model = models.mobilenet_v2(pretrained=False)
model.classifier[1] = nn.Linear(model.last_channel, 38)
```

```python
path = '/content/drive/MyDrive/plant_disease_mobilenet.pth'
```

```python
model.load_state_dict(torch.load(path,map_location=torch.device('cpu')))
```

```
<All keys matched successfully>
```

```python
class_names[predict_image(image_path,model)]
```

```python
def predict_image(image_path, model, device='cpu'):
    # Define the same transform used during training/validation
    transform = transforms.Compose([
        transforms.Resize((224, 224)),
        transforms.ToTensor(),
        transforms.Normalize([0.485, 0.456, 0.406], [0.229, 0.224, 0.225])
    ])

    # Load the image and apply the transformation
    image = Image.open(image_path).convert('RGB')
    image_tensor = transform(image).unsqueeze(0)  # Add batch dimension

    # Move the model and tensor to the correct device (CPU or GPU)
    # model.to(device)
    # image_tensor = image_tensor.to(device)

    # Set the model to evaluation mode
    # model.eval()

    # Run inference without gradient tracking
    with torch.no_grad():
        output = model(image_tensor)

    # Get predicted class (if classification task)
    _, predicted_class = torch.max(output, 1)

    return predicted_class.item()
```

```python
import torch
import torch.nn as nn
import torch.optim as optim
from torchvision import datasets, transforms, models
from torch.utils.data import DataLoader
import matplotlib.pyplot as plt
import copy
import time
```

```python
# Dataset paths
data_dir = "/root/.cache/kagglehub/datasets/vipoooool/new-plant-diseases-dataset/versions/2/New Plant Diseases Dataset(Augmented)/New Plant Diseases Dataset(Augmented)/"
train_dir = data_dir + "train"
val_dir = data_dir + "valid"

# Image transformations
data_transforms = {
    'train': transforms.Compose([
        transforms.Resize((224, 224)),
        transforms.RandomHorizontalFlip(),
        transforms.RandomRotation(10),
        transforms.ToTensor(),
        transforms.Normalize([0.485, 0.456, 0.406], [0.229, 0.224, 0.225])  # ImageNet normalization
    ]),
    'val': transforms.Compose([
        transforms.Resize((224, 224)),
        transforms.ToTensor(),
        transforms.Normalize([0.485, 0.456, 0.406], [0.229, 0.224, 0.225])  # ImageNet normalization
    ]),
}

# Load datasets
train_dataset = datasets.ImageFolder(train_dir, transform=data_transforms['train'])
```

```python
# Create data loaders
train_loader = DataLoader(train_dataset, batch_size=32, shuffle=True, num_workers=2, pin_memory=True)
val_loader = DataLoader(val_dataset, batch_size=32, shuffle=False, num_workers=2, pin_memory=True)

# Class names and number of classes
class_names = train_dataset.classes
num_classes = len(class_names)
print(f"Number of classes: {num_classes}")
```

```
Number of classes: 38
```

```python
class_names = ['Apple___Apple_scab',
 'Apple___Black_rot',
 'Apple___Cedar_apple_rust',
 'Apple___healthy',
 'Blueberry___healthy',
 'Cherry_(including_sour)___Powdery_mildew',
 'Cherry_(including_sour)___healthy',
 'Corn_(maize)___Cercospora_leaf_spot Gray_leaf_spot',
 'Corn_(maize)___Common_rust_',
 'Corn_(maize)___Northern_Leaf_Blight',
 'Corn_(maize)___healthy',
 'Grape___Black_rot',
 'Grape___Esca_(Black_Measles)',
 'Grape___Leaf_blight_(Isariopsis_Leaf_Spot)',
 'Grape___healthy',
 'Orange___Haunglongbing_(Citrus_greening)',
 'Peach___Bacterial_spot',
 'Peach___healthy',
 'Pepper,_bell___Bacterial_spot',
 'Pepper,_bell___healthy',
 'Potato___Early_blight',
 'Potato___Late_blight',
 'Potato___healthy',
```

```python
# Load pretrained MobileNetV2
model = models.mobilenet_v2(pretrained=True)

# Freeze all layers except classifier
for param in model.features.parameters():
    param.requires_grad = False

# Modify classifier for plant disease classification
model.classifier[1] = nn.Linear(model.last_channel, num_classes)

# Move model to GPU if available
device = torch.device("cuda:0" if torch.cuda.is_available() else "cpu")
model = model.to(device)
print(f"Model running on: {device}")
```

```
/usr/local/lib/python3.11/dist-packages/torchvision/models/_utils.py:208: UserWarning: The parameter 'pretrained' is deprecated since 0.13 and may be removed in the future, please use
  warnings.warn(
/usr/local/lib/python3.11/dist-packages/torchvision/models/_utils.py:223: UserWarning: Arguments other than a weight enum or `None` for 'weights' are deprecated since 0.13 and may be r
  warnings.warn(msg)
Downloading: "https://download.pytorch.org/models/mobilenet_v2-b0353104.pth" to /root/.cache/torch/hub/checkpoints/mobilenet_v2-b0353104.pth
100%|██████████| 13.6M/13.6M [00:00<00:00, 100MB/s]
Model running on: cuda:0
```

```python
# Training and evaluation function
def train_model(model, train_loader, val_loader, criterion, optimizer, num_epochs=5):
    start_time = time.time()
    best_model_wts = copy.deepcopy(model.state_dict())
    best_acc = 0.0
```

```python
    for epoch in range(num_epochs):
        print(f'Epoch {epoch+1}/{num_epochs}')
        print('-' * 20)

        for phase in ['train', 'val']:
            if phase == 'train':
                model.train()
                data_loader = train_loader
            else:
                model.eval()
                data_loader = val_loader

            running_loss = 0.0
            running_corrects = 0

            for inputs, labels in data_loader:
                inputs, labels = inputs.to(device), labels.to(device)

                with torch.set_grad_enabled(phase == 'train'):
                    outputs = model(inputs)
                    _, preds = torch.max(outputs, 1)
                    loss = criterion(outputs, labels)

                    if phase == 'train':
                        optimizer.zero_grad()
                        loss.backward()
                        optimizer.step()

                running_loss += loss.item() * inputs.size(0)
                running_corrects += torch.sum(preds == labels.data)

            epoch_loss = running_loss / len(data_loader.dataset)
            epoch_acc = running_corrects.double() / len(data_loader.dataset)

            print(f'{phase.capitalize()} Loss: {epoch_loss:.4f} Acc: {epoch_acc:.4f}')
```

International Journal of Scientific Research in Engineering and Management (IJSREM)
Volume: 09 Issue: 05 | May - 2025 SJIF Rating: 8.586 ISSN: 2582-3930

```python
                print(f'{phase.capitalize()} Loss: {epoch_loss:.4f} Acc: {epoch_acc:.4f}')

                # Save best model
                if phase == 'val' and epoch_acc > best_acc:
                    best_acc = epoch_acc
                    best_model_wts = copy.deepcopy(model.state_dict())

            print()

        time_elapsed = time.time() - start_time
        print(f'Training complete in {time_elapsed // 60:.0f}m {time_elapsed % 60:.0f}s')
        print(f'Best Val Accuracy: {best_acc:.4f}')

        # Load best model weights
        model.load_state_dict(best_model_wts)
        return model
```

Python

```python
# Define loss and optimizer
criterion = nn.CrossEntropyLoss()
optimizer = optim.Adam(model.classifier.parameters(), lr=0.001)

# Train only the classifier head (first stage)
model = train_model(model, train_loader, val_loader, criterion, optimizer, num_epochs=5)
```

Python

```
Epoch 1/5
--------------------
Train Loss: 0.4183 Acc: 0.8953
Val Loss: 0.1673 Acc: 0.9490

Epoch 2/5
--------------------
```

```
Epoch 1/5
--------------------
Train Loss: 0.4183 Acc: 0.8953
Val Loss: 0.1673 Acc: 0.9490

Epoch 2/5
--------------------
Train Loss: 0.2083 Acc: 0.9342
Val Loss: 0.1500 Acc: 0.9495

Epoch 3/5
--------------------
Train Loss: 0.1785 Acc: 0.9410
Val Loss: 0.1225 Acc: 0.9603

Epoch 4/5
--------------------
Train Loss: 0.1760 Acc: 0.9415
Val Loss: 0.1390 Acc: 0.9571

Epoch 5/5
--------------------
Train Loss: 0.1700 Acc: 0.9427
Val Loss: 0.1116 Acc: 0.9649

Training complete in 23m 8s
Best Val Accuracy: 0.9649
```

```python
# Unfreeze entire model for fine-tuning
for param in model.features.parameters():
    param.requires_grad = True

# Define new optimizer with very low learning rate for fine-tuning
optimizer = optim.Adam(model.parameters(), lr=1e-5)
```

```
Epoch 1/2
--------------------
Train Loss: 0.0705 Acc: 0.9737
Val Loss: 0.0410 Acc: 0.9809

Epoch 2/2
--------------------
Train Loss: 0.0634 Acc: 0.9853
Val Loss: 0.0270 Acc: 0.9916

Training complete in 13m 29s
Best Val Accuracy: 0.9916
```

```python
# Save the fine-tuned model
model_path = "/content/drive/MyDrive/plant_disease_mobilenet.pth"
torch.save(model.state_dict(), model_path)
print(f"Model saved successfully at {model_path}!")
```

Python

```
Model saved successfully at /content/drive/MyDrive/plant_disease_mobilenet.pth!
```

```python
# Define inference function
def predict_image(image_path, model, class_names):
    # Load image
    image = datasets.folder.default_loader(image_path)
    transform = transforms.Compose([
        transforms.Resize((224, 224)),
        transforms.ToTensor(),
        transforms.Normalize([0.485, 0.456, 0.406], [0.229, 0.224, 0.225])
    ])
    # image_tensor = transform(image).unsqueeze(0).to(device)
    image_tensor = transform(image).unsqueeze(0)
```

```python
    model.eval()
    with torch.no_grad():
        output = model(image_tensor)
        _, pred = torch.max(output, 1)
        class_name = class_names[pred.item()]
        print(pred.item())
    return class_name
```

```python
predict_image('/content/tomato_curl.png',model,class_names)
```

```
29
'Tomato__Early_blight'
```

```python
# getting all predictions (actual label vs predicted)
for i, (img, label) in enumerate(test):
    print('Label:', test_images[i], ', Predicted:', predict_image(img, model))
```

```
Label: AppleCedarRust1.JPG , Predicted: Apple___Cedar_apple_rust
Label: AppleCedarRust2.JPG , Predicted: Apple___Cedar_apple_rust
Label: AppleCedarRust3.JPG , Predicted: Apple___Cedar_apple_rust
Label: AppleCedarRust4.JPG , Predicted: Tomato___healthy
Label: AppleScab1.JPG , Predicted: Tomato___healthy
```

```python
# Example image path
sample_image_path = "/content/drive/MyDrive/plant_disease_dataset//class_name/sample_image.jpg"

# Load trained model for inference
model.load_state_dict(torch.load(model_path))
model.eval()

# Run prediction
predicted_class = predict_image(sample_image_path, model, class_names)
print(f'Predicted Class: {predicted_class}')
```

```python
model_path = '/content/drive/MyDrive/plant_disease_mobilenet.pth'
```

```python
import torch
import torchvision.models as models

# Load your trained PyTorch model

model.load_state_dict(torch.load(model_path, map_location=torch.device('cpu')))  # Move to CPU
model.eval()

# Move model to CPU
model.to('cpu')

# Dummy input for tracing (on CPU)
dummy_input = torch.randn(1, 3, 224, 224, device='cpu')

# Export to ONNX
torch.onnx.export(model, dummy_input, "/content/drive/MyDrive/plant_disease_mobilenet.onnx", opset_version=12)
```

```python
import onnxruntime as ort
print("ONNX Runtime version:", ort.__version__)
```

```
ONNX Runtime version: 1.21.0
```

```python
# ✅ Load ONNX model
onnx_model_path = "/content/drive/MyDrive/plant_disease_mobilenet.onnx"
session = ort.InferenceSession(onnx_model_path)

# ✅ Load input image (replace with your image path)
from PIL import Image
from torchvision import transforms

# Load and preprocess the image
image_path = test_dir + '/test/' + 'AppleScab1.JPG'
image = Image.open(image_path).convert('RGB')
preprocess = transforms.Compose([
    transforms.Resize((224, 224)),
    transforms.ToTensor(),
])
input_tensor = preprocess(image).unsqueeze(0).numpy()  # Add batch dimension

# ✅ Run ONNX model inference
input_name = session.get_inputs()[0].name
output = session.run(None, {input_name: input_tensor})

# ✅ Get prediction
predictions = output[0]
predicted_class = np.argmax(predictions)

print(f"Predicted class: {predicted_class}")
```

International Journal of Scientific Research in Engineering and Management (IJSREM)
Volume: 09 Issue: 05 | May - 2025 SJIF Rating: 8.586 ISSN: 2582-3930

```python
class_names[37]
```

```
'Tomato___healthy'
```

```python
!pip install onnxruntime
```

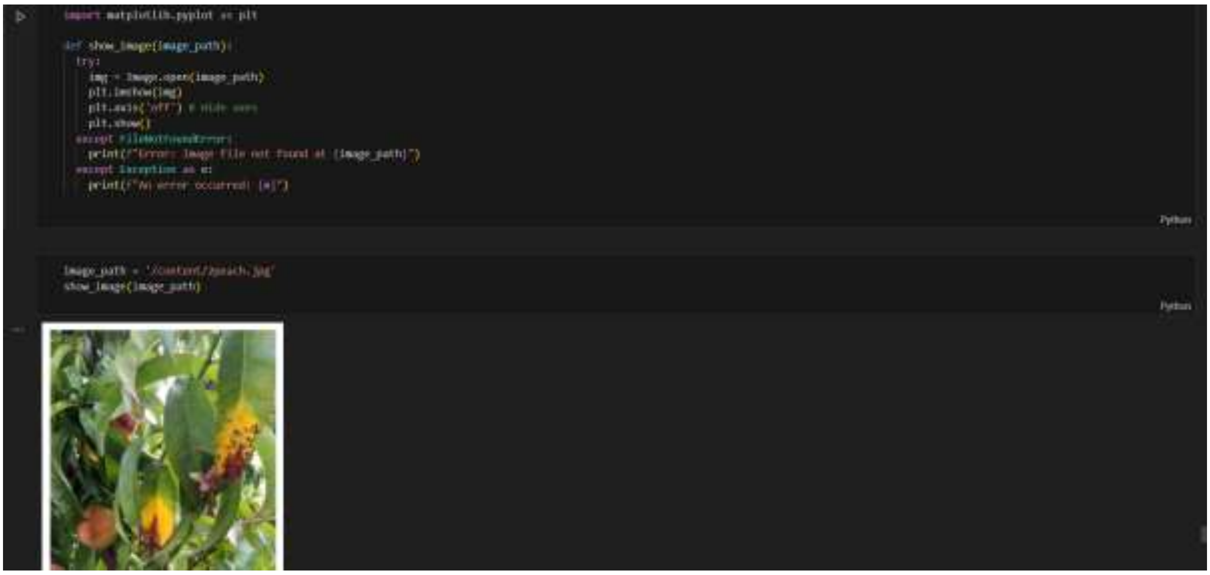
```
Collecting onnxruntime
  Downloading onnxruntime-1.21.0-cp311-cp311-manylinux_2_27_x86_64.manylinux_2_28_x86_64.whl.metadata (4.5 kB)
Collecting coloredlogs (from onnxruntime)
  Downloading coloredlogs-15.0.1-py2.py3-none-any.whl.metadata (12 kB)
Requirement already satisfied: flatbuffers in /usr/local/lib/python3.11/dist-packages (from onnxruntime) (25.2.10)
Requirement already satisfied: numpy>=1.21.6 in /usr/local/lib/python3.11/dist-packages (from onnxruntime) (2.0.2)
Requirement already satisfied: packaging in /usr/local/lib/python3.11/dist-packages (from onnxruntime) (24.2)
Requirement already satisfied: protobuf in /usr/local/lib/python3.11/dist-packages (from onnxruntime) (5.29.4)
Requirement already satisfied: sympy in /usr/local/lib/python3.11/dist-packages (from onnxruntime) (1.13.1)
Collecting humanfriendly>=9.1 (from coloredlogs->onnxruntime)
  Downloading humanfriendly-10.0-py2.py3-none-any.whl.metadata (9.2 kB)
Requirement already satisfied: mpmath<1.4,>=1.1.0 in /usr/local/lib/python3.11/dist-packages (from sympy->onnxruntime) (1.3.0)
Downloading onnxruntime-1.21.0-cp311-cp311-manylinux_2_27_x86_64.manylinux_2_28_x86_64.whl (16.0 MB)
   ---------------------------------------- 16.0/16.0 MB eta 0:00:00
Downloading coloredlogs-15.0.1-py2.py3-none-any.whl (46 kB)
   ---------------------------------------- 46.0/46.0 kB eta 0:00:00
Downloading humanfriendly-10.0-py2.py3-none-any.whl (86 kB)
   ---------------------------------------- 86.8/86.8 kB eta 0:00:00
Installing collected packages: humanfriendly, coloredlogs, onnxruntime
Successfully installed coloredlogs-15.0.1 humanfriendly-10.0 onnxruntime-1.21.0
```

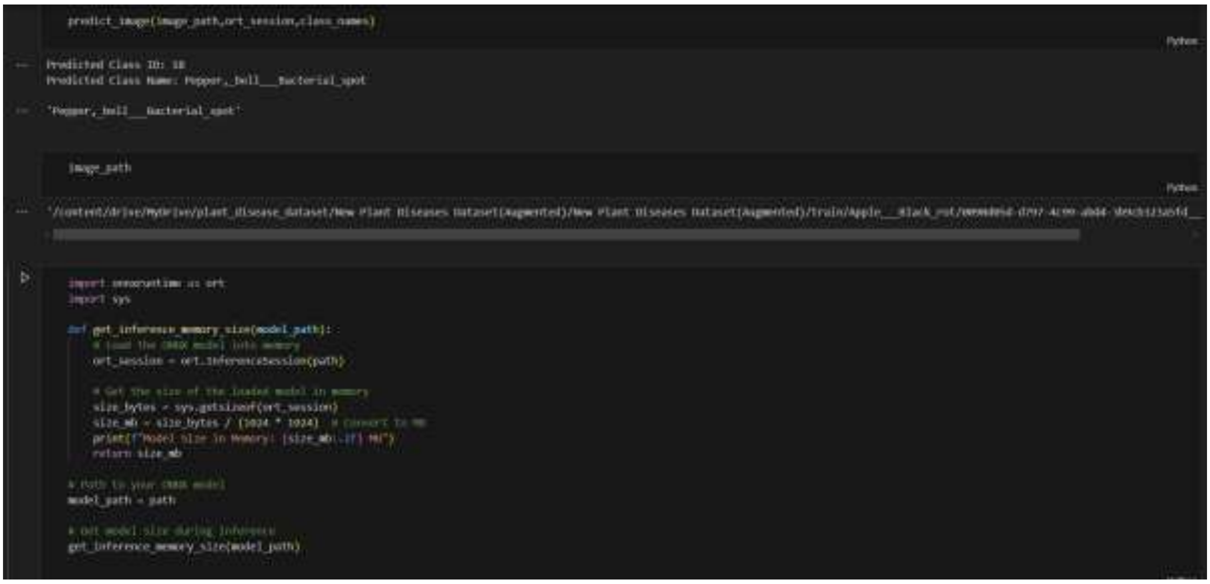
```python
import onnxruntime as ort
import numpy as np
from torchvision import transforms
from torchvision import datasets
from PIL import Image
```

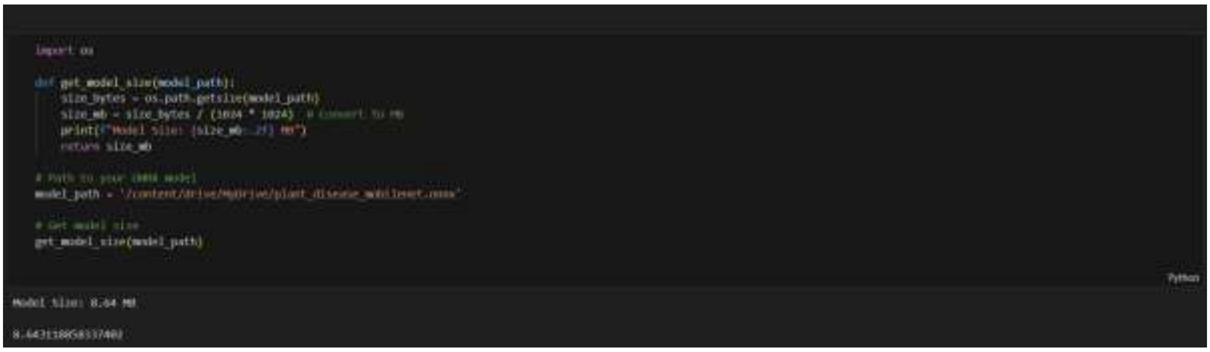
```
Collecting onnx-tf
  Downloading onnx_tf-1.10.0-py3-none-any.whl.metadata (510 bytes)
Collecting onnx>=1.10.2 (from onnx-tf)
  Downloading onnx-1.17.0-cp311-cp311-manylinux_2_17_x86_64.manylinux2014_x86_64.whl.metadata (16 kB)
Requirement already satisfied: PyYAML in /usr/local/lib/python3.11/dist-packages (from onnx-tf) (6.0.2)
Collecting tensorflow-addons (from onnx-tf)
  Downloading tensorflow_addons-0.23.0-cp311-cp311-manylinux_2_17_x86_64.manylinux2014_x86_64.whl.metadata (1.8 kB)
Requirement already satisfied: numpy>=1.20 in /usr/local/lib/python3.11/dist-packages (from onnx>=1.10.2->onnx-tf) (2.0.2)
Requirement already satisfied: protobuf>=3.20.2 in /usr/local/lib/python3.11/dist-packages (from onnx>=1.10.2->onnx-tf) (5.29.4)
Requirement already satisfied: packaging in /usr/local/lib/python3.11/dist-packages (from tensorflow-addons->onnx-tf) (24.2)
Collecting typeguard<3.0.0,>=2.7 (from tensorflow-addons->onnx-tf)
  Downloading typeguard-2.13.3-py3-none-any.whl.metadata (3.6 kB)
Downloading onnx_tf-1.10.0-py3-none-any.whl (226 kB)
   ---------------------------------------- 226.1/226.1 kB eta 0:00:00
Downloading onnx-1.17.0-cp311-cp311-manylinux_2_17_x86_64.manylinux2014_x86_64.whl (16.0 MB)
   ---------------------------------------- 16.0/16.0 MB eta 0:00:00
Downloading tensorflow_addons-0.23.0-cp311-cp311-manylinux_2_17_x86_64.manylinux2014_x86_64.whl (611 kB)
   ---------------------------------------- 611.8/611.8 kB eta 0:00:00
Downloading typeguard-2.13.3-py3-none-any.whl (17 kB)
Installing collected packages: typeguard, onnx, tensorflow-addons, onnx-tf
  Attempting uninstall: typeguard
    Found existing installation: typeguard 4.4.2
    Uninstalling typeguard-4.4.2:
```

```python
def predict_image(image_path, ort_session, class_names):
    # Load image using PIL (similar to datasets.folder.default_loader)
    image = Image.open(image_path).convert('RGB')

    # Define the transform: Resize, normalize, and convert to tensor
    transform = transforms.Compose([
        transforms.Resize((224, 224)),  # Resize to 224x224 (standard for models like ResNet)
        transforms.ToTensor(),
        transforms.Normalize([0.485, 0.456, 0.406], [0.229, 0.224, 0.225])  # Standard normalization
    ])

    # Apply the transformation
    image_tensor = transform(image).unsqueeze(0)  # Add batch dimension

    # Convert tensor to numpy for ONNX inference
    image_numpy = image_tensor.numpy()

    # Load the ONNX model


    # Get the model's input name
    input_name = ort_session.get_inputs()[0].name

    # Run inference
    output = ort_session.run(None, {input_name: image_numpy})

    # Get predicted class (argmax of the output)
    pred = np.argmax(output[0])
    class_name = class_names[pred]

    print(f"Predicted Class ID: {pred}")
    print(f"Predicted Class Name: {class_name}")
    return class_name
```

International Journal of Scientific Research in Engineering and Management (IJSREM)
Volume: 09 Issue: 05 | May - 2025 SJIF Rating: 8.586 ISSN: 2582-3930







10. Findings:

10.1 Test:

The PyTorch framework serves as the foundation for the experiment. It is an Intel (R) Core (TM) i9 CPU. The graphics card is an NVIDIA GeForce RTX3080 10 GB, and the RAM is 16 GB. Because every model in the VGG, ResNet, and DenseNet series included several sub-models. Furthermore, the tests that followed, which tested the accuracy of various combinations of activation functions using various sub-models and functions, were overly complex. As a result, all three networks' submodels underwent benchmarking. display the experimental outcomes. Among the three network models, VGG19, ResNet50, and DenseNet161 were found to perform the best. These three sub-models would thus be used in further studies to evaluate the self-network models.

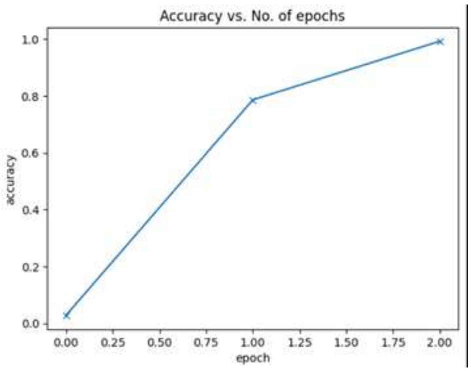


Fig-3 : Accuracy Vs No. of epochs

The graph *"Accuracy vs. No. of Epochs"* presents the progression of a machine learning model's accuracy over three training epochs. The horizontal axis denotes the number of epochs, while the vertical axis indicates the accuracy achieved, ranging from 0 to 1. Initially, at epoch 0, the model's accuracy is close to zero, signifying poor performance due to the lack of training. By epoch 1, there is a notable improvement, with the accuracy rising sharply to approximately 0.78, reflecting the model's learning from the data. At epoch 2, the model reaches an accuracy of 1.0, indicating perfect performance on the training data. While this may appear ideal, such a rapid increase to perfect accuracy in just a few epochs could suggest overfitting, where the model memorizes the training data rather than generalizing from it. This emphasizes the need for further evaluation on validation or test datasets to ensure the model's robustness and generalization capability.

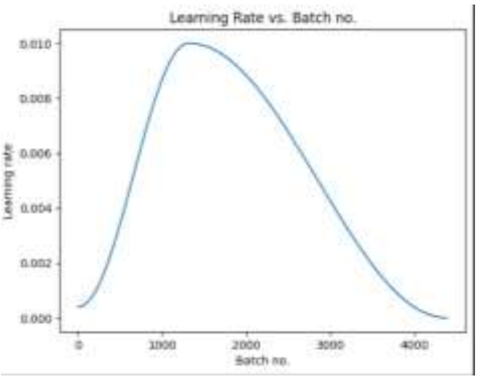


Fig-4: Learning Rate vs. Batch no.

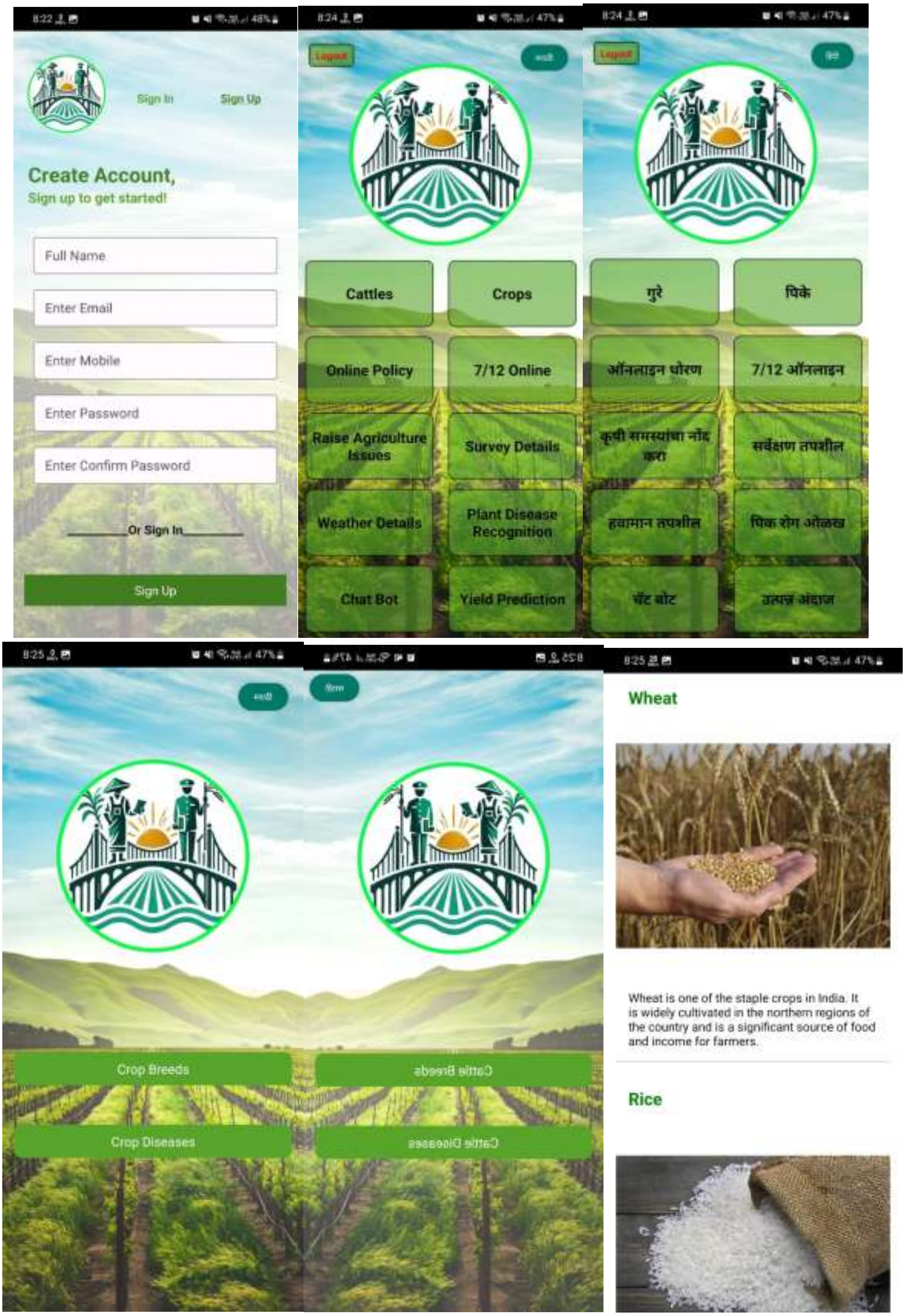
The graph illustrates the variation of the learning rate with respect to the batch number during the training of a machine learning model. Initially, the learning rate increases steadily, reaching a peak value of approximately 0.01 around batch number 1300. After this point, the learning rate gradually decreases. This cyclical learning rate schedule, often referred to as a triangular or cosine annealing schedule, is typically used to improve convergence and model performance

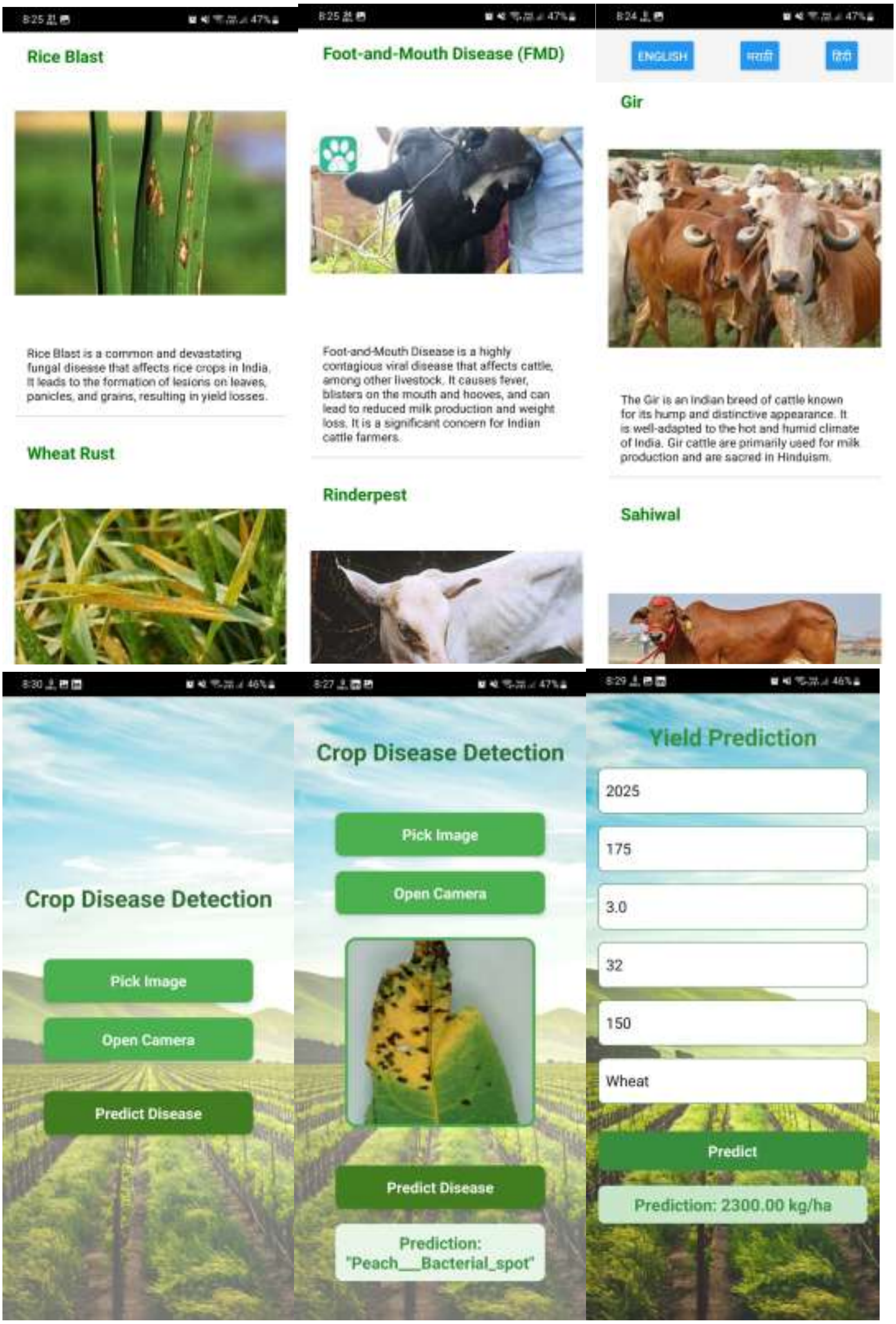
10.1.1. Method of Training:

PyTorch provides the pre-training model parameters utilized in this work, which are based on the ImageNet dataset. ImageNet is a classification task that needs to divide the photos into 1000 classes. In this study, the 1000 parameters of the network's last completely linked layer must be changed to four.

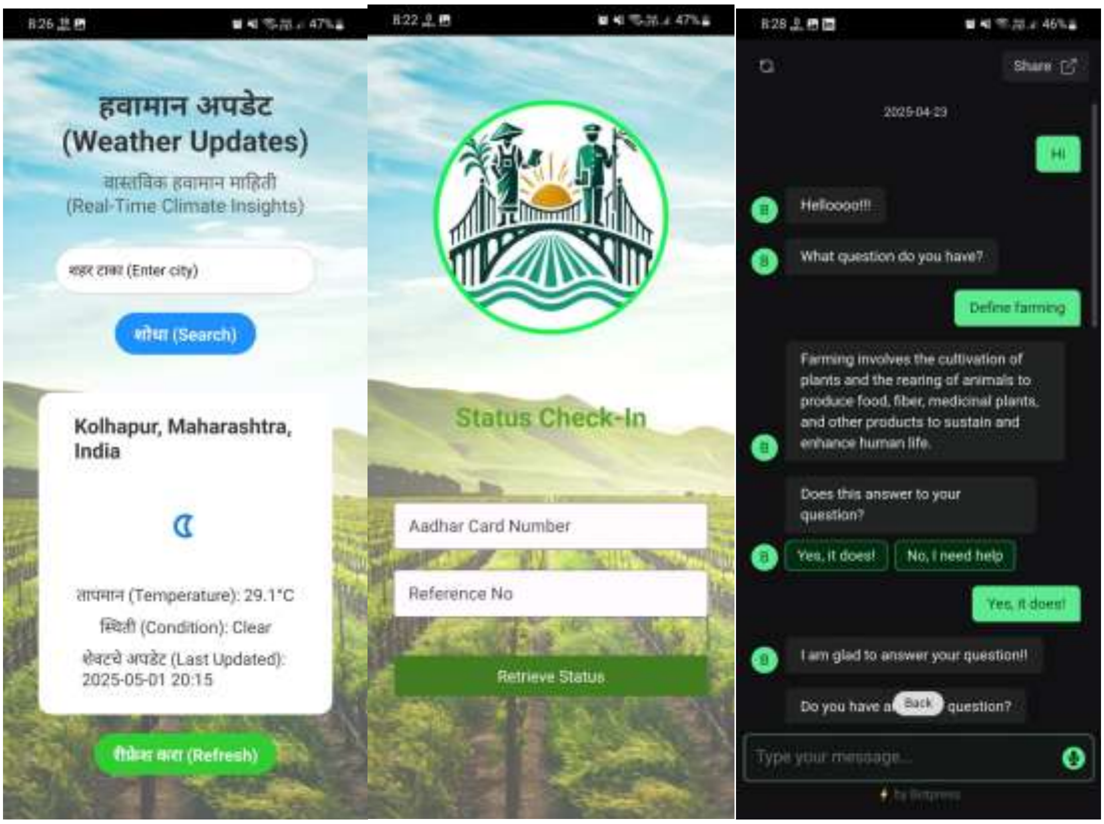
The last fully connected layer has one-250th of the initial number of parameters, whereas the first convolutional layer has one-third of the original amount. The initialization technique used in this work is the Kaiming initialization technique, which was put out by Kaiming [20]. The non-saturated activation function ReLU and its variant types are a good fit for this approach. The samples used in this study were split into training and validation sets using a 9:1 split. SGD (stochastic gradient descent) [21] was the loss function optimization technique utilized for training, with a batch size value of 50 and a momentum parameter of 0.9. The validation set's accuracy tended to converge after 50 rounds. Overfitting and a decline in the validation set's accuracy will result with additional training. Consequently, following 200 iterations, the model parameters were chosen as the model parameters trained.

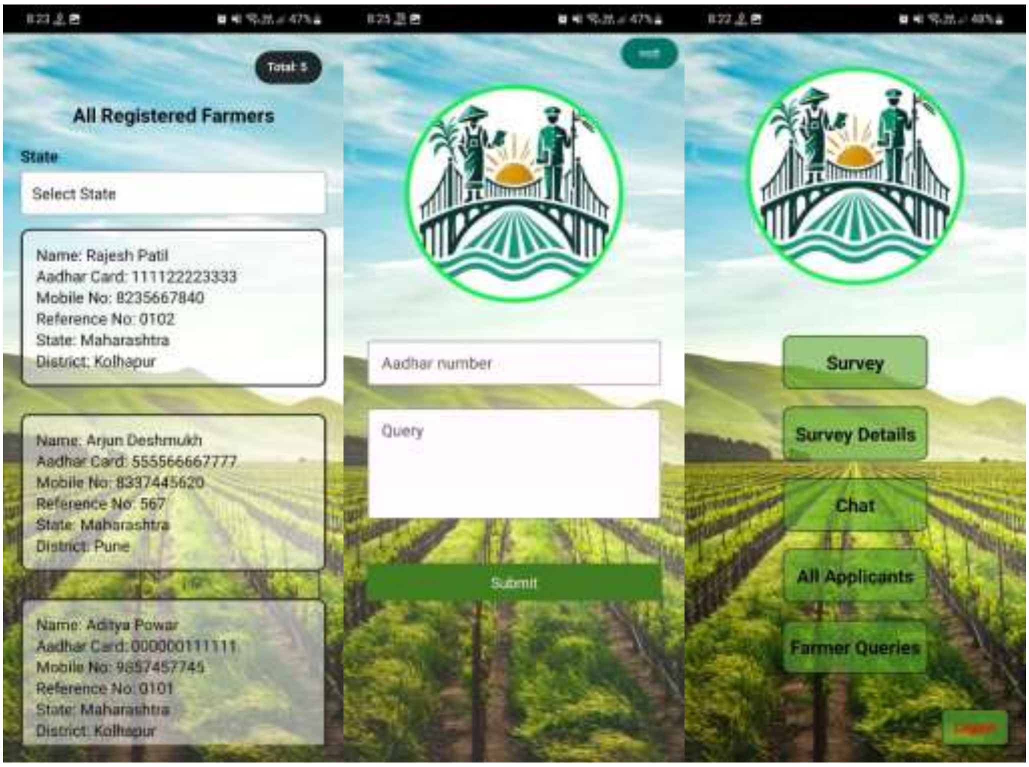
11. Snapshoots of Agri Nama Application:


Sign In
Sign Up
Create Account,
Sign up to get started!
Full Name
Enter Email
Enter Mobile
Enter Password
Enter Confirm Password
Or Sign In
Sign Up
Cattles
Crops
Online Policy
7/12 Online
Raise Agriculture Issues
Survey Details
Weather Details
Plant Disease Recognition
Chat Bot
Yield Prediction
गुरे
पिके
ऑनलाइन धोरण
7/12 ऑनलाइन
कृषी समस्यांचा नोंद करा
सर्वेक्षण तपशील
हवामान तपशील
पिक रोग ओळख
चॅट बॉट
उत्पन्न अंदाज
Crop Breeds
Crop Diseases
Cattle Breeds
Cattle Diseases
Wheat
Wheat is one of the staple crops in India. It is widely cultivated in the northern regions of the country and is a significant source of food and income for farmers.
Rice


Rice Blast
Rice Blast is a common and devastating fungal disease that affects rice crops in India. It leads to the formation of lesions on leaves, panicles, and grains, resulting in yield losses.
Wheat Rust
Foot-and-Mouth Disease (FMD)
Foot-and-Mouth Disease is a highly contagious viral disease that affects cattle, among other livestock. It causes fever, blisters on the mouth and hooves, and can lead to reduced milk production and weight loss. It is a significant concern for Indian cattle farmers.
Rinderpest
ENGLISH
मराठी
हिंदी
Gir
The Gir is an Indian breed of cattle known for its hump and distinctive appearance. It is well-adapted to the hot and humid climate of India. Gir cattle are primarily used for milk production and are sacred in Hinduism.
Sahiwal
Crop Disease Detection
Pick Image
Open Camera
Predict Disease
Crop Disease Detection
Pick Image
Open Camera
Predict Disease
Prediction:
"Peach___Bacterial_spot"
Yield Prediction
2025
175
3.0
32
150
Wheat
Predict
Prediction: 2300.00 kg/ha

International Journal of Scientific Research in Engineering and Management (IJSREM)
Volume: 09 Issue: 05 | May - 2025 SJIF Rating: 8.586 ISSN: 2582-3930






Fill The Survey Form
All fields are mandatory
Account No .... (11 digit)
Farmer name
Aaddhar card no .... (12 digit)
Mobile no .... (10 digit)
Bank Account Number
IFSC Code
Total: 3
All Registered Farmers
State
Maharashtra
District
Kolhapur
Name: Rajesh Patil
Aadhar Card: 111122223333
Mobile No: 8235667840
Reference No: 0102
State: Maharashtra
District: Kolhapur
Name: Aditya Powar
Aadhar Card: 000000111111
Mobile No: 9857457745
Reference No: 0101
State: Maharashtra
District: Kolhapur
Name: Ram patil
Aadhar Card: 111111444444
Status Check-In
Aadhar Card Number
Reference No
Retrieve Status


AgriNama Chat
Chaitanya
Sujal
Karan
Shubham
Farmer's Queries
Aadhar Number: 000000111111
Query: Which crop is best to grow in my area this season?
Status: Pending Approval
Notify Farmer
Aadhar Number: 000000222222
Query: Why are my cotton plants turning yellow?
Status: Pending Approval
Notify Farmer
Aadhar Number: 111111000000
Query: Can I mix pesticides and fertilizers while spraying?
Status: Approved by Officer
Delete
Aadhar Number: 555566667777
Query: Is there any subsidy for drip irrigation?
Status: Pending Approval

12. CONCLUSION:

With the purpose of providing farmers, government representatives, and agricultural specialists with technologically advanced resources, our application functions as a full digital solution for the agriculture industry. We guarantee accurate and fast information delivery by integrating modules like LiveQuery, Chatbot Assistance, and Weather Forecasting. Improved awareness, transparency, and effective government-farmer interactions are encouraged by modules such as GovLinks, AgriSurvey, and Crops and Cattles Information. Crop disease detection and yield prediction's sophisticated features use machine learning to improve crop management and decision-making. These modules work together to create a strong ecosystem that aims to raise farmers' incomes, lower risks, and increase production. This application represents a significant and scalable step toward a more intelligent and sustainable agricultural future.

REFRENCES:

[1] S. Veenadhari, Dr. Bharat Misra, Dr. CD Singh. Machine learning approach for forecasting crop yield based on climatic parameters. International Conference on Computer Communication and Informatics (ICCCI).

[2] Shweta K Shahane, Prajakta V Tawale. Prediction On Crop Cultivation. IInternational Journal of Advanced Research in Computer Science and Electronics Engineering (IJARCSEE) Volume 5, Issue 10, October 2016.

[3] D Ramesh, B Vishnu Vardhan. Analysis Of Crop Yield Prediction Using Data Mining Techniques. IJRET: International Journal of Research in Engineering and Technology.

[4] Sharma OP. 2014. Science communication through mobile devices. In *Communicating Science to the Public*, Tan Wee Hin L, Subramaniam R (eds). Springer: Dordrecht; 247–260, DOI: 10.1007/ 978-94-017-9097-0.

[5] Sink SA. 1995. Determining the public's understanding of precipitation forecasts: results of a survey. *Natl. Weather Dig.* **19**(3): 9–15.

[6] Sivle AD, Kolstø SD, Kirkeby Hansen PJ, Kristiansen J. 2014. How do laypeople evaluate the degree of certainty in a weather report? A case study of the use of the web service yr. no. *Weather Clim. Soc.* **6**(3): 399–412.

[7] Joslyn SL, Nichols RM. 2009. Probability or frequency? Expressing forecast uncertainty in public weather forecasts. *Meteorol. Appl.* **314**: 309–314.

[8] Joslyn SL, Pak K, Jones D, Pyles J, Hunt E. 2007. The effect of probabilistic information on threshold forecasts. *Weather Forecasting* **22**: 804–812.

[9] Joslyn S, Savelli S. 2010. Communicating forecast uncertainty: public perception of weather forecast uncertainty. *Meteorol. Appl.* **17**: 180–195.

[10]Research Paper: "Chatbots in Agriculture"

Citation: Kasinathan, R., et al. "AI Powered Chatbots in Agriculture." *International Journal of Scientific Research in Computer Science Applications and Management Studies*, Vol. 8, Issue 1, 2019.

[11] Botpress Official Documentation:

Description: Comprehensive guide on building chatbots with Botpress URL: https://botpress.com/docs/

[12] OpenWeatherMap API (Weather Forecast API):

Description: Used for integrating real-time weather forecasts URL: https://openweathermap.org/api

[13] Radoglou-Grammatikis, P.; Sarigiannidis, P.; Lagkas, T.; Moscholios, I. A compilation of UAV applications for precision agriculture. Comput. Netw. **2020**, 172, 107148. [CrossRef]

[14] Terentev, A.; Dolzhenko, V.; Fedotov, A.; Eremenko, D. Current state of hyperspectral remote sensing for early plant disease detection: A review. Sensors **2022**, 22, 757. [CrossRef]

[15] Tsouros, D.C.; Bibi, S.; Sarigiannidis, P.G. A review on UAV-based applications for precision agriculture. Information **2019**, 10, 349. [CrossRef]

[16] Huang, S.; Tang, L.; Hupy, J.P.;Wang, Y.; Shao, G. A commentary review on the use of normalized difference vegetation index (NDVI) in the era of popular remote sensing. J. For. Res. **2021**, 32, 1–6. [CrossRef]

[17] Sanseechan, P.; Saengprachathanarug, K.; Posom, J.; Wongpichet, S.; Chea, C.; Wongphati, M. Use of vegetation indices in monitoring sugarcane white leaf disease symptoms in sugarcane field using multispectral UAV aerial imagery. In Proceedings of the IOP Conference Series: Earth and Environmental Science; IOP Publishing: Bristol, UK, 2019; Volume 301, p. 012025.

[18] Kauth, R.J.; Thomas, G. The tasselled cap–a graphic description of the spectral-temporal development of agricultural crops as seen by Landsat. In Proceedings of the LARS Symposia,West Lafayette, IN, USA, 29 June–1 July 1976; p. 159.

BIOGRAPHIES (Optional not mandatory)

Mr. Shubham Kapase is a software engineering student with a focus on MI and mobile app development. His Pursuing B.E Degree from DY Patil Collage of Engineering and Technology Kolhapur

Mr. Prabuddha Sonalkar is a software engineering student with a focus on MI and mobile app development. His Pursuing B.E Degree from DY Patil Collage of Engineering and Technology Kolhapur

Mr. Karan Patil is a software engineering student with a focus on MI and mobile app development. His Pursuing B.E Degree from DY Patil Collage of

Engineering and Technology Kolhapur

Mr. Sujal Vadgave is a software engineering student with a focus on MI and mobile app development. His Pursuing B.E Degree from DY Patil Collage of Engineering and Technology Kolhapur

Mr. Chaitanya Chougale is a software engineering student with a focus on MI and mobile app development. His Pursuing B.E Degree from DY Patil Collage of Engineering and Technology Kolhapur