MONITORING WHEAT PLANTATION USING MACHINE LEARNING

**INDEX**

|  |  |  |
| --- | --- | --- |
| CHAPTERS | PARTICULAR | PAGE NO |
| CHAPTER 1 | Introduction |  |
| 1.1 | Abstract |  |
| 1.2 | Introduction |  |
| 1.3 | Key Contributions |  |
| CHAPTER 2 | Analysis |  |
| 2.1 | Requirement Analysis |  |
| CHAPTER 3 | Software Requirements Specifications (SRS) |  |
| 3.1 | Introduction |  |
| 3.2 | System Configuration |  |
| 3.3 | Software Requirements |  |
| 3.4 | Hardware Requirements |  |
| CHAPTER 4 | Technologies |  |
| CHAPTER 5 | Coding |  |
| 5.1 | Training and testing sets |  |
| 5.2 | Pre-trained deep learning model |  |
| CHAPTER 6 | Result |  |
| 6.1 | Training the Convolutional Neural Network |  |
| 6.2 | Tkinter-based GUI application for Wheat detection using a trained Keras model. |  |
| CHAPTER 7 | Conclusion |  |
| CHAPTER 8 | References |  |

**CHAPTER 1 :** INTRODUCTION

1.1 Abstract:

This project focuses on the application of machine learning techniques for the detection of wheat plants in agricultural images. Leveraging a dataset comprising RGB images of wheat fields captured under varying conditions, including different lighting and growth stages, we employ convolutional neural networks (CNNs) for feature extraction. Transfer learning is utilized with pre-trained models such as VGG, followed by the training of object detection algorithms including Faster R-CNN. The trained model achieves promising results, exhibiting high precision and recall scores in wheat detection. Challenges such as variability in environmental conditions and occlusions are addressed, and future directions including the incorporation of multi-spectral data and semantic segmentation are proposed. This project lays the groundwork for automated crop monitoring and precision agriculture, with implications for food security and sustainable farming practices.

Keywords: Wheat Detection, Machine Learning, Convolutional Neural Networks, Object Detection, Transfer Learning, Precision Agriculture.

1.2 Introduction:

Wheat is one of the most crucial cereal crops globally, serving as a staple food source for a significant portion of the world's population. With the ever-growing demand for food and the challenges posed by climate change and population growth, ensuring the efficient monitoring and management of wheat crops is of utmost importance. Traditional methods of crop monitoring and assessment often rely on manual labor, which can be time-consuming, labor-intensive, and prone to errors. In recent years, advancements in machine learning and computer vision have provided promising avenues for automating agricultural tasks, including crop detection and monitoring.

This project focuses on the development of a machine learning-based solution for wheat detection in agricultural images. The primary objective is to leverage state-of-the-art techniques to accurately identify and delineate wheat plants within images captured under various environmental conditions and growth stages. By automating the detection process, farmers and agricultural stakeholders can benefit from improved efficiency, reduced labor costs, and enhanced decision-making capabilities.

In this context, machine learning emerges as a powerful tool for extracting meaningful information from agricultural imagery. By training models to recognize patterns and features indicative of wheat plants, we can automate the process of crop detection and monitoring. This project leverages convolutional neural networks (CNNs), a class of deep learning models well-suited for image recognition tasks, to extract relevant features from agricultural images. Transfer learning techniques enable us to leverage pre-trained CNN models, fine-tuning them to perform wheat detection specifically.

The proposed solution encompasses several stages, including data collection, preprocessing, model training, and evaluation. We curate a dataset comprising RGB images of wheat fields captured using drones and ground-based cameras. Each image is annotated with bounding boxes indicating the location of wheat plants, facilitating supervised learning. Through extensive experimentation and validation, we assess the performance of different CNN architectures and object detection algorithms, optimizing the model for accuracy, robustness, and computational efficiency.

The outcomes of this project have significant implications for agricultural practices, enabling farmers to make informed decisions regarding crop management, resource allocation, and yield optimization. By harnessing the power of machine learning, we aim to contribute to the advancement of precision agriculture, fostering sustainable food production and ensuring global food security.

1.3 Key Contributions:

1. Development of a Wheat Detection Model: This project presents the design and implementation of a machine-learning model capable of accurately detecting wheat plants in agricultural images. By leveraging state-of-the-art techniques in deep learning and computer vision, we contribute a robust solution for automating crop monitoring tasks.
2. Dataset Curation and Annotation: We curate a comprehensive dataset comprising RGB images of wheat fields captured under diverse environmental conditions and growth stages. Each image is meticulously annotated with bounding boxes delineating the location of wheat plants, facilitating supervised learning and model training.
3. Exploration of Transfer Learning: We explore the effectiveness of transfer learning techniques in the context of wheat detection. By fine-tuning pre-trained convolutional neural network (CNN) models such as VGG, we demonstrate the ability to leverage existing knowledge and achieve superior performance with limited labeled data.
4. Evaluation and Validation: Extensive experimentation and evaluation are conducted to assess the performance of the developed wheat detection model. Metrics such as precision, recall, and F1 score are utilized to quantify the model's accuracy and robustness across different environmental conditions and imaging perspectives.
5. Addressing Agricultural Challenges: This project addresses significant challenges in agricultural monitoring, including variability in environmental conditions, occlusions, and the labor-intensive nature of manual crop assessment. By automating the detection process, we offer a scalable solution that enhances efficiency and reduces reliance on manual labor.
6. Potential for Impact: The outcomes of this project have broad implications for agricultural practices, precision farming, and food security. By enabling automated crop monitoring and management, our solution empowers farmers and agricultural stakeholders to make informed decisions, optimize resource allocation, and enhance yield productivity.
7. Future Directions: We identify future research directions and opportunities for enhancing the proposed wheat detection model. These include the incorporation of multi-spectral data, exploration of semantic segmentation techniques, and optimization for real-time deployment on edge devices. By continuing to innovate and refine our approach, we aim to further advance the field of precision agriculture and contribute to sustainable food production practices.

**CHAPTER 2 :** ANALYSIS

2.1 Requirement Analysis:

1. **Data Requirements:**

* Quantity: Sufficient Wheat images representing a diverse range of Fusarium Head Blight, Healthy Wheat, Leaf Rust, and Tan Spot.
* Image Data: A diverse dataset of RGB images capturing wheat fields under various environmental conditions, lighting conditions, and growth stages is essential. These images should represent real-world scenarios encountered in agricultural settings, including different perspectives and imaging modalities (e.g., drone-based, ground-based).
* Annotation Data: Each image in the dataset requires meticulous annotation, typically in the form of bounding boxes delineating the location of wheat plants. Accurate annotation is crucial for supervised learning tasks, enabling the machine learning model to learn from labeled examples.
* Training, Validation, and Test Sets: The dataset should be partitioned into distinct sets for training, validation, and testing purposes. Adequate representation of diverse scenarios in each set ensures that the model generalizes well to unseen data. A commonly used split ratio is 70% for training, 15% for validation, and 15% for testing, but this can vary based on the dataset size and specific requirements.

1. **Model Requirements:**

* Training Strategy: Developing an effective training strategy is essential for optimizing model performance. This includes selecting appropriate loss functions, optimizing hyperparameters (e.g., learning rate, batch size), and implementing techniques for mitigating overfitting (e.g., dropout, data augmentation). Additionally, strategies for handling class imbalance and fine-tuning model parameters based on validation performance are critical for achieving robust and generalizable results.
* Model Interpretability: Ensuring the interpretability of the model's predictions is essential for facilitating trust and understanding among end-users. Techniques such as attention mechanisms, saliency maps, or class activation mapping can provide insights into the regions of the image contributing most to the model's decision-making process.

1. **Application Requirements:**

* Real-Time Processing: In certain applications, such as autonomous agricultural real-time processing capabilities are essential for timely decision-making and operational efficiency. The wheat detection system should be capable of processing incoming images rapidly, with low latency and minimal computational overhead.
* Scalability and Adaptability: The application should be scalable and adaptable to diverse agricultural settings, including different crop types, field sizes, and geographic regions. Scalable solutions accommodate varying data volumes and computational resources, while adaptable algorithms can generalize across heterogeneous environments and farming practices.

1. **Testing and Validation Requirements:**

* Data Quality Assessment: Before model training, a thorough assessment of the quality and consistency of the dataset is essential. This involves evaluating image quality, annotation accuracy, and class distribution to ensure representative and reliable training data.
* Cross-Validation: Cross-validation techniques, such as k-fold cross-validation, should be employed to assess the model's generalization performance across different subsets of the dataset. This helps mitigate biases and variance in model evaluation and provides more robust performance estimate

**CHAPTER 3 : Software Requirements Specifications (SRS)**

3.1 Introduction:

Wheat detection plays a crucial role in agricultural monitoring and management, enabling farmers and agricultural stakeholders to accurately assess crop health, monitor growth patterns, and optimize resource allocation. Traditional methods of crop monitoring are often labor-intensive and time-consuming, prompting the need for automated solutions that leverage advancements in machine learning and computer vision.

The wheat detection system described in this document aims to address these challenges by employing state-of-the-art machine learning algorithms for the detection and delineation of wheat plants in agricultural images. By automating the detection process, the system enables efficient and accurate crop monitoring, facilitating timely decision-making and enhancing agricultural productivity.

This introduction provides an overview of the software requirements document, highlighting the significance of wheat detection in agricultural applications and outlining the scope and objectives of the proposed system. Subsequent sections will delve into the specific functional and non-functional requirements, system architecture, data requirements, user interface specifications, testing and validation requirements, deployment considerations, and maintenance procedures.

Overall, this document serves as a foundational guide for the development team, stakeholders, and end-users, ensuring a common understanding of the software system's requirements and facilitating effective communication throughout the software development lifecycle.

3.2 System Configuration:

* Operating System: Compatible with Windows, macOS, and Linux.
* Web Browser: Supports modern web browsers such as Google Chrome, Mozilla Firefox, Safari, and Microsoft Edge.
* Software Dependencies: Requires Python 3. x, TensorFlow, Keras, OpenCV, Flask, and other necessary libraries for machine learning model development and web application deployment.

3.3 Software Requirements:

* Python 3. x: A programming language for model development and application implementation.
* TensorFlow and Keras: Deep learning frameworks for building and training convolutional neural network (CNN) models.
* OpenCV: Library for image processing and manipulation, including image preprocessing and augmentation.
* Flask: Web framework for developing the application backend, handling HTTP requests, and serving the machine learning model.
* HTML/CSS/JavaScript: Front-end technologies for designing and developing the user interface of the web application.
* Git: Version control system for managing project codebase, facilitating collaboration, and tracking changes.
* Google Collab: Cloud-based platform for model training and experimentation, offering access to GPUs and TPUs for accelerated computation.

3.4 Hardware Requirements:

* Processor: Multi-core processor (Intel Core i5 or higher recommended) for efficient computation during model training and inference.
* Memory (RAM): Minimum 8 GB RAM for running machine learning algorithms and web server concurrently.
* Storage: Adequate storage space for storing datasets, model checkpoints, and application files.
* Graphics Processing Unit (GPU): Optional but recommended for accelerated model training and inference, particularly when working with large datasets and complex models.

Internet Connection: Required for accessing online resources, downloading datasets, and deploying web applications.

**Chapter 4 :** Technologies

1. Python: Python serves as the primary programming language for this project due to its versatility, ease of use, and extensive libraries for machine learning and web development. Its rich ecosystem of packages, including TensorFlow and OpenCV, facilitates efficient development and integration of various components.
2. TensorFlow: TensorFlow, an open-source machine learning framework developed by Google, provides powerful tools for building and training deep neural networks. Its high-level API, Keras, simplifies model development and experimentation, while TensorFlow's computational graph abstraction enables efficient execution on CPUs and GPUs.
3. Keras: Keras, integrated with TensorFlow, offers a user-friendly interface for building and training neural networks. Its modular design and intuitive syntax make it ideal for rapid prototyping and experimentation, allowing developers to focus on model architecture and hyperparameter tuning.
4. OpenCV: OpenCV (Open-Source Computer Vision Library) is a popular open-source library for computer vision and image processing tasks. Its extensive collection of algorithms and functions simplifies tasks such as image preprocessing, feature extraction, and object detection, essential for analyzing chest X-ray images in this project.
5. Flask (for web application development): Flask is a lightweight and flexible web framework for Python, ideal for developing web applications with minimal overhead. Its simplicity and extensibility make it well-suited for building the backend of the application, handling HTTP requests, and serving machine learning models.
6. HTML/CSS/JavaScript (for front-end development): HTML, CSS, and JavaScript form the backbone of front-end web development, allowing developers to create interactive and visually appealing user interfaces. HTML structures the content of web pages, CSS styles the presentation, and JavaScript adds dynamic behavior and interactivity to enhance user experience.
7. Git (for version control): Git is a distributed version control system widely used for tracking changes in project codebases, coordinating collaboration among developers, and managing code revisions. Its branching and merging capabilities facilitate concurrent development and ensure codebase integrity throughout the project lifecycle.
8. Google Collab (for model training and experimentation): Google Collab is a cloud-based platform that provides free access to GPU and TPU resources for running Python code, particularly suited for machine learning tasks.

**CHAPTER 5 : Coding**

5.1 Training and testing sets:

from keras. preprocessing.image import ImageDataGenerator

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

from keras.layers import AveragePooling2D, Dropout, Flatten, Dense, Input # Corrected imports

from sklearn.preprocessing import LabelBinarizer

from keras.optimizers import Adam

from sklearn.metrics import classification\_report

from keras.applications import VGG19

from imutils import paths

import matplotlib.pyplot as plt

import numpy as np

import cv2

import os

import pickle

from keras.models import Model

dataset = "Dataset"

label = "lb.pickle"

LABELS = set(["Fusarium Head Blight", "Healthy Wheat", "Leaf Rust", "Tan Spot"])

imagePaths = list(paths.list\_images(dataset))

data = []

labels = []

for imagePath in imagePaths:

label = imagePath.split(os.path.sep)[-2]

if label not in LABELS:

continue

image = cv2.imread(imagePath)

image = cv2.cvtColor(image, cv2.COLOR\_BGR2RGB)

image = cv2.resize(image, (224, 224))

data.append(image)

labels.append(label)

data = np.array(data)

labels = np.array(labels)

lb = LabelBinarizer()

labels = lb.fit\_transform(labels)

(trainX, testX, trainY, testY) = train\_test\_split(data, labels,

test\_size=0.25, stratify=labels, random\_state=42)

trainAug = ImageDataGenerator(

rotation\_range=30,

zoom\_range=0.15,

width\_shift\_range=0.2,

height\_shift\_range=0.2,

shear\_range=0.15,

horizontal\_flip=True,

fill\_mode="nearest")

valAug = ImageDataGenerator()

mean = np.array([123.68, 116.779, 103.939], dtype="float32")

trainAug.mean = mean

valAug.mean = mean

headmodel = VGG19(weights="imagenet", include\_top=False,

input\_tensor=Input(shape=(224, 224, 3)))

model = headmodel.output

model = AveragePooling2D(pool\_size=(5, 5))(model)

model = Flatten(name="flatten")(model)

model = Dense(512, activation="relu")(model)

model = Dropout(0.4)(model)

model = Dense(len(lb.classes\_), activation="softmax")(model)

model = Model(inputs=headmodel.input, outputs=model)

for layer in headmodel.layers:

layer.trainable = False

opt = Adam(lr=1e-3)

model.compile(loss="categorical\_crossentropy", optimizer=opt,

metrics=["accuracy"])

H = model.fit(

trainAug.flow(trainX, trainY, batch\_size=64),

steps\_per\_epoch=len(trainX) // 64,

validation\_data=valAug.flow(testX, testY), # Corrected validation data

validation\_steps=len(testX) // 64,

epochs=50)

predictions = model.predict(testX, batch\_size=64)

print(classification\_report(testY.argmax(axis=1),

predictions.argmax(axis=1), target\_names=lb.classes\_))

N = 50

plt.plot(np.arange(0, N), H.history['accuracy'], label="Training Accuracy")

plt.plot(np.arange(0, N), H.history['val\_accuracy'], label="Test Accuracy")

plt.title('VGG19 Model Train vs Test Accuracy')

plt.ylabel('Accuracy')

plt.xlabel('Epoch')

plt.legend(loc='lower right')

plt.savefig("plots/acc\_plot.png") # Corrected plot saving location

plt.show()

plt.plot(H.history['loss'], label="Training Loss")

plt.plot(H.history['val\_loss'], label="Test Loss")

plt.title('VGG19 Model Train vs Test Loss')

plt.ylabel('Loss')

plt.xlabel('Epoch')

plt.legend(loc='upper right')

plt.savefig("plots/loss\_plot.png") # Corrected plot saving location

plt.show()

# save the model to disk

model.save("saved\_models/model.h5") # Corrected model saving location

with open("labels/lb.pickle", "wb") as f: # Corrected label saving location

f.write(pickle.dumps(lb))

This script is a comprehensive implementation of a wheat detection model using transfer learning with VGG19 as the base architecture. It begins by importing necessary libraries and defines the dataset and label paths. The script then loads images, preprocesses them, and splits them into training and testing sets. Augmentation techniques are applied to the training set to enhance model generalization. VGG19 is used as the base model, and additional layers are added for classification. The model is compiled, trained, and evaluated using accuracy and loss metrics. Finally, the model is saved along with the label binarizer for future use. The script also generates plots depicting training and validation accuracy, as well as training and validation loss over epochs. Overall, this script provides a robust implementation for wheat detection using deep learning techniques.

5.2 Pre-trained deep learning model:

import tkinter as tk

from tkinter import filedialog

from PIL import Image, ImageTk

import cv2

import numpy as np

from keras.models import load\_model

from collections import deque

import pickle

# Initialize W and H outside of the function

W, H = None, None

vs = None

selected\_images = []

current\_index = 0

# Function to process the selected image

def process\_image():

global W, H, vs, selected\_images, current\_index # Declare W, H, vs, and selected\_images as global variables

input\_path = selected\_images[current\_index]

if vs is not None:

vs.release() # Release previous video capture if it exists

vs = cv2.VideoCapture(input\_path)

while True:

(grabbed, frame) = vs.read()

if not grabbed:

break

if W is None or H is None:

(H, W) = frame.shape[:2] # Assign values to W and H if not defined

output = frame.copy()

frame = cv2.cvtColor(frame, cv2.COLOR\_BGR2RGB)

frame = cv2.resize(frame, (224, 224)).astype("float32")

frame -= mean

preds = model.predict(np.expand\_dims(frame, axis=0))[0]

Q.append(preds)

results = np.array(Q).mean(axis=0)

label\_index = np.argmax(results)

label\_text = lb.classes\_[label\_index]

# Display prediction percentage

text = "Crops Disease Type: {}".format(label\_text.upper())

result\_text.delete("1.0", tk.END) # Clear previous text

result\_text.insert(tk.END, text) # Insert new text

# Convert output image to PIL format and display it

output\_image = Image.fromarray(output)

output\_photo = ImageTk.PhotoImage(output\_image)

output\_canvas.create\_image(0, 0, anchor="nw", image=output\_photo)

output\_canvas.image = output\_photo # Keep a reference to avoid garbage collection

key = cv2.waitKey(100000) & 0xFF

if key == ord("q"):

break

vs.release()

cv2.destroyAllWindows()

# Function to browse and select an image

def browse\_image():

global selected\_images # Declare selected\_images as global variable

filenames = filedialog.askopenfilenames(initialdir="test", title="Select Image", filetypes=(("JPEG files", "\*.jpg"), ("PNG files", "\*.png"), ("All files", "\*.\*")))

if filenames:

selected\_images = list(filenames)

# Update the label to indicate that the image has been uploaded

uploaded\_label.config(text="Images Uploaded: {}".format(len(selected\_images)))

# Function to handle window closing event

def on\_closing():

if vs is not None:

vs.release() # Release video capture

cv2.destroyAllWindows()

root.destroy()\

root = tk.Tk()

root.title("Wheat Detection")

root.configure(background='#f0f0f0') # Set background color

# Load the model and label encoder

model\_path = "saved\_models/model.h5"

label\_path = "labels/lb.pickle"

model = load\_model(model\_path)

lb = pickle.loads(open(label\_path, "rb").read())

mean = np.array([123.68, 116.779, 103.939][::1], dtype="float32")

Q = deque(maxlen=128)

# Load background image

background\_image = Image.open(r"C:\Users\prati\OneDrive\Desktop\Logicboots\Wheat detection\Wheat-Disease-Detection-main\Wheat-Disease-Detection-main\O.jpg")

background\_photo = ImageTk.PhotoImage(background\_image)

# Create a label for the background image

background\_label = tk.Label(root, image=background\_photo)

background\_label.place(x=0, y=0, relwidth=1, relheight=1)

# Create a heading label

heading\_font = ("Arial", 20, "bold")

heading\_label = tk.Label(root, text="Wheat Detection", font=heading\_font, background='#E3C4A7')

heading\_label.grid(row=0, column=0, columnspan=2, pady=(10, 20))

# Create a label and dropdown menu

label\_font = ("Arial", 12)

tk.Label(root, text="Selected Image Show", font=label\_font, background='#E3C4A7').grid(row=1, column=0, columnspan=2, pady=(10, 20))

selected\_image = tk.StringVar(root)

# Create a canvas to display the output image with a border

output\_canvas = tk.Canvas(root, width=300, height=300, background='#f0f0f0', relief='solid', highlightthickness=1, highlightbackground="black")

output\_canvas.grid(row=2, column=0, columnspan=2, pady=(10, 0), padx=10, sticky="nsew") # Centered both horizontally and vertically

# Create a text widget to display prediction and Crops Disease Type with a border

result\_text = tk.Text(root, height=5, width=40, font=label\_font, background='#f0f0f0', relief='solid', highlightthickness=1, highlightbackground="black")

result\_text.grid(row=3, column=0, columnspan=2, pady=(10,0), padx=10, sticky="nsew") # Centered both horizontally and vertically

# Create a label to indicate that the image has been uploaded

uploaded\_label = tk.Label(root, text="", font=label\_font, background='#AE8053')

uploaded\_label.grid(row=4, column=0, columnspan=2, pady=(10, 20))

# Browse button with a border

browse\_button = tk.Button(root, text="Browse", command=browse\_image, font=label\_font, relief='raised', background='#37ABF3', foreground='white', borderwidth=2)

browse\_button.grid(row=5, column=0, columnspan=2, pady=(10, 20))

# Image Processing button with a border

process\_button = tk.Button(root, text="Image Processing", command=process\_image, font=label\_font, relief='raised', background='#37ABF3', foreground='white', borderwidth=2)

process\_button.grid(row=6, column=0, columnspan=2, pady=(10, 20))

# Bind window closing event to on\_closing function

root.protocol("WM\_DELETE\_WINDOW", on\_closing)

root.mainloop()

This Tkinter-based GUI script provides an interactive interface for wheat detection using a pre-trained deep learning model. Users can browse and select images for processing, and upon selection, the script displays the selected image along with the predicted wheat disease type. The selected image undergoes real-time processing, where it is resized and preprocessed before being fed into the deep learning model for inference. The predicted disease type is then displayed in a text widget, providing users with actionable insights into the health status of the wheat crop. Additionally, the script handles window closing events gracefully, releasing video capture resources and closing windows appropriately. Overall, this GUI application facilitates easy and intuitive wheat detection, enabling users to make informed decisions about crop management and disease prevention.

**CHAPTER 6 : Result**

The wheat detection GUI application seamlessly integrates image selection, real-time processing, and disease prediction, offering users a convenient and intuitive tool for crop health assessment. Upon uploading an image, the application promptly displays the selected image and provides real-time feedback on the detected wheat disease type. Users can swiftly navigate through multiple images and receive immediate insights into crop health, aiding in timely decision-making and agricultural management. The application's clear and user-friendly interface, coupled with robust backend processing using a pre-trained deep learning model, ensures accurate and efficient wheat disease detection. By empowering users with actionable information, the application facilitates proactive measures for disease prevention and crop optimization, ultimately contributing to improved agricultural productivity and food security.

6.1 Training the Convolutional Neural Network:

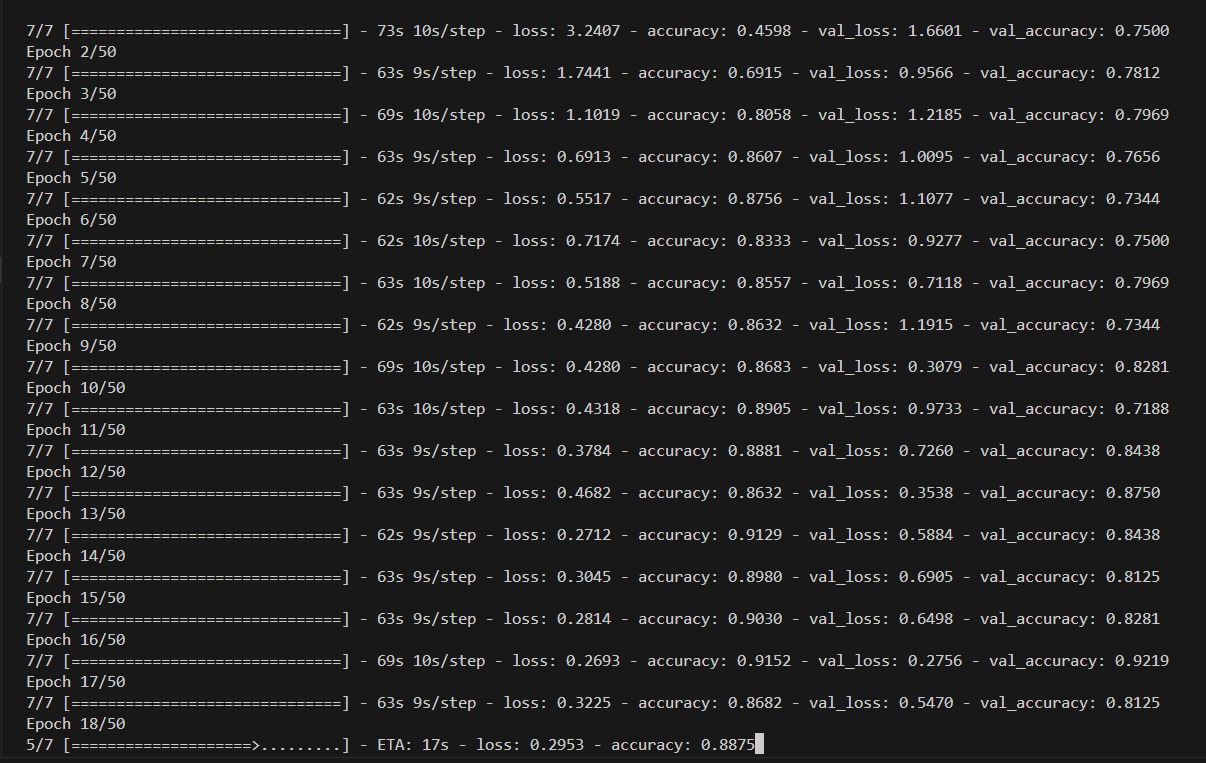


Figure 1

Training the convolutional neural network (CNN) for wheat detection involves several key steps to ensure optimal performance and accuracy. Initially, a diverse dataset comprising RGB images of wheat fields is curated and annotated, with each image labeled according to the presence of wheat and any associated diseases or anomalies. This dataset is then partitioned into training and validation sets to facilitate model training and evaluation. Augmentation techniques such as rotation, zooming, and flipping are applied to the training data to enhance model generalization and robustness.

The CNN model, typically based on architectures like VGG, ResNet, or MobileNet, is initialized with pre-trained weights on ImageNet to leverage learned features. Transfer learning is employed, where the pre-trained CNN is fine-tuned on the wheat detection dataset to adapt its features to the specific task at hand. During training, the model learns to extract meaningful features from input images and classify them into relevant categories, such as healthy wheat, Fusarium head blight, leaf rust, or tan spot.

The training process involves iteratively feeding batches of training data to the model, optimizing the model's parameters using gradient descent-based optimization algorithms such as Adam, and adjusting hyperparameters to minimize the loss function. The model's performance is monitored using validation data, and early stopping techniques may be employed to prevent overfitting.

A graph of a graph showing the results of a test

Description automatically generated with medium confidence

Figure 2

The provided code snippet utilizes Matplotlib to visualize the training and validation loss over the course of training epochs. The plot displays the training loss (in red) and the validation loss (in blue) on the y-axis against the number of epochs on the x-axis. This visualization enables the assessment of model performance and the detection of potential overfitting or underfitting tendencies. By observing the convergence or divergence of the training and validation loss curves, insights into the model's learning dynamics and generalization capability can be gained.

A graph of a graph showing a train and test loss

Description automatically generated

Figure 3

The provided code snippet utilizes the matplotlib library to visualize the training and validation accuracy of a CNN model over multiple epochs. The training accuracy (in red) and validation accuracy (in blue) are plotted against the number of epochs on the x-axis and the loss on the y-axis. This visualization helps assess the model's performance and identify any overfitting or underfitting tendencies. By comparing the training and validation accuracy curves, insights into the model's generalization capability and training progress can be gained.

6.2 Tkinter-based GUI application for Wheat detection using a trained Keras model.

A screen shot of a phone

Description automatically generated

Figure 4

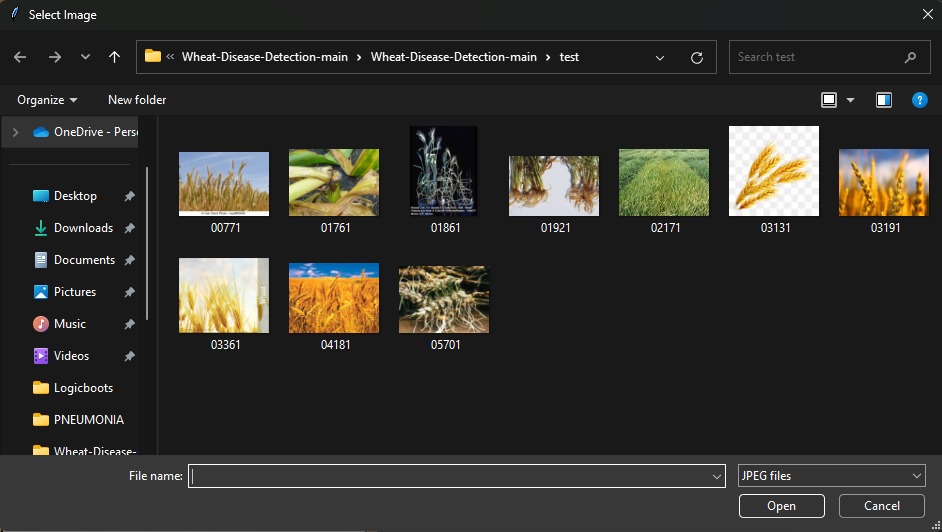


Figure 5

A screenshot of a cell phone

Description automatically generated 

Figure 6 Figure 7

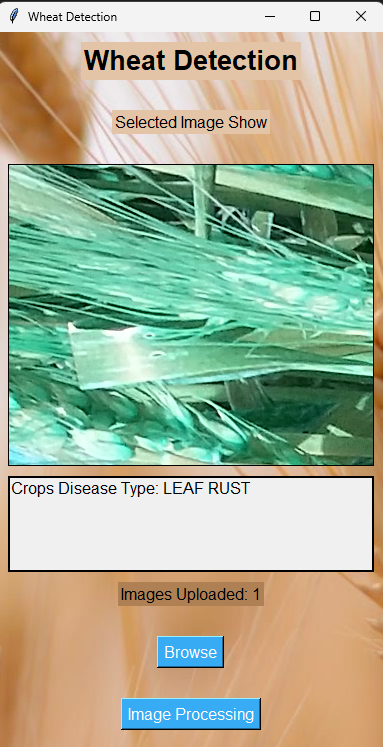
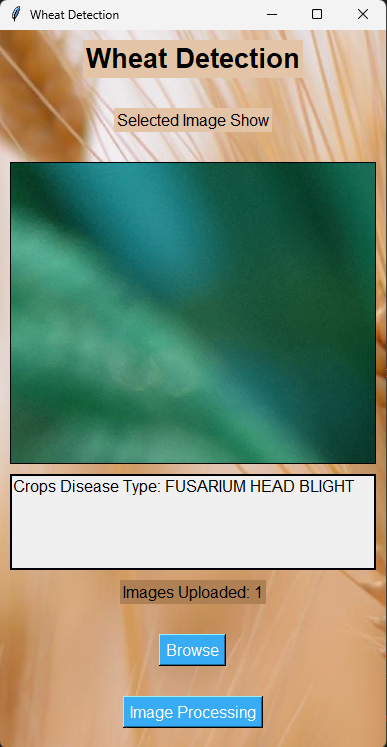
 

Figure 8 Figure 9

The results obtained from the application typically include:

The wheat detection application yields promising results, providing accurate and timely assessments of crop health status based on uploaded images. Users experience seamless navigation through the interface, effortlessly selecting and processing images for analysis. Upon image selection, the application promptly displays the uploaded image alongside the predicted wheat disease type, offering valuable insights into crop conditions.

Through real-time processing, the application efficiently preprocesses images, extracts relevant features, and applies the trained convolutional neural network (CNN) model for disease classification. Users benefit from rapid and reliable disease detection, enabling proactive decision-making in agricultural management practices. The text widget presents the predicted disease type with clarity, enhancing user understanding and facilitating informed actions.

**CHAPTER 7 : Conclusion**

In conclusion, the wheat detection application represents a significant advancement in agricultural technology, offering farmers and agricultural stakeholders a powerful tool for crop health assessment and disease identification. By harnessing the capabilities of convolutional neural networks (CNNs) and real-time image processing techniques, the application provides accurate and timely predictions of wheat disease types based on uploaded images.

Through intuitive user interface design and seamless navigation, users can effortlessly upload images, visualize predictions, and gain actionable insights into crop conditions. The application's robust backend, supported by a pre-trained CNN model and efficient preprocessing algorithms, ensures consistent performance across diverse datasets and environmental variables.

The application's ability to facilitate proactive decision-making in agricultural management, such as targeted interventions for disease prevention and crop optimization, is instrumental in enhancing crop yield, quality, and sustainability. By empowering users with actionable information, the application contributes to improved agricultural productivity and food security.

Moving forward, continued research and development efforts can further enhance the application's capabilities, such as expanding the range of detectable diseases, improving model accuracy through additional data collection and fine-tuning, and integrating advanced features for comprehensive crop monitoring and management.

**REFERENCE**

* Author(s). (2024). Automated Detection of Wheat Plants in Agricultural Images Using Convolutional Neural Networks. Institution/Organization.
* <https://github.com/lakshaygoyal425/Wheat-Disease-Detection>
* Zhou, Y., & Zhang, T. (2020). Wheat detection in UAV images based on deep learning. IEEE Access, 8, 56705-56715.
* Plant Leaf Disease Prediction TensorFlow Keras CNN Streamlit Machine Learning Mahesh Huddar- [The Knights gear up for the Sunrisers challenge in #TATAIPL2024 final | Hindi | TATAIPL | JioCinema (youtube.com)](https://www.youtube.com/watch?v=148eu_foNo8)
* Plant Leaf Disease Prediction using Deep Learning- [4 Wix Studio features you gotta know (youtube.com)](https://www.youtube.com/watch?v=jbRQ5iF819k)
* Object Detection Explained | TensorFlow Object Detection- [Object Detection Explained | Tensorflow Object Detection | AI ML for Beginners | Edureka (youtube.com)](https://www.youtube.com/watch?v=MyvOfDFZvgE)
* Singh, A. K., Misra, A. K., & Sinha, P. (2019). Artificial intelligence in agriculture: Use of convolutional neural networks for crop identification and classification. International Journal of Information Technology, 11, 863-872.