PREDICTIVE AGRICULTURE:IMPROVING CROP YIELDS WITH DATA-DRIVEN INSIGHTS

A PROJECT REPORT

Submitted by

ABINASH M	**422619104001**
AKALYA A	**422619104003**
BHAVESHRAJ A	**422619104006**

In partial fulfilment 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 2023

PREDICTIVE AGRICULTURE:IMPROVING CROP YIELDS WITH DATA-DRIVEN INSIGHTS

A PROJECT REPORT

Submitted by

ABINASH M	**422619104001**
AKALYA A	**422619104003**
BHAVESHRAJ A	**422619104006**

In partial fulfilment 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 2023

ANNA UNIVERSITY: CHENNAI 600 025

BONAFIDE CERTIFICATE

Certified that this project report "**PREDICTIVE AGRICULTURE: IMPROVING CROP YIELDS WITH DATA-DRIVEN INSIGHTS**" is the bonafide work of "**ABINASH M (422619104001), AKALYA A (422619104003), BHAVESH RAJ A (422619104006)**" who carried out the project work under my supervision.

SIGNATURE
Dr.D.Muruganandam, M.Tech,
Ph.D., Assistant Professor / CSE

HEAD OF DEPARTMENT
Computer Science & Engineering
University College of Engineering,
Panruti – 607106

SIGNATURE
Mrs.D.Anitha M.E.,
TF/CSE

SUPERVISOR
Computer Science & Engineering
University College of Engineering,
Panruti – 607106

EXAMINATION HELD ON _____

INTERNAL EXAMINER **EXTERNAL EXAMINER**

DECLARATION

We hereby declare that the work entitled "**PREDICTIVE AGRICULTURE: IMPROVING CROP YIELDS WITH DATA-DRIVEN INSIGHTS**" is submitted in partial fulfilment for the award of the degree in **Bachelor of Engineering in Computer Science & Engineering. University College of Engineering, Panruti** is a record of our own work carried out by us during the academic year 2020-2021. Under the supervision and guidance of **Mrs.A.Sivaranjani M.Tech.,TF/CSE**, Department of Computer Science and Engineering, UNIVERSITY COLLEGE OF ENGINEERING PANRUTI. The extent and source of information are derived from the existing literature and have been indicated through dissertation at the appropriate places. The matter embodied in this work is original and has not been submitted for the award of any other degree or diploma,either in this or any other university.

REGISTER NUMBER	NAME	SIGNATURE
422619104001	ABINASH M	
422619104003	AKALYA A	
422619104006	BHAVESHRAJ A	

I certify that the declaration made above by the candidate is true

SIGNATURE
Mrs.D.Anitha M.E.,
TF/CSE
SUPERVISOR
Computer Science & Engineering
University College of Engineering,
Panruti - 607106

ABSTRACT

This study presents a machine learning image prediction model to assist farmers in the field of agriculture by predicting patches in crop fields. The proposed model uses convolutional neural networks to analyse high-resolution images of crop fields and identify areas that require attention. The model is trained on a large dataset of annotated images, and uses a sliding window approach to scan the entire image and predict the patches that need attention. The model achieved high accuracy in detecting various patches in crop fields, such as areas affected by waterlogging, nutrient deficiency, or pest infestation. The system also includes a user-friendly interface that allows farmers to easily upload images and receive real-time predictions, enabling them to take timely action to prevent crop damage and improve yields. The results of this study demonstrate the potential of machine learning in agriculture, and highlight the importance of integrating new technologies to support sustainable farming practices.It have the potential to transform the agriculture industry, enabling farmers to make more informed decisions and optimize their crop yields. As the agriculture industry continues to face challenges such as climate change, water scarcity, and population growth, it is likely that these technologies will become increasingly important in supporting sustainable farming practices.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1
INTRODUCTION

1.1 DATA COLLECTION

Data collection is the process of gathering and acquiring information from various sources. It involves the systematic gathering of data or information from a variety of sources for a specific purpose. The data collected may be in the form of numbers, words, images, or other formats.

- **Identify data sources:** Once you have defined your problem statement, the next step is to identify potential data sources. This can include public datasets, proprietary data, web scraping, and third-party data providers.

- **Data quality:** Ensure that the data you collect is of high quality, relevant, and unbiased. Low-quality data can negatively impact the accuracy of your machine learning model, and biassed data can lead to incorrect predictions.

- **Data quantity:** Collect enough data to ensure that your machine learning model has enough examples to learn from and generalise to new data.

- **Data labelling:** If your machine learning model requires labelled data, you may need to label the data manually or use crowdsourcing platforms to help with the labelling process.

- **Data storage:** Store your data in a way that is easily accessible, secure, and scalable. Cloud storage solutions such as Amazon S3, Microsoft Azure, and Google Cloud Storage are popular options.

- **Data augmentation:** Data augmentation techniques can help increase the diversity of your dataset and improve the performance of your machine learning model. Techniques include flipping images, adding noise to audio data, and synthesising new data.

Overall, data collection is a critical step in a machine learning project, and careful consideration of the above factors can help ensure that you collect high-quality data that leads to accurate and robust machine learning models.

1.2 MODEL ARCHITECTURE

A model architecture refers to the overall design and structure of a machine learning model. It is the blueprint that determines how the model processes input data and generates output predictions.

A model architecture is typically composed of several layers of interconnected nodes, where each layer performs a specific type of computation on the input data. The number and size of the layers, as well as the types of computations performed, can vary widely depending on the specific problem being solved.

The choice of model architecture is a crucial step in developing a machine learning model. Different architectures can have a significant impact on the model's performance, speed, and ability to generalise to new data. For example, a convolutional neural network (CNN) architecture is often used for image recognition tasks, while a recurrent neural network (RNN) is often used for natural language processing tasks.

1.3 COMPILE THE MODEL

In machine learning, compiling a model refers to the process of configuring a model with a specific optimizer, loss function, and metrics before it can be trained on a dataset. Compiling a model is a necessary step before training a model.

- **Define the model architecture:** This involves creating and configuring the layers of the neural network or machine learning model.

- **Compile the model:** This involves specifying the optimizer, loss function, and metrics to be used during training.

- **Optimizer:** An optimizer is an algorithm used to update the parameters of the model in order to minimise the loss function. Some commonly used optimizers include Stochastic Gradient Descent (SGD), Adam, and Adagrad.

- **Loss function:** A loss function is a function that measures how well the model performs on the training data. It is used to compare the predicted output of the model with the actual output. Some commonly used loss functions include mean squared error, categorical cross-entropy, and binary cross-entropy.

- **Metrics:** Metrics are used to evaluate the performance of the model during training and testing. Some commonly used metrics include accuracy, precision, recall, and F1 score.

- **Train the model:** Once the model has been compiled, it can be trained on a dataset using the specified optimizer, loss function, and metrics.

1.4 TRAIN THE MODEL

Training a machine learning model involves feeding the model with input data and its corresponding output, known as labels or targets, in order to adjust the model's weights and biases to optimise its performance on the given task. The training process typically involves the following steps:

1. **Split the data:** The dataset is split into training data and validation data. The training data is used to train the model, while the validation data is used to evaluate the model's performance during training.

2. **Prepare the data:** The data is preprocessed and transformed into a format that can be used by the model. This may involve tasks such as data cleaning, normalisation, and feature engineering.

3. **Define the model:** The model architecture is defined and compiled with an optimizer, loss function, and evaluation metric.

4. **Train the model:** The model is trained on the training data by iterating through the dataset and adjusting the model's parameters to minimise the loss function. This process is repeated for a fixed number of epochs or until a stopping criterion is met.

5. **Evaluate the model:** The model is evaluated on the validation data to measure its performance on data that it has not seen before. The evaluation metric is used to assess the model's performance and identify areas for improvement.

6. **Fine-tune the model:** Based on the evaluation results, the model may be fine-tuned by adjusting the model architecture or hyperparameters and repeating the training and evaluation process.

1.5 EVALUATE THE MODEL

Evaluating a machine learning model involves measuring its performance on a separate set of data, known as the test data, that the model has not seen before. This is done to get an estimate of how well the model can generalise to new, unseen data.

- **Load the test data:** The test data is loaded into memory and preprocessed in the same way as the training data.

- **Predict the outputs:** The model is used to predict the outputs for the test data.

- **Calculate the evaluation metrics:** The predicted outputs are compared to the true outputs in the test data to calculate the evaluation metrics, such as accuracy, precision, recall, F1 score, or mean squared error, depending on the type of problem.

1.6 FINE TUNING:

Fine-tuning a machine learning model involves adjusting the model's architecture or hyperparameters to improve its performance on a specific task. Fine-tuning can be done after the initial training of the model, using the validation data to assess the model's performance and make adjustments.

1. **Adjusting the learning rate:** The learning rate controls the step size taken during gradient descent optimization. If the learning rate is too high, the model may overshoot the minimum of the loss function and fail to converge. If the learning rate is too low, the model may converge too slowly. Fine-tuning the learning rate involves experimenting with different values to find the optimal rate for the model.

2. **Changing the optimizer:** The choice of optimizer can have a significant impact on the model's performance. Different optimizers use different algorithms to update the model's parameters during training. Fine-tuning the optimizer involves experimenting with different optimizers, such as SGD, Adam, or RMSprop, to find the one that works best for the given task.

3. **Adding or removing layers:** The model architecture can be modified by adding or removing layers. Adding more layers can increase the model's capacity and enable it to learn more complex representations, while removing layers can reduce overfitting and simplify the model. Fine-tuning the model architecture involves experimenting with different architectures to find the one that works best for the given task.

4. **Adjusting regularisation:** Regularisation techniques, such as L1 or L2 regularisation, can be used to prevent overfitting by adding a penalty term

to the loss function. Fine-tuning the regularisation involves experimenting with different values of the regularisation parameter to find the optimal value for the given task.

5. **Using pre-trained models:** Pre-trained models are models that have been trained on a large dataset and can be fine-tuned on a smaller dataset for a specific task. Fine-tuning a pre-trained model involves freezing some of the layers of the model and training only the remaining layers on the new dataset, or fine-tuning all the layers of the model on the new dataset.

1.7 DATA AUGMENTATION

Data augmentation is a technique used in machine learning to increase the amount of training data by creating new examples from the existing data. The idea is to apply various transformations to the input data such as rotation, zooming, flipping, and shearing to create new examples that are still representative of the original data but can help to improve the model's performance. Data augmentation can be especially useful in situations where the training data is limited, noisy or unbalanced.

Common data augmentation techniques:

- **Rotation:** Rotating an image at various angles to create new images.

- **Flipping:** Flipping an image horizontally or vertically to create a mirror image.

- **Zooming:** Zooming in or out of an image to create new examples.

- **Shearing:** Applying a shearing transformation to the image to skew it in one direction.

- **Translation:** Shifting an image horizontally or vertically to create new examples.

- **Gaussian noise:** Adding Gaussian noise to an image to make it more robust to noise.

- **Colour jittering:** Randomly adjusting the brightness, contrast, saturation, and hue of an image to create new examples.

1.8 DEPLOYING THE MODEL

Deploying a machine learning model involves taking the trained model and making it available to users or other systems in a production environment. There are several steps involved in deploying an ML model, including:

1. **Saving the trained model:** The first step is to save the trained model to a file or object that can be loaded and used in a production environment.

2. **Creating an API:** Once the model is saved, it is typically necessary to create an API (application programming interface) that exposes the model to other systems or users. This can be done using a web framework like Flask or Django.

3. **Deploying the API:** The API can then be deployed to a web server or cloud platform like AWS or Google Cloud Platform, where it can be accessed by users or other systems.

4. **Scaling and monitoring:** Once the API is deployed, it is important to monitor its performance and scale it as needed to handle increasing demand.

5. **Updating the model:** As new data becomes available or the model's performance improves, it may be necessary to update the model and redeploy the API.

Overall, deploying an ML model involves taking the trained model and making it available in a production environment, where it can be accessed by users or other systems. This involves creating an API, deploying it to a web server or cloud platform, monitoring its performance, and updating the model as needed.

CHAPTER 2
LITERATURE SURVEY

2.1 TITLE: CROP YIELD PREDICTION USING MACHINE LEARNING ALGORITHM.

AUTHOR: ISWARIYA, NAGAPOOJA, RAGAVI.(2022)

Agriculture is the backbone of the Indian economy, with more than half of the country's people relying on it for subsistence. Crop production is predicted using machine learning techniques based on parameters such as rainfall, crop, and meteorological conditions. The most popular and powerful supervised machine learning algorithm, Random Forest, can do both classification and regression tasks. They are used in crop selection to reduce crop yield output losses, regardless of the distracting environment. Weather, climate, and other related environmental elements have posed a significant danger to agriculture's long-term viability. Machine learning (ML) is significant since it offers a decision-support tool for Crop Yield Prediction (CYP), which may help with decisions like which crops to cultivate and what to do during the crop's growing season. Crop yield estimation's major purpose is to boost agricultural crop production, and it does so using a variety of well-established models. Machine learning is increasingly widely used around the world due to its success in a range of disciplines such as forecasting, fault detection, pattern identification, and so on. A key agricultural concern is a yield prediction. Farmers will be able to determine the yield of their crop before growing on the agricultural field using the results of this study, allowing them to make informed decisions. Machine learning is increasingly widely used around the world due to its success in a range of disciplines such as forecasting, fault detection, pattern identification, and so on. A key agricultural concern is a yield prediction. Farmers will be able to determine the yield of their crop before growing.

2.2 TITLE: CROP AND WEED CLASSIFICATION IN AERIAL IMAGERY SESAME CROP FIELD USING PATCH BASED DEEP LEARNING MODELS.

AUTHOR: UMARS.KHAN, WAQARS.QURESHI , WAHIR NAMAS.(2022)

Sesame (Sesamum indicum L.) is an important commercial and food crop, and its yields are limited by many insects, pests, diseases, and weeds. Autonomous aerial agrochemicals spray application on sesame fields using a drone aims to save crops from these yield limiting factors and in addition agrochemicals application quantity and site could be controlled, and human health is expected to be protected. For accurate and selective spray application, autonomous systems would need some parameters to distinguish between crop, weed and background. In this research an aerial sesame field dataset has been collected with the focus to classify patch areas of sesame and weeds present in the field. Dataset was captured using Agrocam. We have developed a patch image-based classification approach along with a novel SesameWeedNet convolutional neural network (CNN) inspired by the layer's configuration of VGG networks and depth-wise convolutions of the MobileNet. The small model contains 6 convolutional layers, and it runs faster and accurately on small patch images. Our approach breaks 1920×1080-pixel images into smaller patch images of size 45×45 pixels. After that, these small patch images are fed to a relatively small CNN for training, validation, and finally for classification. Patch based model ensemble and dataset grouping are two major parts in our methodology. Our system recommends the dataset grouping according to vegetation present in the images to enhance classification results. We have achieved accuracy up to 96.7% with our proposed method. We have tested our system under sunlight variation, in wet and dry soil conditions and at different growth stages.

2.3 TITLE: A PATCH-IMAGE BASED CLASSIFICATION APPROACH FOR DETECTION OF WEEDS IN SUGAR BEET CROP.

AUTHOR: S.IMRAN MOAZZAM, UMAR S.KHAN, WAQAR S,JAVID IQBAL, AND AMIR HAMZA.(2021)

Weeds affect crops health as it shares water and nutrients from the soil, as a result it decreases crop yield. Manual weedicide spray through bag-pack is hazardous to human health. Localised autonomous weedicide spray through aerial spraying units can help save water, weedicide chemicals and effect less on human health. Such systems require multi-spectral cues to classify crop, weed, and soil surface. Our focus in this paper is on the detection of weeds in the sugar beet crop, using air-borne multispectral camera sensors, which is considered as an alternative crop to sugarcane to obtain sugar in Pakistan. We developed a new framework for weed identification; a patch-based classification approach as opposed to semantic segmentation that is more realistic for real-time intelligent aerial spraying systems. Our approach converts a 3-class pixel classification problem into a 2-class crop-weed patch classification problem which in turns improves crop and weed classification accuracy. For classification, we developed a new VGG-Beet convolutional neural network (CNN), which is based on the generic VGG16 (visual graphics group) CNN model with 11 convolutional layers. For experiments, we captured a sugar beet dataset with 3-channel multispectral sensor with a ground sampling distance (GSD) of 0.2 cm/pixel and a height of 4 metres. For better comparison, we used two publicly available sugar beet crop aerial imagery datasets, captured using a 5-channel multispectral sensor and a 4-Channel multispectral sensor with a ground sampling distance of 1 cm and a height of 10 metres. We observed that patch-based methods are more robust to different lighting conditions.

2.4 TITLE: IMPROVISED EXTREME LEARNING MACHINE FOR CROP YIELD PREDICTION

AUTHOR: SWATI VASHISHT, PRAVEEN KUMAR, MUNESH CHANDRA.(2021)

Machine learning is effectively involved in crop yield prediction as a decision support tool including which crops to grow in a specific region and how to increase their yield in the reaping and growing season of the crops. A number of deep learning techniques, neural network architectures and prediction models have been employed to assist crop yield prediction research. This study proposes an extreme learning machine to predict crop yield. We performed data pre- processing using Kalman Filter Algorithm. Certain features have been extracted using Linear Discriminant Analysis and crop prediction is done by using an improved version of Extreme Linear Machine. Yield of rice crop has been predicted based upon geography, season and area of cultivation. Improving agricultural yield forecasts is an important component of national economics, and it is a topic that many agro-meteorologists are interested in researching. The most difficult aspect of agricultural yield prediction is choosing an effective algorithm. To overcome such a challenge, an efficient and optimal machine learning technique will be used in this work for achieving an effective performance in crop yield prediction.These performance indicators are as follows: The estimated proportion of accurately anticipated observations to total observations is called accuracy. Future research should look into incorporating more data values into the models and using yield predictions to make appropriate decisions.

2.5 TITLE: PREDICTION OF CROP YIELD BASED ON SOIL MOISTURE USING MACHINE LEARNING ALGORITHMS.

AUTHOR: SINDHU MADHURI G, SAMEER PAUDEL, PRASANTH GIRI, SHANTHANU BAGATHUR KARKI. (2020)

Agriculture planning is playing an important role as the economic growth is very high day by day in our daily life. There is a lot of research study going on as there are few important issues like soil nutrients, crop prediction, farming system, crop monitoring in agriculture with modern farming systems. Crop prediction and crop monitoring is the main factor to produce good quality of crops for farmers to predict crop yield based on soil moisture. Prediction of crop yield includes forecasting factors like temperature, humidity, rainfall, etc., and crop yield based on soil moisture includes a few measures like NPK (Nitrogen, Phosphorus and potassium) and pH values using various sensors. Machine learning (ML) is a useful decision-making model for estimating crop yields, and also for deciding what crops to plant and what to do during the crop's growing seasons. To aid agricultural yield prediction studies, a number of analytical techniques have been used. In this study Farmers can predict or come to a decision on the type of soil moisture values; Farmers can choose the type of crop they want to sow. In this paper, Author proposed a decision tree supervised machine learning algorithm to improve the results for the prediction of crop yield based on soil moisture parameters to achieve better error rate and accuracy for economic growth. It also includes the few machine learning algorithms which are discussed in the literature survey. Furthermore, the Author highlighted the proposed system in methodology, and compared the analysis in results to give it a balanced view.

2.6 OVERVIEW OF LITERATURE SURVEY

Table 2.6 Overview of Literature Survey

TITLE	AUTHOR	YEAR	MERITS	DEMERITS
Crop yield prediction using machine learning algorithm.	Iswariya, Nagapooja, Ragavi.	2022	Svm method is used to classify crop data CNN	SVM may not be able to learn complex patterns and relationships in the data
Crop and weed classification in aerial imagery sesame crop field using patch based deep learning models.	Umars.khan, Waqars.qureshi , Wahir namas.	2022	Patch image based classification approach (CNN) and dataset was captured using agrocam.	CNNs require large amounts of labeled training data to learn effectively
A patch-image based classification approach for detection of weeds in sugar beet crop.	S.Imran moazzam, Umar s.khan, Waqar s,javid iqbal, and Amir hamza.	2021	developed a new(visual graphics group) vgg-beet	requiring large amounts of memory and computational resources.
Improvised extreme learning	Swati vashisht,	2021	Deep learning techniques,	Deep learning models require

machine for crop yield prediction	Praveen kumar, Munesh chandra.		neural network architecture and prediction models.	large amounts of labelled training data to learn effectively
Prediction of crop yield based on soil moisture using machine learning algorithms.	Sindhu madhuri g, Sameer paudel, Prasanth giri, Shanthanu bagathur karki	2020	Artificial neural network (Ann) Machine learning algorithm	requiring large amounts of computational resources to train effectively

CHAPTER 3
SYSTEM ANALYSIS

3.1 EXISTING SYSTEM

A crop yield prediction model based on CNN and Geographical Index. The existing model had an issue with agricultural drifts for crop cultivation that were not compatible with environmental elements such as temperature, weather, and soil condition. BPNN was utilized to train the created CNN model that utilized spatial characteristics as input for error prediction. The created model had the advantage of being deployed on a real-time dataset derived from legitimate geospatial resources. However, while the new model reduced relative error, it decreased crop yield forecast efficiency.The previous model employed SVM to classify crop data based on the texture, shape, and colour of patterns on the sick surface since it includes a clear perception of the faults. A previously utilised technology, CNN, reduced the relative inaccuracy as well as the crop production forecast. Similarly, an existing model that combined a time series model with a Back Propagation Neural Network (BPNN) and used a smaller dataset size had inferior performance since fewer samples were used for prediction. In the realm of selection stability and precision, machine learning methods were used.ML has several useful techniques for determining the input and output link in yield and crop prediction.Data cleaning and processing, missing value analysis, exploratory analysis, and model creation and evaluation were all part of the analytical process. Finally, we use a machine learning method to predict the crop, with varying outcomes. This leads to some of the following crop forecast insights. Because this system will cover the most sorts of crops, farmers will be able to learn about crops that have never been farmed before and will be able to see a list of all possible crops, which will aid them in deciding which crop to cultivate. Furthermore, this method takes into account

previous data production, allowing the farmer to gain insight into market demand and costs for particular crops. The user-friendly web page built for estimating crop yield can be utilised by any user with their choice of the crop by giving climate data for that location. Agriculture played a key part. This study examines the many agricultural strategies that employ machine learning, as well as their benefits and drawbacks.

3.2 DISADVANTAGES:

Lack of interpretability: ML models can be very complex and difficult to interpret, making it difficult to understand how they arrived at a particular prediction or decision.This can make it challenging to identify and correct errors or biases in the model.

Data quality issues: ML models are highly dependent on the quality of the data used to train them. If the data is incomplete, noisy, or biassed, the model's performance may suffer.

3.3 PROPOSED SYSTEM

The proposed work is to create a machine learning (ML) model that can predict good and patchy crops in a dataset. The goal is to develop a model that can assist farmers in agriculture by providing predictions that can help them make informed decisions about crop management.To accomplish this, the first step would be to collect and preprocess the relevant data. This would include data on crop yields, weather conditions, soil moisture, and other relevant variables. The data would then be cleaned, normalised, and split into training and testing sets.

Next, a suitable ML algorithm would be selected and trained using the training data. The model would be optimised to minimise errors and improve accuracy, and various techniques such as cross-validation and hyperparameter tuning would be used to ensure that the model is performing optimally.Once the model is trained, it can be used to make predictions on new data. Farmers can input data on current weather conditions, soil moisture, and other relevant factors, and the model can provide predictions on the likelihood of a good or patchy crop. This can help farmers make informed decisions about irrigation, fertilisation, and other crop management practices.Overall, the proposed work aims to develop an ML model that can assist farmers in agriculture by providing accurate predictions on crop yields. By leveraging the power of ML, farmers can make better-informed decisions and improve their overall crop management practices.

3.4 ADVANTAGES:

1. **Improved accuracy:** ML models can often achieve higher accuracy than traditional rule-based or statistical models, particularly for complex problems and large datasets. This can lead to better decision-making and improved outcomes.

2. **Scalability:** ML models can be trained on large datasets and can handle a wide range of data types and formats, making them highly scalable and flexible.

3. **Automation:** ML models can automate many tasks that would otherwise require human intervention, such as data preprocessing, feature selection, and model selection. This can save time and improve efficiency.

4. **Adaptability:** ML models can adapt to changing data and conditions, making them well-suited for dynamic and evolving environments.

5. **Exploration of complex relationships:** ML models can uncover complex relationships and patterns in data that may be difficult or impossible to detect using traditional methods. This can lead to new insights and discoveries.

6. **Continuous learning:** ML models can continue to learn and improve over time, as they are exposed to new data and feedback. This can lead to ongoing improvements in accuracy and performance.

COMPARISON SCENARIO

Table 3.1 comparison scenario

Existing systems	Proposed system
ML models have been trained and tested on large datasets, and their performance has been evaluated	ML models can be tailored to the specific needs of a particular application or domain, leading to better performance and accuracy.
It may not be customizable to the specific needs of a particular application or domain.	It will be customizable to the specific needs of a particular application or domain
It can overfit to the specific dataset and may not generalise well to new data.	It can be designed to be transparent, making it easier to interpret how they arrive at their predictions.
It can be used as a starting point for transfer learning, where the model is fine-tuned on a new dataset for a specific task.	It can introduce new approaches and techniques that can improve performance and accuracy in image classification tasks.
These models may be difficult to interpret, making it challenging to understand how they arrive at their predictions.	These models will easy to interpret, making it compatible to understand how they arrive at their predictions

CHAPTER 4

SYSTEM REQUIREMENTS

4.1 HARDWARE REQUIREMENTS

Processor : Intel core processor 2.6.0 GHZ

RAM : 1GB

Hard disk : 160 GB

Compact Disk : 650 Mb

Keyboard : Standard keyboard

Monitor : 15 inch colour monitor

4.2 SOFTWARE REQUIREMENTS

Operating system : Windows OS

Front End : JUPYTER NOTEBOOK

CHAPTER 5
SOFTWARE DESCRIPTION

5.1 FRONT END: JUPYTER NOTEBOOK

Jupyter Notebook is an open-source web application that allows users to create and share documents containing live code, equations, visualisations, and narrative text. Jupyter Notebook provides an interactive computing environment for a variety of programming languages, including Python, R, and Julia.

When it comes to creating a front end using Jupyter Notebook, there are several options available. One popular approach is to use widgets, which are interactive GUI components that allow users to manipulate data and parameters. Jupyter Notebook provides a built-in widget library called ipywidgets, which includes a wide range of widgets such as sliders, dropdowns, and text boxes.

Another option for creating a front end in Jupyter Notebook is to use a dashboarding tool such as Voilà. Voilà allows users to convert Jupyter Notebook files into standalone web applications with a custom-designed user interface. With Voilà, users can create interactive dashboards and data visualisations that can be easily shared with others.

Overall, Jupyter Notebook is a powerful tool for creating front-ends and interactive applications. Whether you choose to use widgets or a dashboarding tool like Voilà, Jupyter Notebook provides a flexible and user-friendly environment for building interactive data-driven applications.

5.2 FEATURES OF JUPYTER NOTEBOOK

Code execution: Jupyter Notebook allows users to write and execute code in a variety of programming languages, including Python, R, and Julia.

Interactive data visualisation: Jupyter Notebook provides built-in support for data visualisation libraries such as Matplotlib and Seaborn, allowing users to create and display charts, graphs, and other visualisations directly within their notebooks.

Collaboration: Jupyter Notebook allows multiple users to collaborate on the same notebook in real-time, making it an ideal tool for team projects and shared research.

Rich text support: Jupyter Notebook supports Markdown syntax, allowing users to easily create rich, formatted text documents that can include images, links, and other media.

Notebook sharing: Jupyter Notebook allows users to share their notebooks with others by exporting them as HTML, PDF, or other formats.

Customization: Jupyter Notebook allows users to customise the appearance and functionality of their notebooks using extensions and themes.

Integration with other tools: Jupyter Notebook can be integrated with a wide range of other tools and platforms, including Git, GitHub, and cloud computing platforms like Google Colab and Microsoft Azure.

5.3 USES OF JUPYTER NOTEBOOK

Data analysis: Jupyter Notebook is often used for data analysis, as it provides a powerful and flexible environment for working with data in a variety of formats.

Scientific computing: Jupyter Notebook is commonly used for scientific computing, as it provides support for a range of programming languages and libraries commonly used in scientific research.

Machine learning: Jupyter Notebook is a popular tool for machine learning, as it provides support for popular machine learning libraries such as TensorFlow and PyTorch.

Education: Jupyter Notebook is widely used in education, as it provides a flexible and interactive environment for teaching and learning programming and data analysis.

Research: Jupyter Notebook is often used in research, as it provides a collaborative and transparent environment for documenting and sharing research workflows and results.

5.4 JUPYTER NOTEBOOK FILES

Notebook files: Notebook files have a .ipynb file extension and contain the code, text, and visualisations created by the user. These files are saved in JSON format and can be executed, edited, and shared with others.

Kernel files: Kernel files are the computing engines that execute the code in the notebook files. When a user runs a notebook, the code is sent to the kernel, which executes it and sends the output back to the notebook for display. Jupyter Notebook supports kernels for a variety of programming languages, including Python, R, and Julia.

CHAPTER 6
SYSTEM DESIGN

6.1 SYSTEM ARCHITECTURE

System architecture is the conceptual model that defines the structure, behaviour, and more views of a system. An architecture description is a formal description and representation of a system, organised in a way that supports reasoning about the structures and behaviours of the system

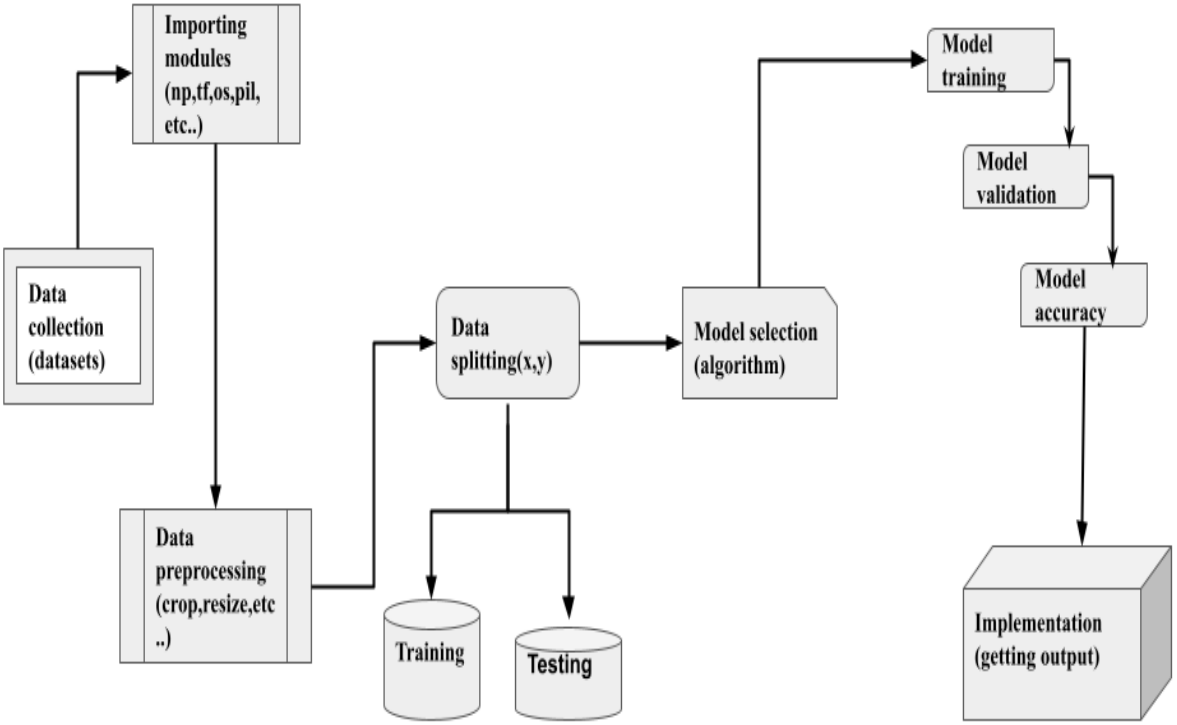
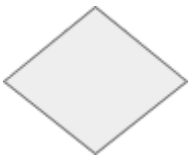


Fig 6.1 System Architecture

6.2 DATA FLOW DIAGRAM

A data flow diagram (DFD) is a graphical representation of the flow of data ugh a system or process. It is used to model the flow of data, as well as the processes, entities, and data stores involved in a system or process. DFDs are often used in software engineering and systems analysis to illustrate the data flow and processing steps in a complex system.

Table 6.1 Data flow diagrams

SYMBOLS	DESCRIPTION
	Data collection.,where collected data will be stored and accessed.
	Data preprocessing ,where images can resize ,crop and ready for use.
	Data splitting,where data will be split into test and train.
	Train and Test sets, where data accessed and save.
	Model Implementation,where finally all the setup will be executed.
	Connectors,where connect the link between other test cases.

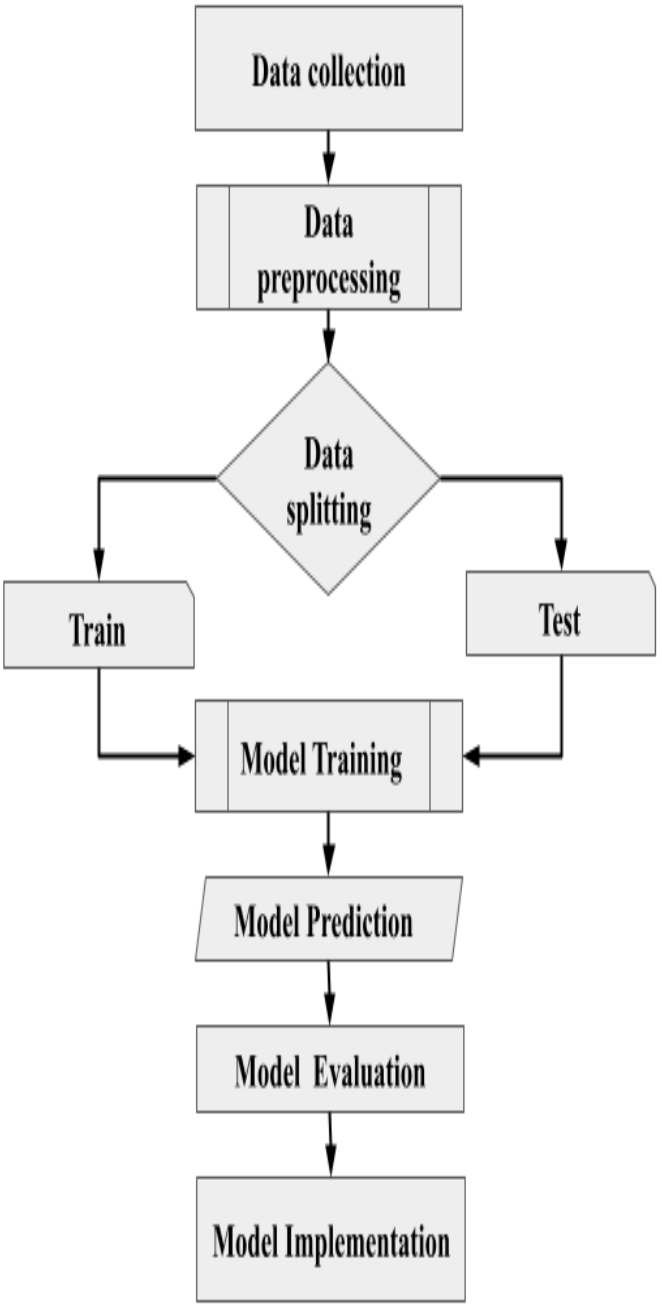


FIG 6.2 Data Flow diagram

CHAPTER 7
SYSTEM IMPLEMENTATION

System implementation is the process of integrating a software system or application into an existing environment or infrastructure. It involves putting the software into use and making it available to users, while ensuring that it operates effectively and meets the needs of the organisation or individuals using it.

7.1 TENSORFLOW

TensorFlow is an open-source software library developed by Google Brain Team for building and training machine learning models, particularly deep neural networks. It provides a powerful framework for building, training, and deploying ML models across a range of applications and platforms.

At its core, TensorFlow is based on a system of data flow graphs, which are directed graphs that describe the flow of data through a system. In TensorFlow, the data flow graph represents the computational model of the ML system, with nodes representing mathematical operations and edges representing the data that flows between them.

TensorFlow supports a wide range of ML models, including neural networks, decision trees, and logistic regression. It also provides a range of tools and resources for building and training models, including a high-level API for easy model building and a low-level API for more fine-grained control over the model architecture.

One of the key advantages of TensorFlow is its ability to scale to large datasets and complex models. It can be used to train models on a single machine or distributed across a cluster of machines, making it well-suited for use in enterprise and cloud computing environments.

TensorFlow is also highly customizable, with a range of extensions and libraries available to support specific use cases and applications. These include TensorFlow Lite for running ML models on mobile and embedded devices, TensorFlow.js for running ML models in web browsers, and TensorFlow Extended (TFX) for building end-to-end ML pipelines.

Overall, TensorFlow is a powerful and flexible ML framework that is widely used by researchers and developers to build and deploy ML models across a range of applications and platforms. Its scalability, customization options, and ease of use make it a popular choice for both academic and commercial applications.

7.2 PREPROCESSING

Preprocessing in machine learning involves a set of techniques used to prepare raw data for use in a machine learning model. The quality of data is crucial to the performance of the model, so preprocessing is an important step in the machine learning pipeline.

Data cleaning: This involves identifying and correcting errors or inconsistencies in the data, such as missing values, outliers, or incorrect formatting.

Feature scaling: This involves rescaling the values of features in the dataset to a common scale to ensure that no single feature dominates the model's output.

Feature encoding: This involves converting categorical data into numerical values that can be used in the model. This can be done through techniques such as one-hot encoding, label encoding, or ordinal encoding.

Feature selection: This involves selecting a subset of the most relevant features from the dataset to use in the model. This can help to reduce the dimensionality of the dataset and improve the model's performance.

Data augmentation: This involves generating additional data from the existing dataset to improve the model's performance. This can be done through techniques such as image flipping, rotation, or zooming.

Preprocessing: It is an iterative process that involves testing and refining the techniques used until the desired performance is achieved. The goal of preprocessing is to ensure that the data is accurate, consistent, and in a format that can be easily used by the machine learning model.

Overall, preprocessing is a critical step in the machine learning pipeline that can have a significant impact on the performance of the model. Proper preprocessing can help to ensure that the model is accurate, robust, and capable of making useful predictions or decisions.

7.3 ALGORITHM USED

CNN stands for Convolutional Neural Network. It is a type of artificial neural network that is commonly used in image recognition and computer vision applications. CNNs are designed to automatically learn and extract features from images, making them a powerful tool for tasks such as object recognition, image classification, and segmentation.

The key components of a CNN are convolutional layers, pooling layers, and fully connected layers. Convolutional layers apply a set of learnable filters to the input image, generating a set of feature maps that highlight important features within the image. Pooling layers downsample the feature maps to reduce their size and make the model more computationally efficient. Finally, fully connected layers are used to generate the final output of the model, such as a classification label.

One of the key advantages of CNNs is their ability to automatically learn features from images without the need for manual feature engineering. This makes them particularly effective for tasks where the relevant features may be difficult to identify or quantify. CNNs can also be trained on large datasets, allowing them to learn a wide range of features and make accurate predictions on new, unseen images.

Overall, CNNs are a powerful tool for image recognition and computer vision tasks, allowing machines to learn and recognize important features within images with high accuracy and speed.

1. **Input Layer:** The first layer of the network takes in the input image, which is represented as a 2D matrix of pixel values.

2. **Convolutional Layers:** The convolutional layers apply a set of learnable filters to the input image, generating a set of feature maps that highlight important features within the image. Each filter slides over the input image, performing a dot product operation at each location and producing a feature map. The size and number of filters used can vary depending on the requirements of the model.

3. **Activation Function:** After each convolutional layer, an activation function is applied to the feature maps. The most commonly used activation function is the Rectified Linear Unit (ReLU), which sets all negative values to zero and preserves positive values.

4. **Pooling Layers:** The pooling layers downsample the feature maps to reduce their size and make the model more computationally efficient. The most commonly used pooling operation is max pooling, which takes the maximum value of a group of pixels within a region.

5. **Fully Connected Layers:** The fully connected layers are used to generate the final output of the model, such as a classification label. The flattened output from the previous layers is passed through a set of fully connected neurons, each of which is connected to all of the neurons in the previous layer.

6. **Softmax Function:** The softmax function is applied to the final output of the model to convert the output into a probability distribution over the possible classes.

7. **Loss Function:** The loss function is used to measure the difference between the predicted output and the actual output. The most commonly used loss function for classification tasks is cross-entropy loss.

8. **Optimization:** The optimization algorithm is used to update the weights of the model to minimise the loss function. The most commonly used optimization algorithm is stochastic gradient descent (SGD).

7.4 TRAINING

Model Training: Once the model architecture has been selected, the next step is to train the model on the training data. The model is trained by adjusting the model parameters to minimise a loss function that measures the difference between the predicted output and the actual output. This process involves backpropagation and gradient descent, which adjusts the weights of the model to minimise the loss function.

7.5 MODEL FITTING

Model fitting is an iterative process that involves selecting an appropriate model architecture, tuning hyperparameters, training the model, evaluating its performance, and deploying it in a production environment. Each step in the process is critical to ensuring that the model is accurate, robust, and capable of making reliable predictions on new data.

7.6 MODEL EVALUATION

Model evaluation is a critical step in the machine learning workflow, as it helps to ensure that the model is accurate, robust, and capable of making reliable predictions on new data. By using techniques such as train/test split, cross-validation, metrics, confusion matrices, and learning curves, we can gain a better understanding of how well the model is able to generalise to new data, and identify any areas where it may need to be improved.

7.7 TESTING

Model evaluation is a critical step in the machine learning workflow, as it helps to ensure that the model is accurate, robust, and capable of making reliable predictions on new data. By using techniques such as train/test split, cross-validation, metrics, confusion matrices, and learning curves, we can gain a

better understanding of how well the model is able to generalise to new data, and identify any areas where it may need to be improved.

Test Set: The test set is a dataset that is held out from the training process, and is used for evaluating the performance of the trained model. It should be representative of the real-world data that the model will encounter, and should be large enough to provide a statistically significant evaluation

7.8 ACCURACY CHECK

Accuracy check is a process of evaluating the performance of a machine learning model by comparing its predicted output to the actual output on a test dataset. Accuracy is a common metric used to measure the performance of a classification model, and it represents the proportion of correctly classified instances out of all the instances in the dataset.

To perform an accuracy check, the test dataset is typically divided into two parts: a set of input features and a set of corresponding output labels. The model is then used to make predictions on the input features, and the predicted output labels are compared to the actual output labels to calculate the accuracy.

While accuracy is a useful metric for evaluating the performance of a model, it should be used with caution in some cases. For example, if the dataset is imbalanced (i.e., one class has many more instances than the other), then a model that always predicts the majority class can have a high accuracy but still be a poor classifier. In such cases, other metrics such as precision, recall, and F1-score may be more appropriate.

CHAPTER 8
SYSTEM TESTING

8.1 TESTING

Testing is the process of evaluating a trained model on a dataset that is separate from the dataset used for training. The goal of testing is to assess how well the model generalises to new, unseen data. Here are some key aspects of testing in machine learning:

Test Set: The test set is a dataset that is held out from the training process, and is used for evaluating the performance of the trained model. It should be representative of the real-world data that the model will encounter, and should be large enough to provide a statistically significant evaluation.

Performance Metrics: To evaluate the performance of the model, we use metrics such as accuracy, precision, recall, F1-score, and ROC AUC for classification problems, and mean squared error, mean absolute error, and R-squared for regression problems. These metrics provide a quantitative measure of how well the model is able to make predictions on new data.

- **Overfitting:** Overfitting occurs when a model performs well on the training set, but poorly on the test set. This can happen when the model becomes too complex and starts to memorise the training set, rather than learning general patterns that can be applied to new data. To avoid overfitting, we can use techniques such as regularisation, early stopping, and dropout.

- **Cross-Validation:** Cross-validation is a technique that can be used to assess the performance of the model on multiple test sets. By splitting the

dataset into multiple folds and evaluating the model on each fold, we can get a more robust estimate of the model's performance.

- **Model Interpretation:** Testing can also be used to interpret the model and gain insights into how it is making predictions. Techniques such as feature importance, partial dependence plots, and SHAP values can help to explain the relationship between the input features and the model's predictions.

Overall, testing is a critical step in the machine learning workflow, as it helps to ensure that the model is accurate, robust, and able to make reliable predictions on new data. By using techniques such as performance metrics, overfitting prevention, cross-validation, and model interpretation, we can gain a better understanding of how well the model is able to generalise to new data, and identify any areas where it may need to be improved.

8.2 TEST CASES

Table 8.1 Test cases

ID	Test cases	Task	Execution	Expected output	Status
Ml_01	Tensorflow	Image classification	pip install	successful	pass
Ml_02	keras	Sequence classification	import	successful	pass
Ml_03	Pillow	Image handling	import	successful	pass

Ml_04	scipy	Calculation tasks	import	successful	pass
Ml_05	dir	Creating directory	Assigning the values to the directory	Read dir without errors	pass
Ml_06	Fetch dir	Fetching dir together and read	Check by call dir	Read dir without errors	pass
Ml_07	labels	Creating labels for each dir	Check by call label	Read labels without errors	pass
Ml_08	numpy	Matrix multiplication	import	successful	fail
Ml_09	display	Display the image from the path	Read by call pil	Display without errors	pass
Ml_10	path_lib	Read the path in the dir	Import	Read without errors	pass
Ml_11	Model splitting	Split the overall data into test and train	Allocating data in train and test sets	Read without errors	pass
Ml_12	Conv2D	Using CNN	Resize the	Run without	pass

			image axis	errors	
Ml_13	Parameter passing	Read different parameter	Read the expect path from the dir	Run without errors	pass
Ml_14	Aug	Data augumentation	Image processing	Run without errors	pass
Ml_15	Model testing	Compile the dir to output	Passing different parameters	Run and give expected output	pass
Ml_16	accuracy	Show accuracy of the model	Run and give expected accuracy	Run without error	pass

CHAPTER 9

SYSTEM STUDY

SOURCE CODE

```
#importing requiring modules
import matplotlib.pyplot as plt
import numpy as np
import scipy
import cv2
import os
import PIL
import tensorflow as tf
from PIL import Image

from pathlib import Path
from tensorflow import keras
from tensorflow.keras import layers
from tensorflow.keras.models import Sequential

data_dir = Path("/kaggle/input/wheat-research-new-datase")

data_dir
data_list = list(data_dir.glob('*/*.jpg'))
data_list

Wheat_images_dict = {
        'wheat' : list(data_dir.glob('wheat/*')),
```

```
        'crop patches' : list(data_dir.glob('crop patches/*')),
}
Wheat_labels_dict = {
        'wheat' : 0,
        'crop patches' : 1,
}
PIL.Image.open(Wheat_images_dict['wheat'][56])
from PIL import Image
import matplotlib.pyplot as plt

# Open the images
images = []
for i in range(10):
 img = Image.open(Wheat_images_dict['crop patches'][i])
 images.append(img)

# Display each image one by one
for img in images:
        plt.imshow(img)
        plt.axis('off')
        plt.show()

from PIL import Image
import matplotlib.pyplot as plt

# Open the 10 images
images = []
for i in range(10):
        img = Image.open(Wheat_images_dict['wheat'][i])
```

```
        images.append(img)

# Create a new image that combines the 10 images into a 2x5 grid
width, height = images[0].size
list_image = Image.new('RGB', (width*5, height*2))
for i in range(10):
        x = i % 5
        y = i // 5
        list_image.paste(images[i], (x*width, y*height))

# Display the grid image using matplotlib
for img in images:
        plt.imshow(img)
        plt.axis('off')
        plt.show()

from PIL import Image
import matplotlib.pyplot as plt

# Open the 10 images
images = []
for i in range(10):
        img = Image.open(Wheat_images_dict['wheat'][i])
        images.append(img)

# Create a new image that combines the 10 images into a 2x5 grid
width, height = images[0].size

list_image = Image.new('RGB', (width*5, height*2))
```

```
for i in range(10):
        x = i % 5
        y = i // 5
        list_image.paste(images[i], (x*width, y*height))

# Display the grid image using matplotlib
for img in images:
        plt.imshow(img)
        plt.axis('off')
        plt.show()

Img.shape
cv2.resize(img, (180, 180)).shape
x ,y = [],[]

for Wheat_name, images in Wheat_images_dict.items():
        for image in images:
                img = cv2.imread(str(image))
                resized_img = cv2.resize(img,(180,180))
                x.append(resized_img)
                y.append(Wheat_labels_dict[Wheat_name])

x = np.array(x)
y = np.array(y)

from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test = train_test_split(x, y, random_state = 0)
len(x_train)
len(x_test)
```

```
x_trained_scaled = x_train / 255
x_text_scaled = x_test / 255

num_classes = 5

model = Sequential([
        layers.Conv2D(16, 3, padding = 'same', activation = 'relu'),
        layers.MaxPooling2D(),
        layers.Conv2D(32, 3, padding = 'same', activation = 'relu'),
        layers.MaxPooling2D(),
        layers.Conv2D(64, 3, padding = 'same', activation = 'relu'),
        layers.MaxPooling2D(),
        layers.Flatten(),
        layers.Dense(128, activation = 'relu'),
        layers.Dense(num_classes)
])

model.compile(
        optimizer = 'adam',
        loss = tf.keras.losses.SparseCategoricalCrossentropy(from_logits = True),
        metrics = ['accuracy']
)

model.evaluate(x_text_scaled, y_test)
prediction = model.predict(x_text_scaled)
prediction

score = tf.nn.softmax(prediction[0])
score
```

```
np.argmax(score)
Y_test[3]

data_augmentation = keras.Sequential([
        layers.experimental.preprocessing.RandomZoom(0.1),
        layers.experimental.preprocessing.RandomRotation(0.1),
        layers.experimental.preprocessing.RandomFlip("horizontal",
        input_shape = (180,180,3))
])

plt.axis('off')
plt.imshow(x[0])

num_classes = 5

model = Sequential([
        data_augmentation,
        layers.Conv2D(16, 3, padding = 'same', activation = 'relu'),
        layers.MaxPooling2D(),
        layers.Conv2D(32, 3, padding = 'same', activation = 'relu'),
        layers.MaxPooling2D(),
        layers.Conv2D(64, 3, padding = 'same', activation = 'relu'),
        layers.MaxPooling2D(),
        layers.Dropout(0.2),
        layers.Flatten(),
        layers.Dense(128, activation = 'relu'),
        layers.Dense(num_classes)
])
model.compile(
```

```
        optimizer = 'adam',
        loss = tf.keras.losses.SparseCategoricalCrossentropy(from_logits = True),
        metrics = ['accuracy']
)

model.fit(x_trained_scaled, y_train, epochs = 10)
model.evaluate(x_text_scaled,y_test)

from tensorflow.keras.models import load_model
from tensorflow.keras.preprocessing import image
import numpy as np
model.save("/kaggle/working/wheat.h5")
model = load_model('/kaggle/working/wheat.h5')

X_test

from sklearn.metrics import accuracy_score

# Get the predicted labels for the test images
y_pred = model.predict(x_test)
y_pred = np.argmax(y_pred, axis=1)

# Calculate the accuracy score
accuracy = accuracy_score(y_test, y_pred)

# Print the accuracy score
print('The accuracy score is:', accuracy)

pred_idx = 1
```

```
op = ['crop patches','wheat']
pred_label = op[pred_idx]
pred_idx = 1
op = ['wheat','crop patches']

pred_label = op[pred_idx]
print("Predicted class label:", pred_label)

from PIL import Image
import numpy as np

# Open and resize the image to 180x180 pixels
img = Image.open('/kaggle/input/wheat-and-patch-image-dataset/wheat/Wheat
(35).jpg')
img = img.resize((180, 180))

# Convert the image to a numpy array
img_array = np.array(img)

# Reshape the array to match the expected input shape of the model
img_array = np.reshape(img_array, (1, 180, 180, 3))

# Pass the image array to the model for prediction
prediction = model.predict(img_array)

from PIL import Image
import numpy as np
import matplotlib.pyplot as plt
from PIL import Image
```

```
# Load the image and resize it to the expected size
img = Image.open('/kaggle/input/wheat-research-new-datase/wheat/WHEAT (11).jpg').resize((180, 180))

# Convert the image to a numpy array and reshape it to match the expected input shape of the model
img_array = np.array(img)
img_array = np.expand_dims(img_array, axis=0)

# Pass the image to the model for prediction
prediction = model.predict(img_array)

# Determine the predicted class label
predicted_class = np.argmax(prediction)

# Check the predicted class label and take appropriate action
if predicted_class == 1:
        print('patched')
else:
        print('good')

plt.imshow(img)
plt.show()

from sklearn.metrics import accuracy_score

# Get the predicted labels for the test images
y_pred = model.predict(x_test)
```

```
y_pred = np.argmax(y_pred, axis=1)

# Calculate the accuracy score
accuracy = accuracy_score(y_test, y_pred)

# Print the accuracy score
print('The accuracy score is:', accuracy)
```

CHAPTER 10

SCREENSHOTS

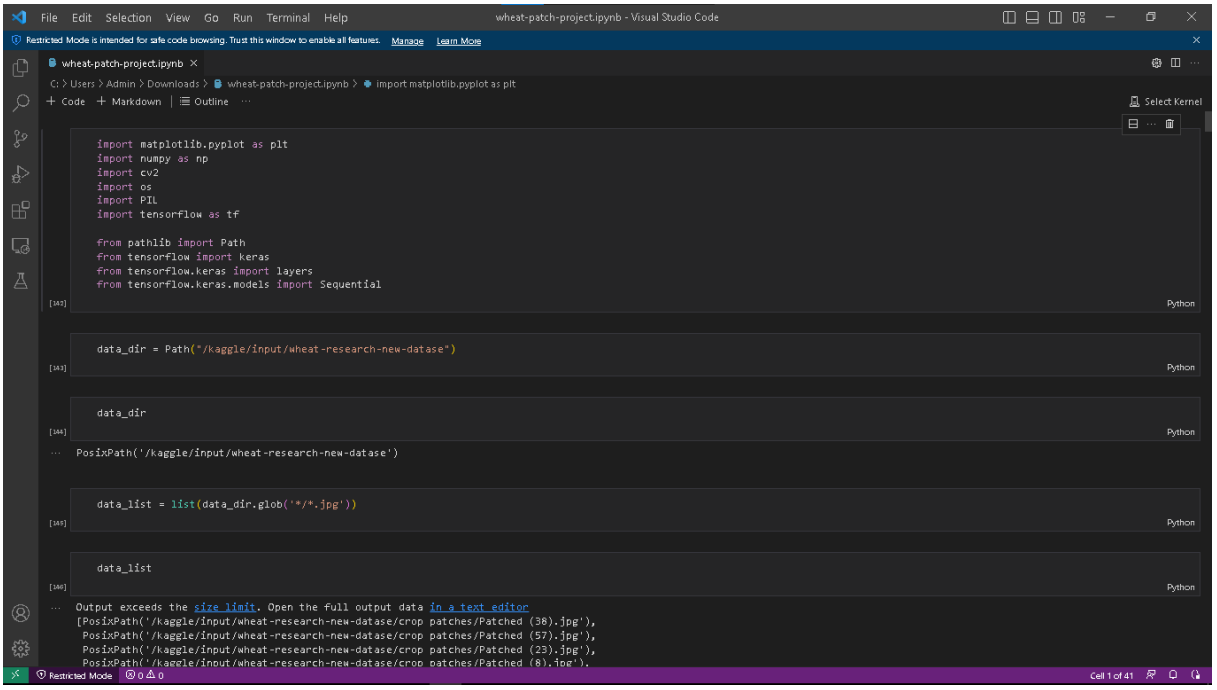


Fig 10.1 Importing modules

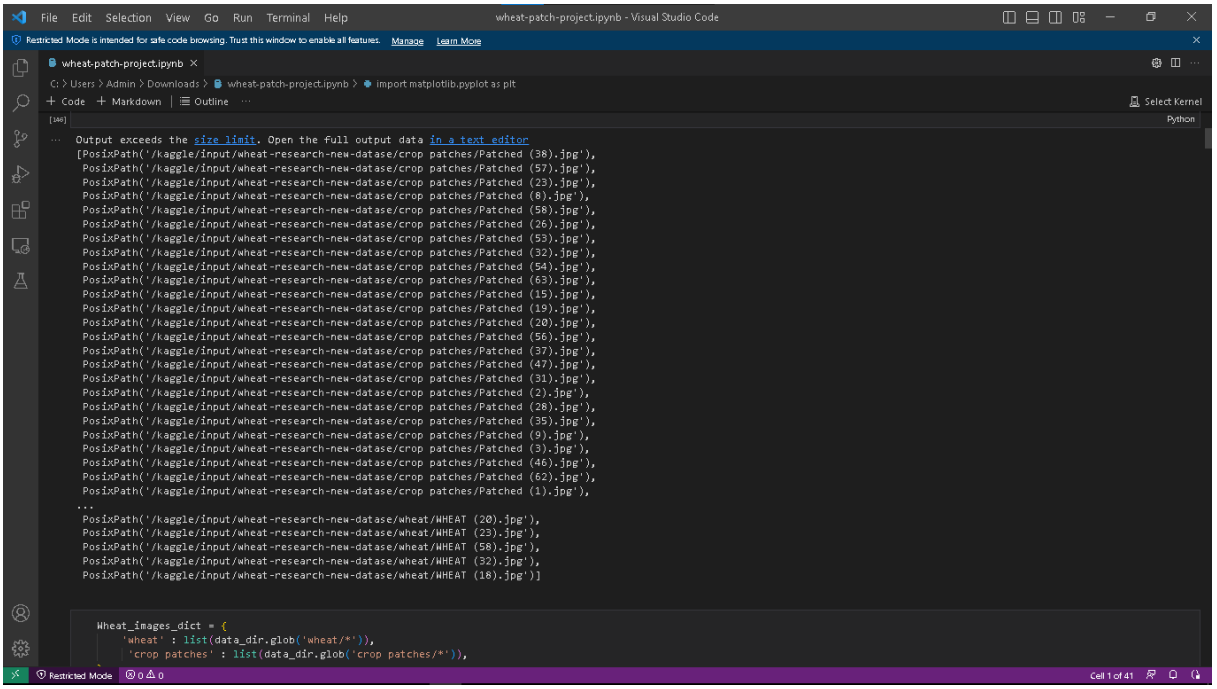


Fig 10.2 Reading dir

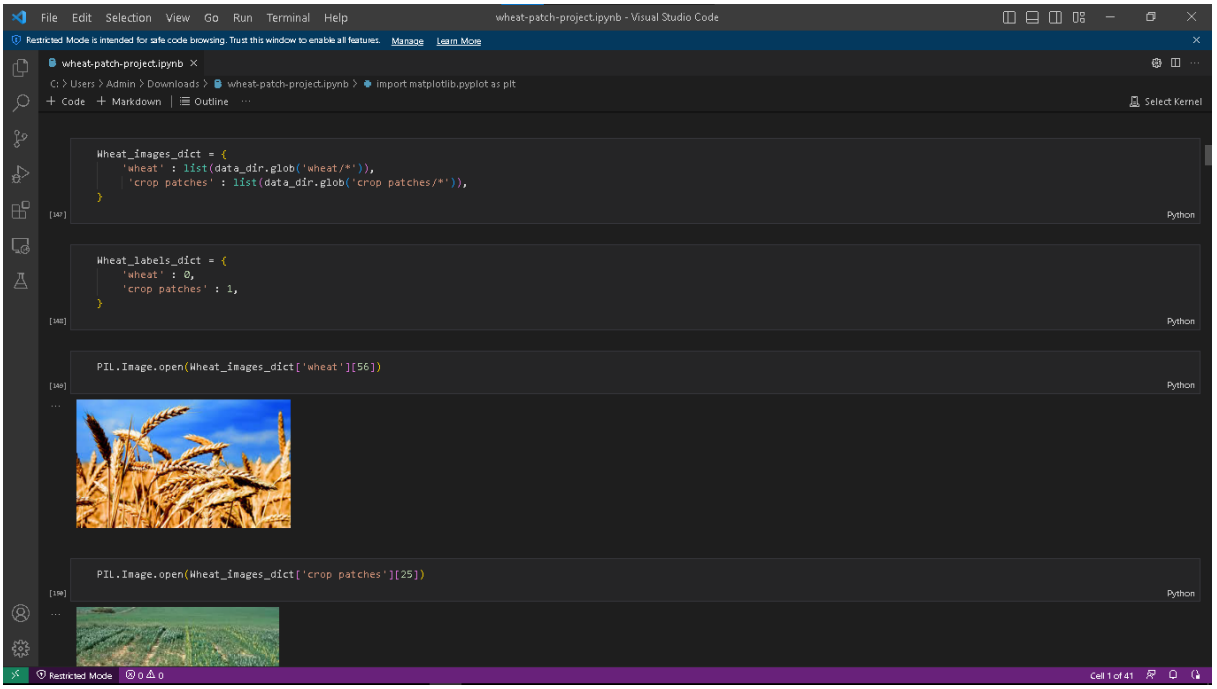


Fig 10.3 Assign dir

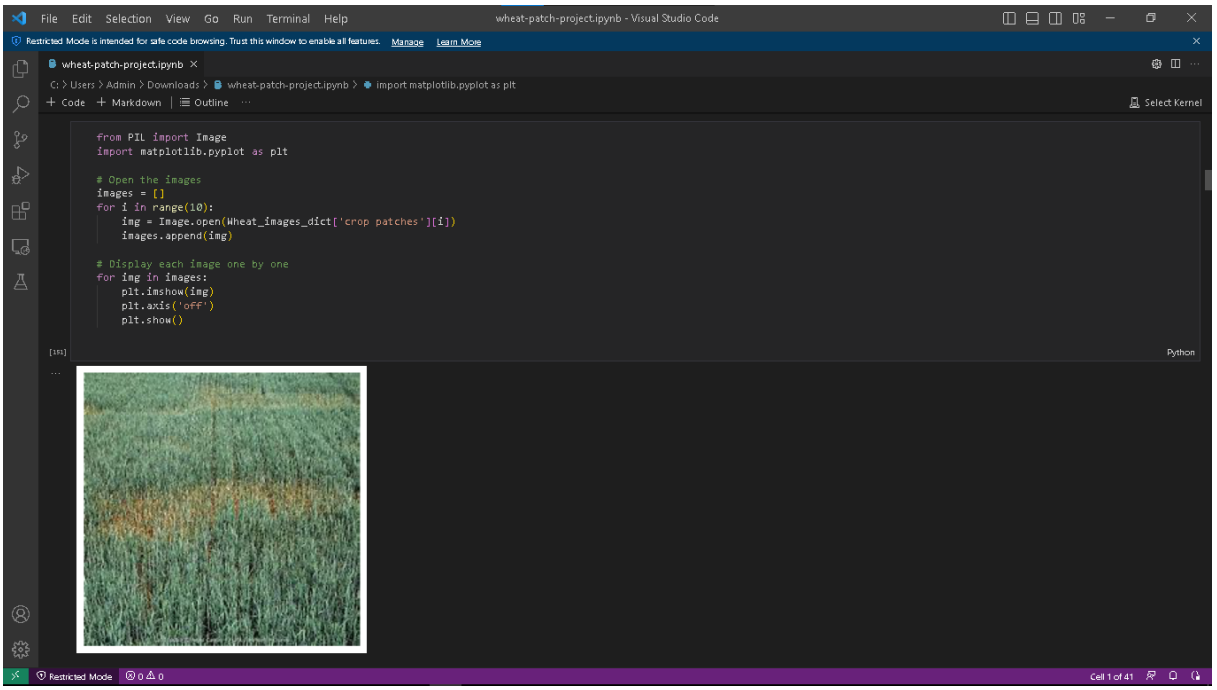


Fig 10.4 Read image using PIL

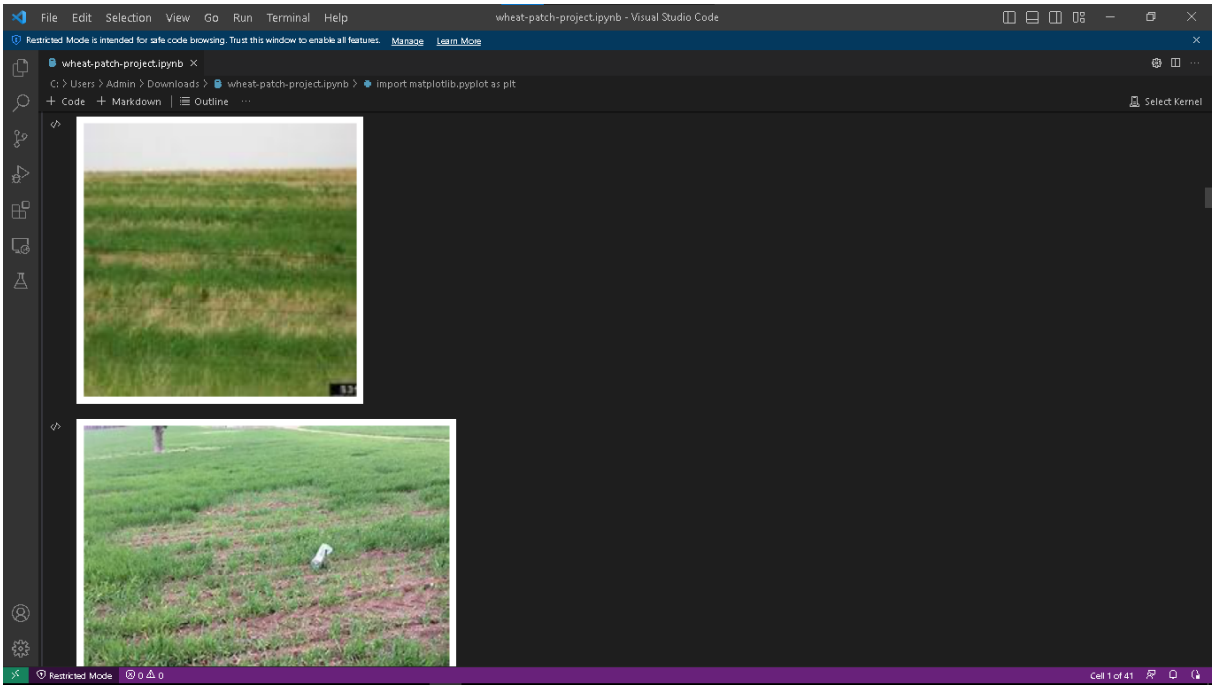


Fig 10.5 Sample images

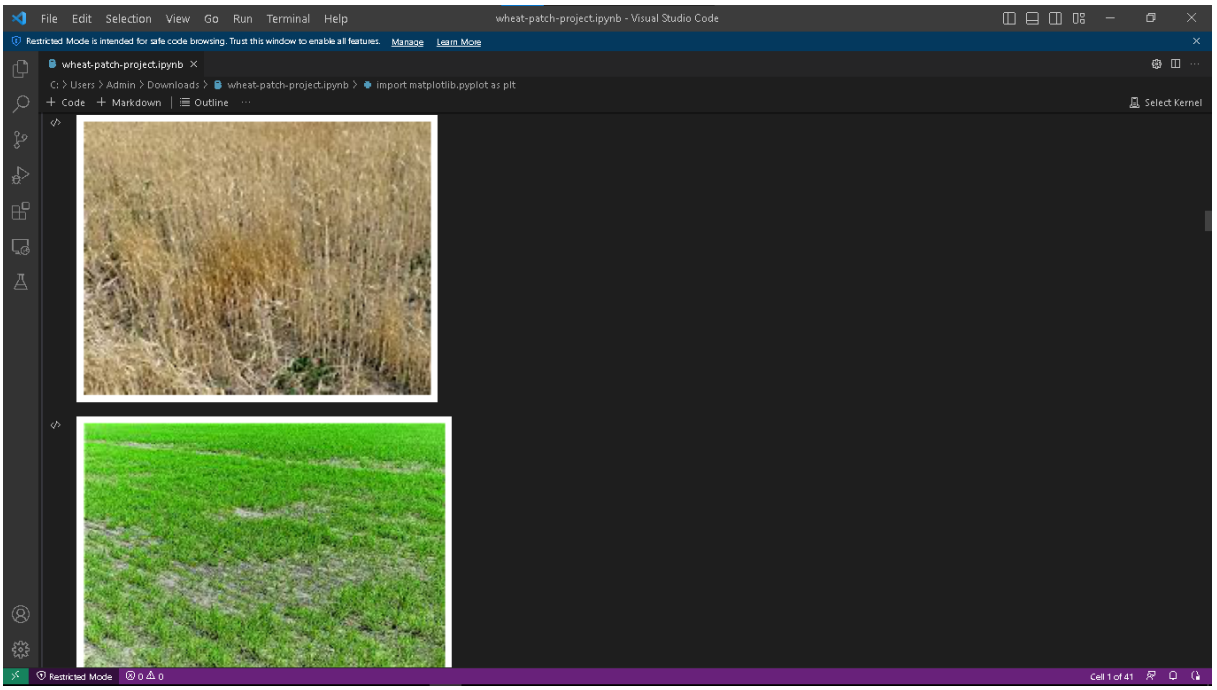


Fig 10.6 Sample images

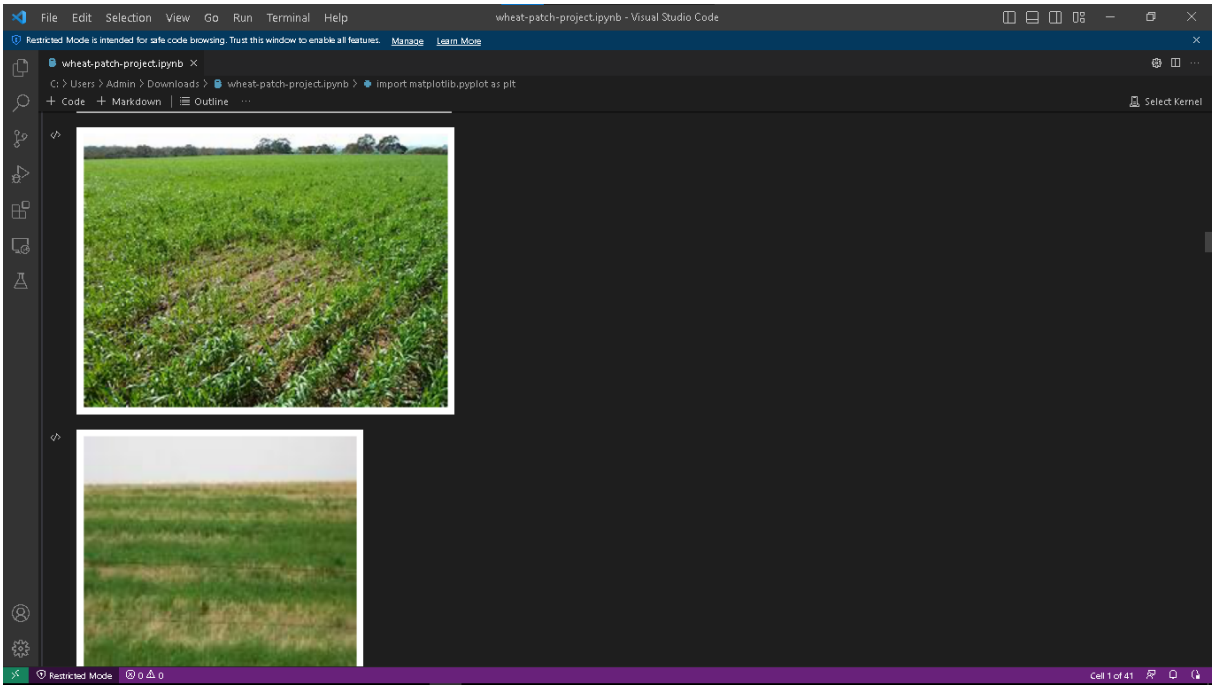


Fig 10.7 Sample images

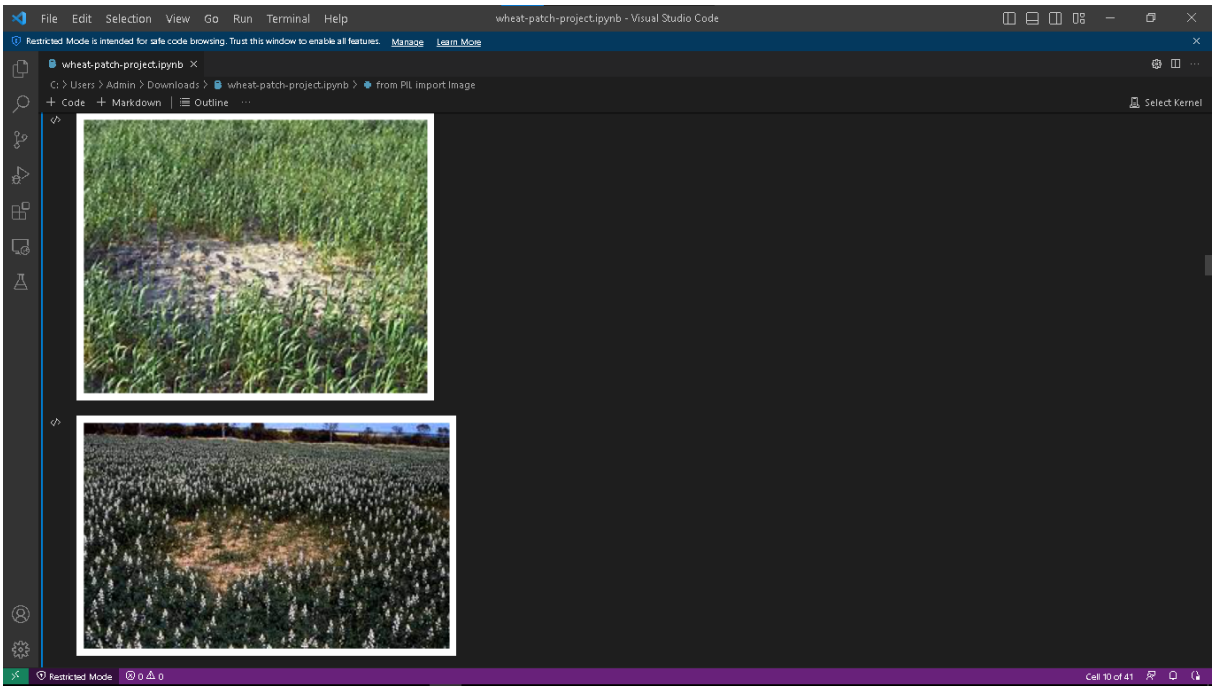


Fig 10.8 Sample images

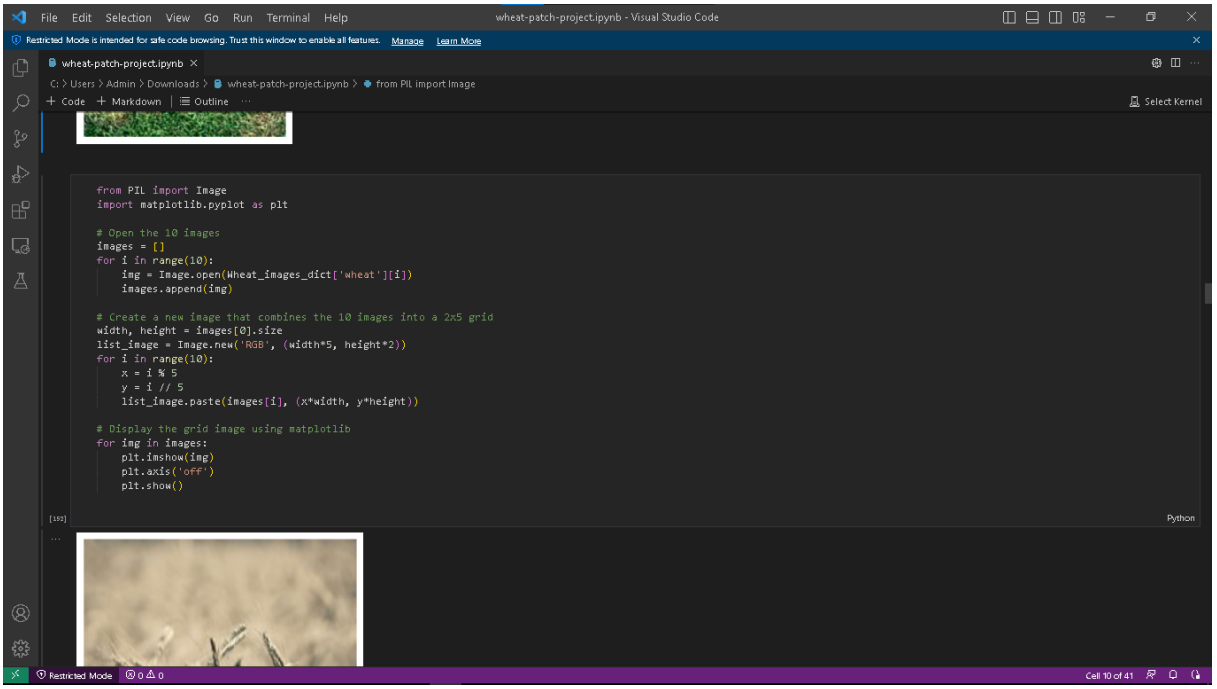


Fig 10.9 Reading second dir

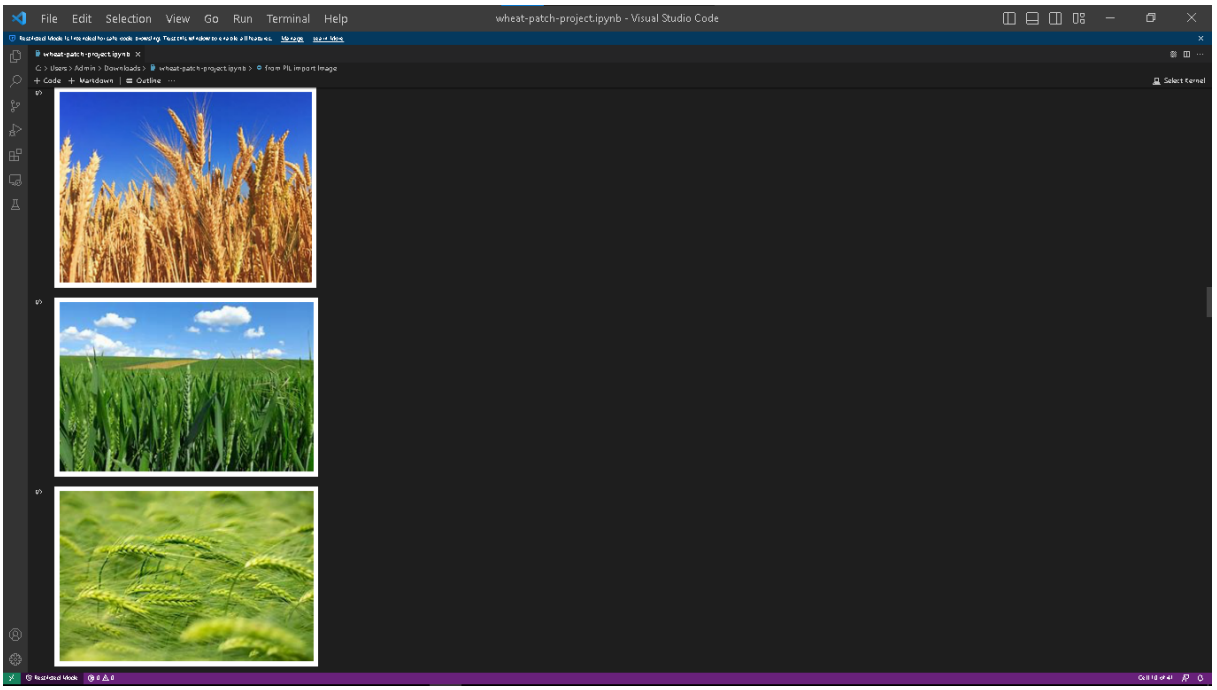


Fig 10.10 Sample images

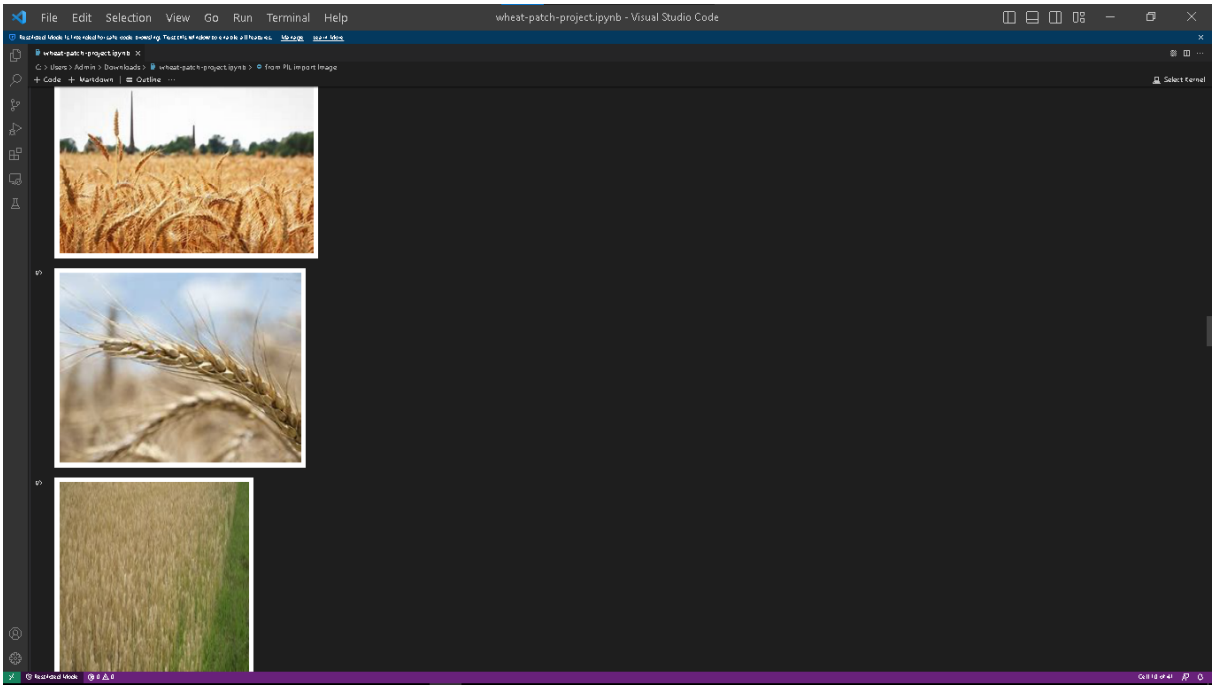


Fig 10.11 Sample images

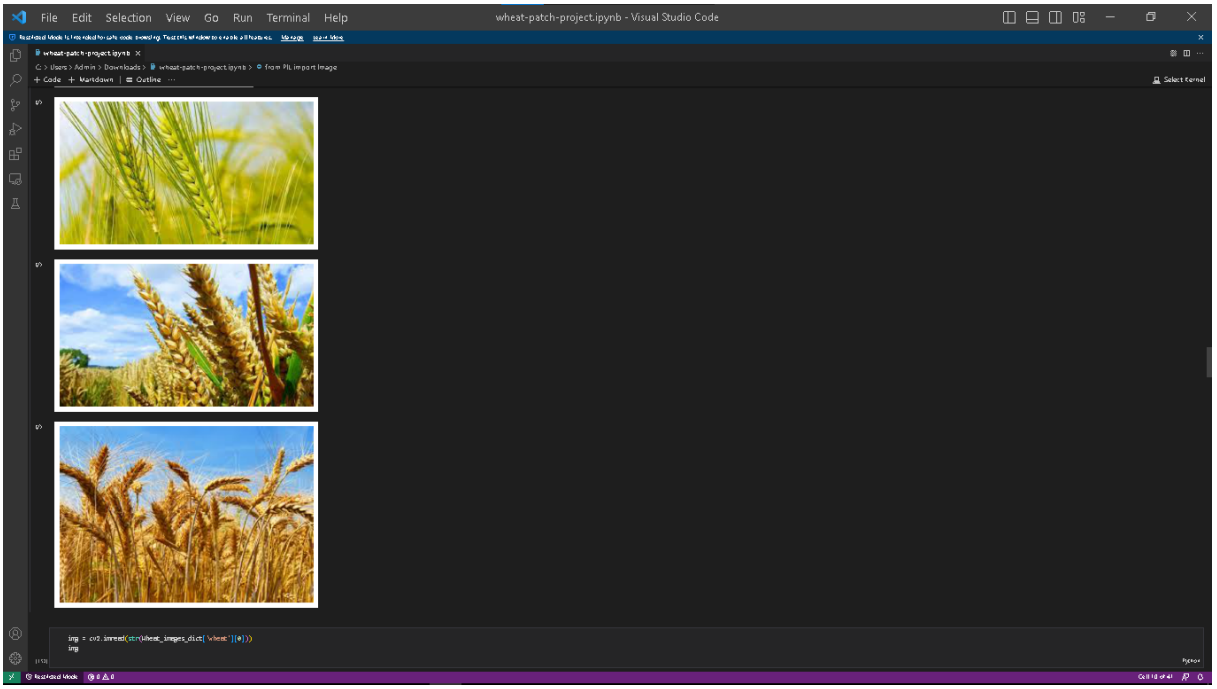


Fig 10.12 Sample images

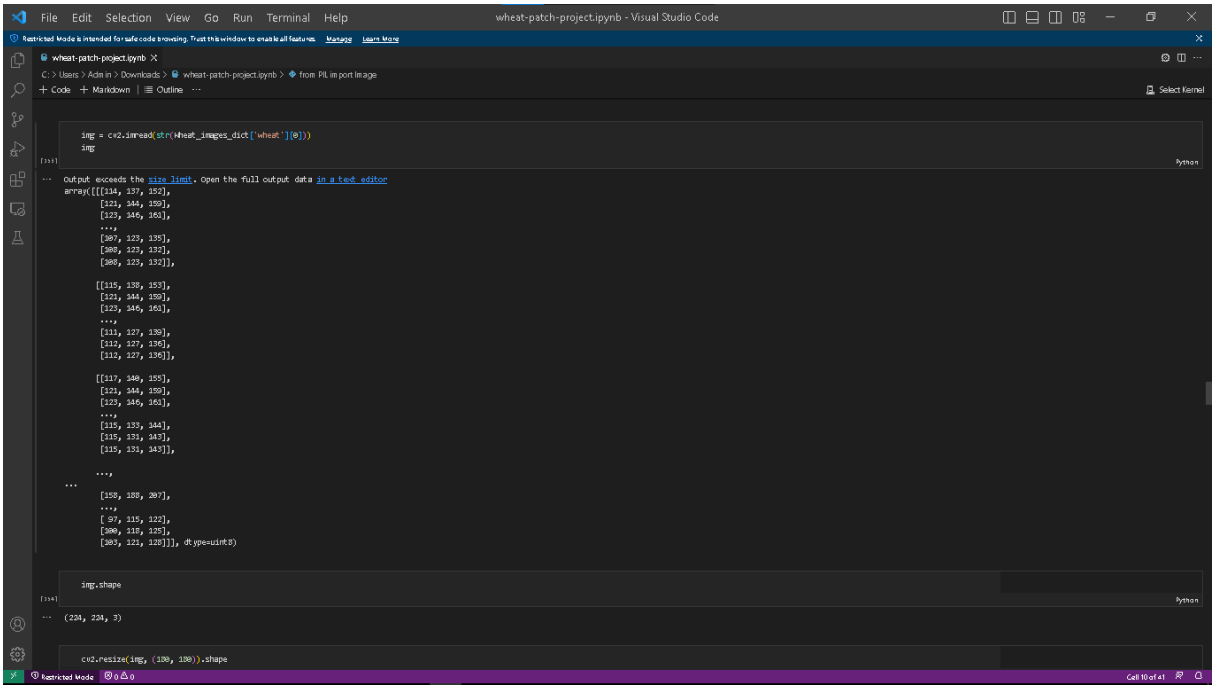


Fig 10.13 Assigning in arrays

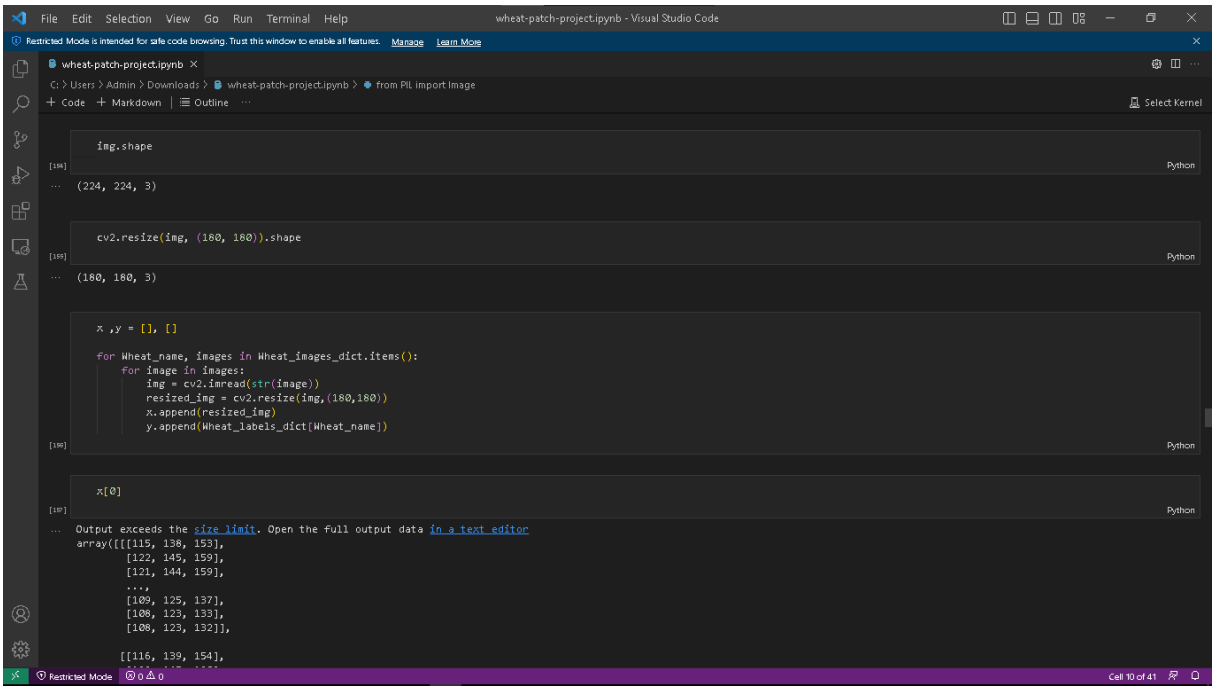


Fig 10.14 check arrays

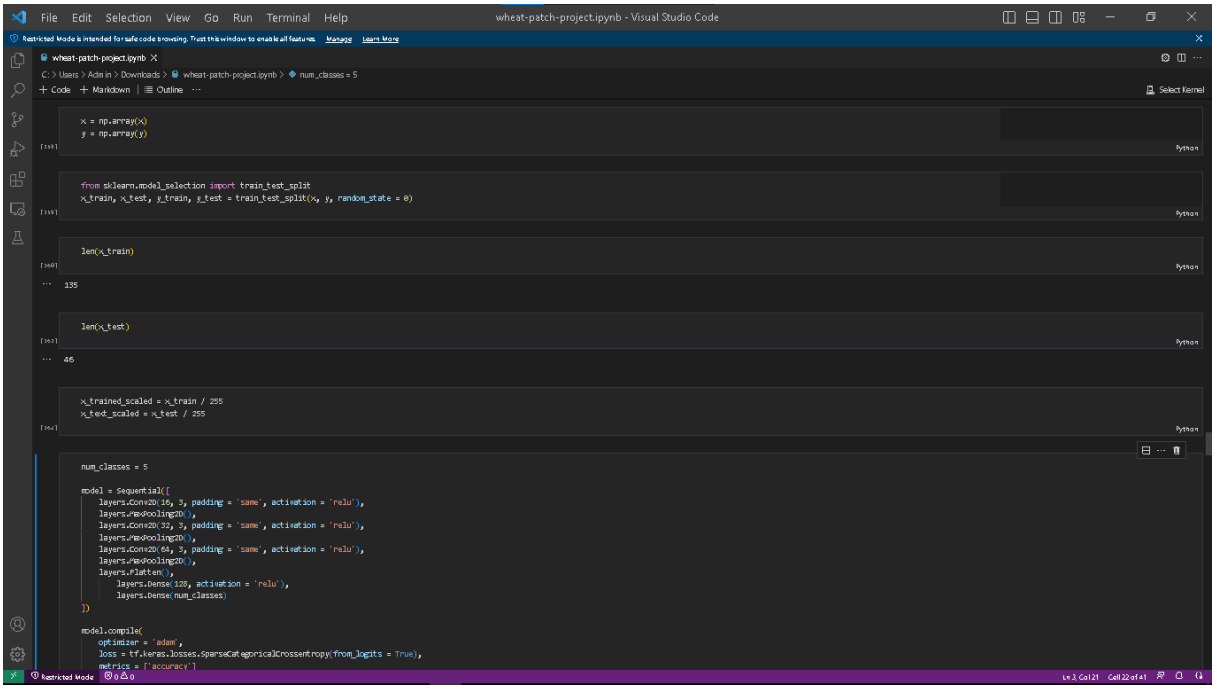


Fig 10.15 Model sequential

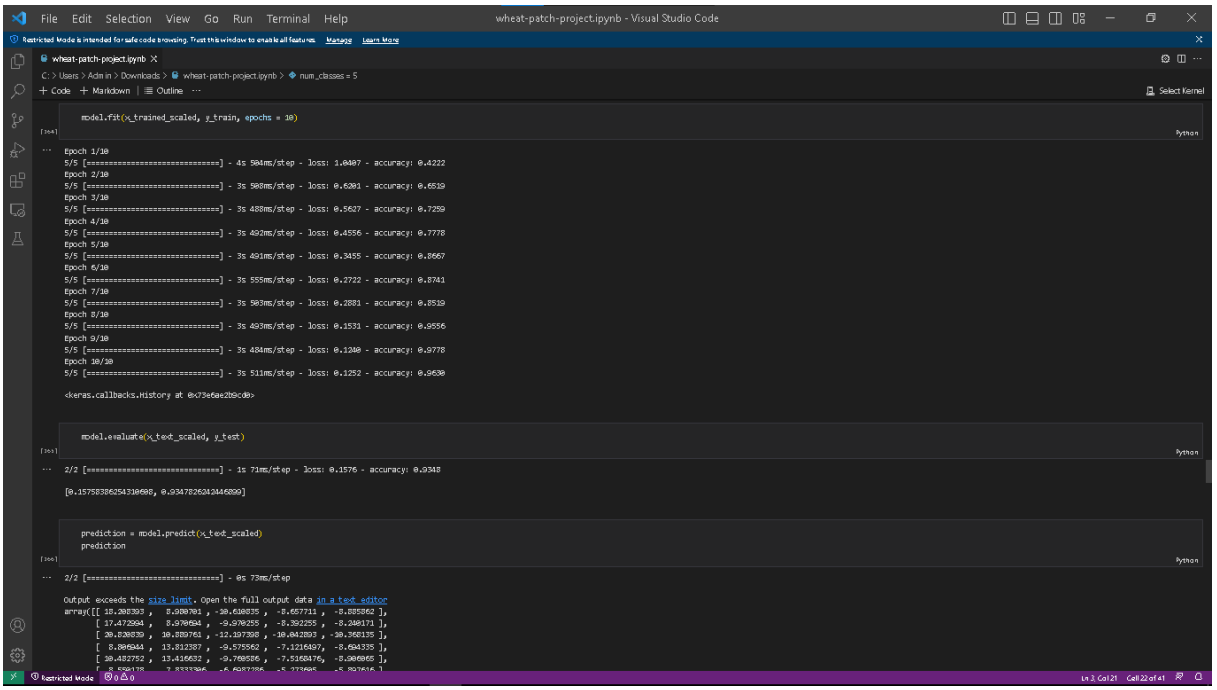


Fig 10.16 Model fitting

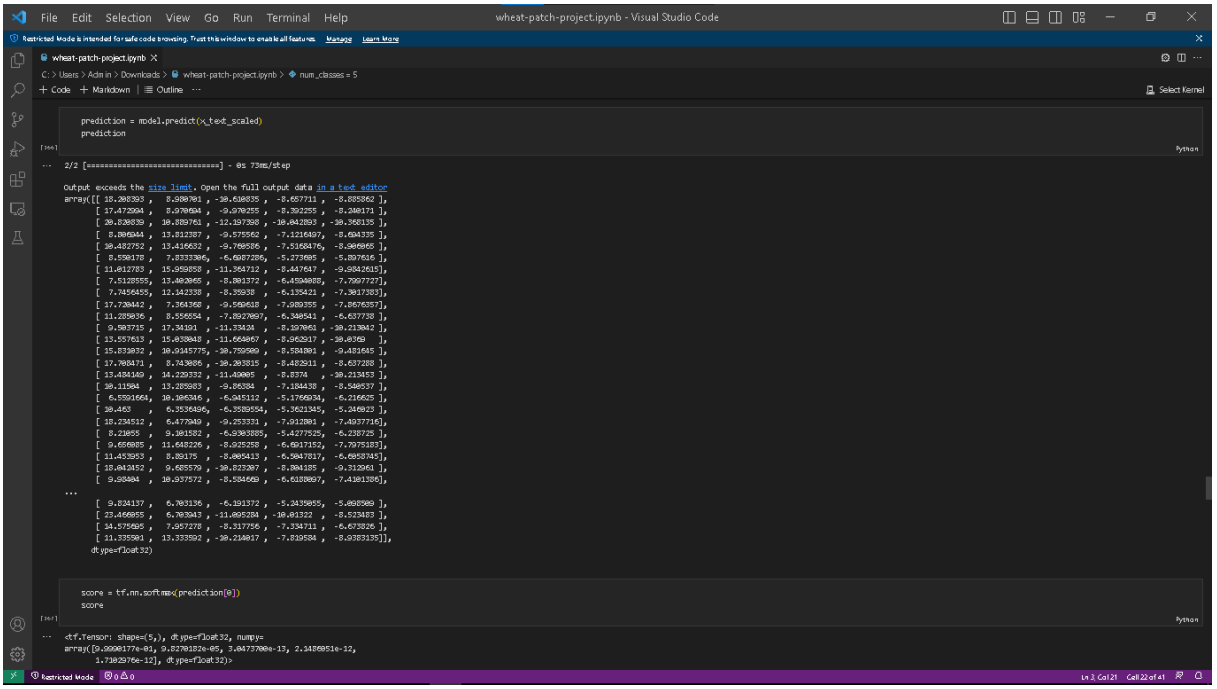


Fig 10.17 Prediction

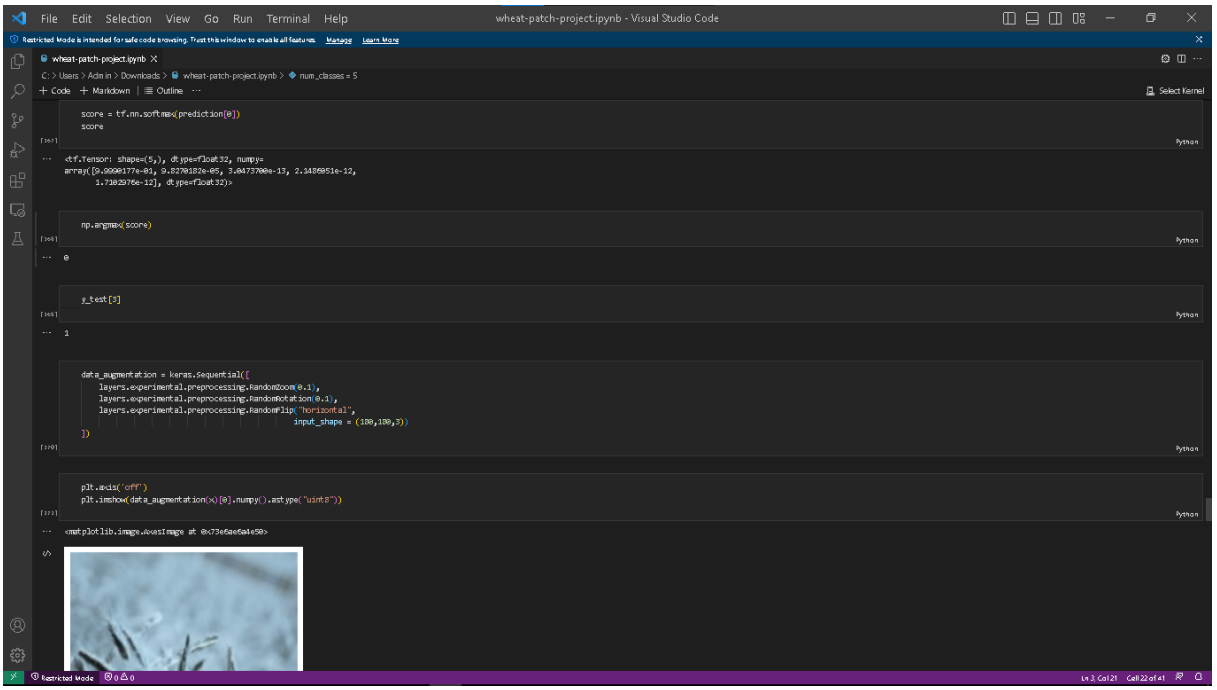


Fig 10.18 Model sequential

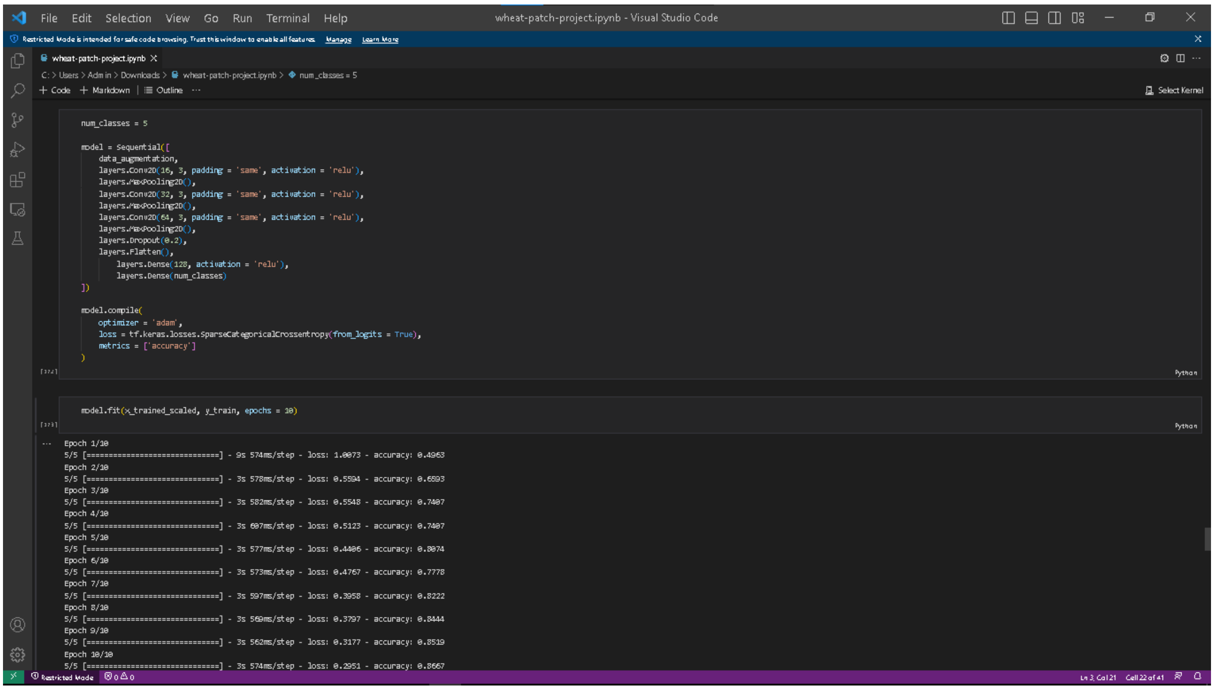


Fig 10.19 Resizing

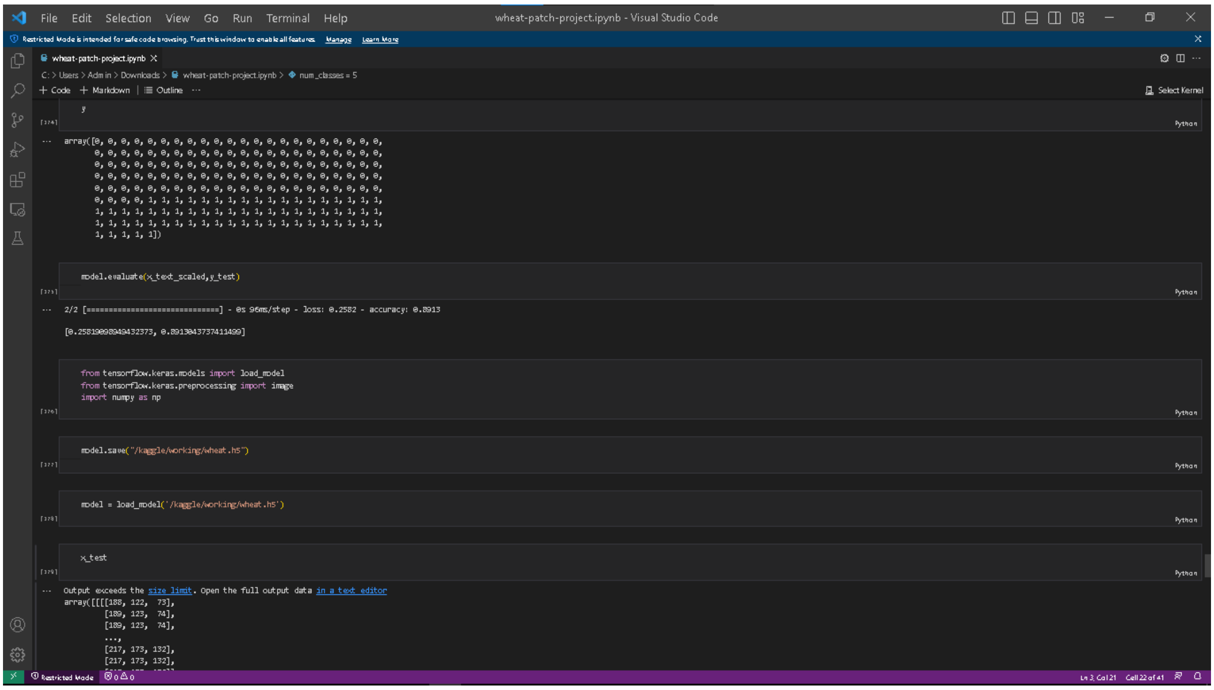


Fig 10.20 Validation

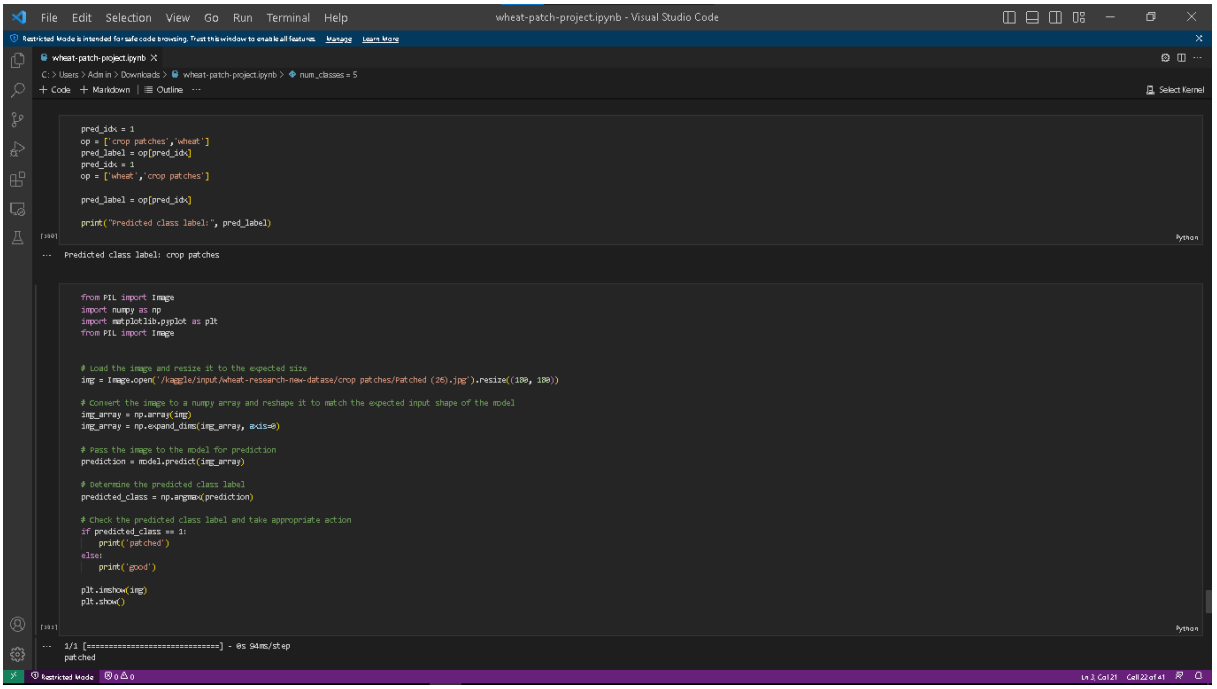


Fig 10.21 Parameter passing

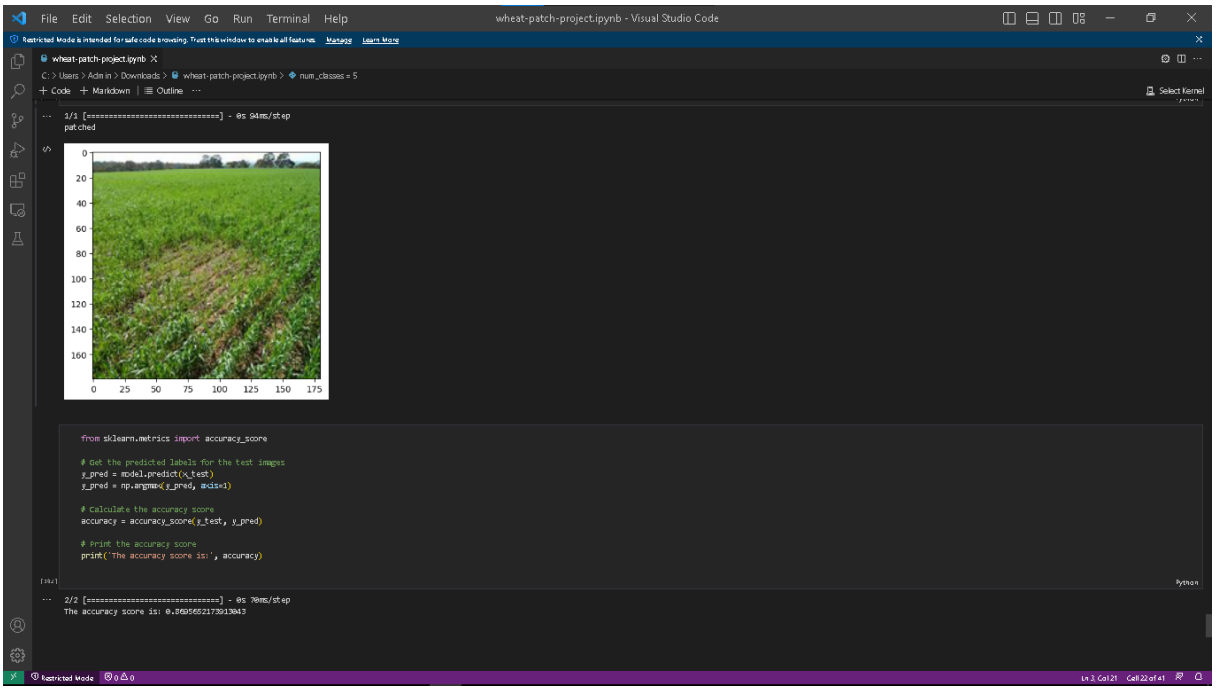


Fig 10.22 Accuracy check

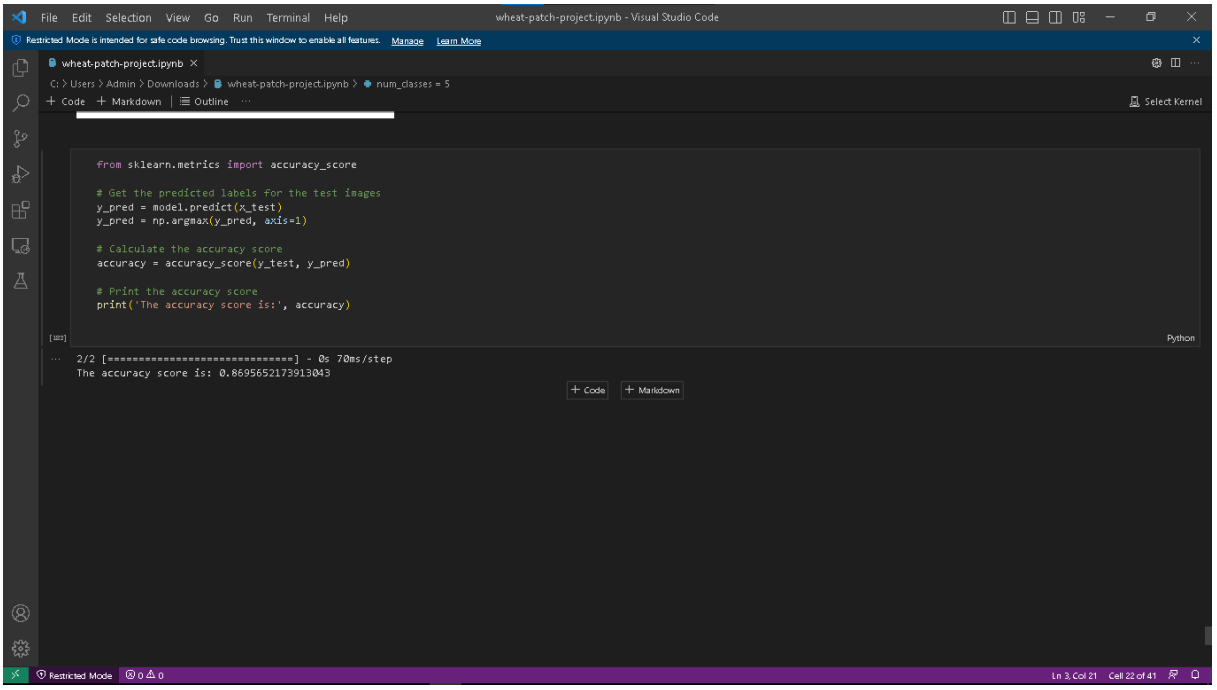


Fig 10.23 Output

CHAPTER 11
CONCLUSION AND FUTURE ENHANCEMENT

11.1 CONCLUSION

Creating a machine learning image prediction model to assist farmers can be a powerful tool in helping them to make better decisions about their crops and improve their yields. By analysing images of crops, the model can detect patches and other issues that may be affecting the crop's health, and provide insights into the best course of action.

The potential benefits of an ML image prediction model for farmers are significant. Such a model can help farmers to make more informed decisions about their crops, leading to higher yields and improved profitability. Additionally,it leads the farmers to environmental benefits.

Developing an ML image prediction model for farmers requires careful consideration of the specific needs and challenges of the agricultural context. With proper attention to these factors, however, such a model can be a valuable tool for improving crop yields and sustainability in agriculture.

11.2 FUTURE ENCHANCEMENT

Precision agriculture: ML algorithms can help farmers to make more informed decisions about crop management, such as identifying the optimal time to plant or harvest crops, predicting crop yields, and optimising irrigation and fertiliser application.

Crop monitoring: ML algorithms can be used to monitor crops for signs of stress, disease, or pest infestation, enabling farmers to take corrective action more quickly and effectively.

Autonomous farming: ML algorithms can be used to develop autonomous farming systems, such as robotic harvesters or drones that can monitor crops and apply treatments.

Climate-smart agriculture: ML algorithms can help farmers to adapt to changing climate conditions, such as predicting weather patterns, identifying drought-resistant crop varieties, and optimising crop management practices to reduce greenhouse gas emissions.

Supply chain optimization: ML algorithms can be used to optimise supply chain management in agriculture, such as predicting market demand for crops, optimising transportation routes, and reducing waste in food production and distribution.

REFERENCES

[1] Umars.khan, Waqars.qureshi, Wahir namas, “Crop and weed classification in aerial imagery sesame crop field using patch based deep learning models”, 2022

[2] Meera gandhi Sasmitha s, Choudhury,“Intellicorp: an ensemble model to predict crop using machine learning algorithms”, 2022

[3] S.Imran moazzam, Umar s.khan, Waqar s, javid iqbal, and Amir hamza“A patch-image based classification approach for detection of weeds in sugar beet crop”, 2021

[4] Swati vashisht, Praveen kumar, Munesh chandra“Improvised extreme learning machine for crop yield prediction”, 2021

[5] S.Imran moazzam, Umar s.khan, Waqar s, javid iqbal, and Amir hamza“A patch-image based classification approach for detection of weeds in sugar beet crop”, 2021

[6] Sindhu madhuri G, Sameer paudel, Prasanth giri, Shanthanu bagathur karki “Prediction of crop yield based on soil moisture using machine learning algorithms”, 2020