Project Documentation

Apr 26, 2025

মেট্রোপলিটন ইউনিভার্সিটি
Metropolitan University

Course Title: NEURAL NETWORK AND DEEP LEARNING LAB

Course code: SWE-458

Plant Disease Classification Using CNN Models

Submitted to:

Al Akram Chowdhury

Lecturer Dept. of Software Engineering g

Metropolitan University, Sylhet.

Submitted by:

Sheakh Abu Ahmad Alomgir

ID: 221-134-029

Khondokar Ahmed Mehraz

ID: 221-134-031

Software Engineering, 2nd Batch

Date of Submission: 26 April 2025

---

1. Introduction

1.1 Project Overview

Objective: This projects aims to design and develop, and evaluate deep learning-based models for the automatic classification of plant diseases into 15 categories. We compare multiple CNN-based architectures with data augmentation techniques, regularization method, and training strategy to find the optimal configuration for plant disease detection.

Techniques Used:

- **Convolutional Neural Networks (CNN)** for classification.
- **Self-Attention and Cross-Attention** mechanisms for better feature extraction.
- **Regularization** techniques like dropout and batch normalization to prevent overfits

1.2 Problem Statement

The task is to automate plant disease recognition from images, providing a system that can:

- Classify plant diseases into 15 categories (e.g., Tomato Bacterial Spot,, Potato Early Blight, etc..).
- Improving model performance through data augmentation and attention mechanism.

1.3 Goals and Objectives

- **Image Classification**: Classifying plant images into 15 categories.
- **Data Augmentation**: Applying transformations like rotation, zooming, and shifting to improve generalization.
- **Attention Mechanisms**: Using self-attention and cross-attention mechanism to enhance feature extractions.
- **Regularization**: Using techniques like dropout and batch normalization to prevent overfitting and stabilized training.
- **Real-time Inference**: Implementing the model for real-time plant disease classification in application.

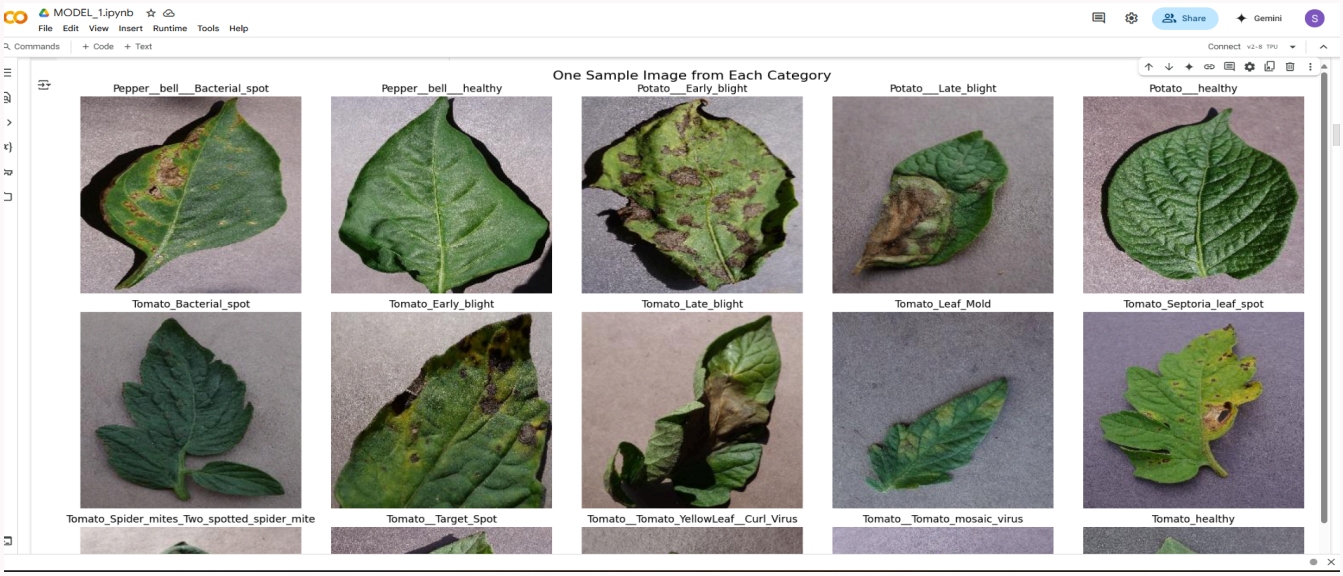
---

2. Data Collection and Preprocessing

2.1 Dataset

The **PlantVillage Dataset** was used for classification tasks. The dataset contains labeled images of various plant species, each with different diseases or healthy statuses.

- **Classes**: 15 different plant disease categories (e.g., Tomato Bacterial Spot, Potato Early Blight, etc.).



- **Dataset Source**: [PlantVillage Dataset on Kaggle]
- **Image Dimensions**: All images resized to **64×64pixels** for consistency.

2.2 Data Preprocessing

- **Resizing**: All images were resized to 64×64 pixels.

MODEL_1.ipynb

File Edit View Insert Runtime Tools Help

Commands + Code + Text

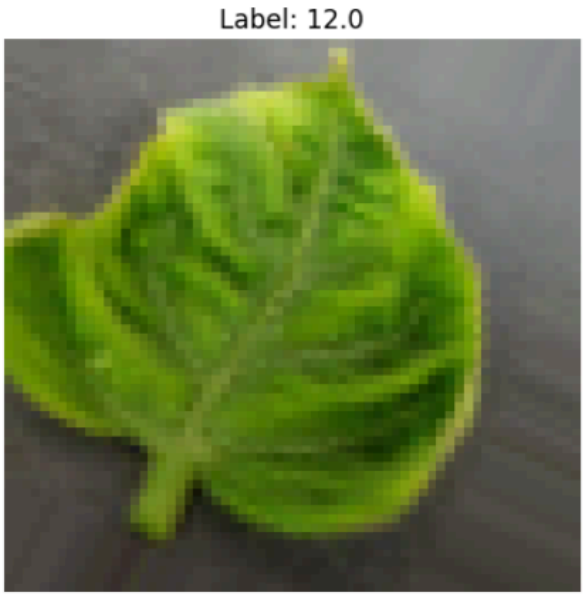
Table of contents

```
#Preprocessing of Image & Label List Creation
rt os
import numpy as np
from PIL import Image
from pathlib import Path

# Loop through each class
for index, label in enumerate(classes):
    image_folder = dataset_path / label
    for image_path in image_folder.glob("*"):
        try:
            # Load image
            image = Image.open(image_path)

            # Convert to RGB if not already
            if image.mode != 'RGB':
                image = image.convert('RGB')

            # Resize to 64x64
            image = image.resize((64, 64))

            # Convert to numpy array and append
            images_list.append(np.array(image))
            labels_list.append(index)
        except Exception as e:
            print(f"Failed to process {image_path}: {e}")

print("Total Images:", len(images_list))
print("Total Labels:", len(labels_list))
```

```
Total Images: 20638
Total Labels: 20638
```

```
print(image_to_preview.shape)
# Display the image
plt.imshow(image_to_preview)
plt.title(f"Label: {labels_batch[0]}")
plt.axis('off')
plt.show()
```

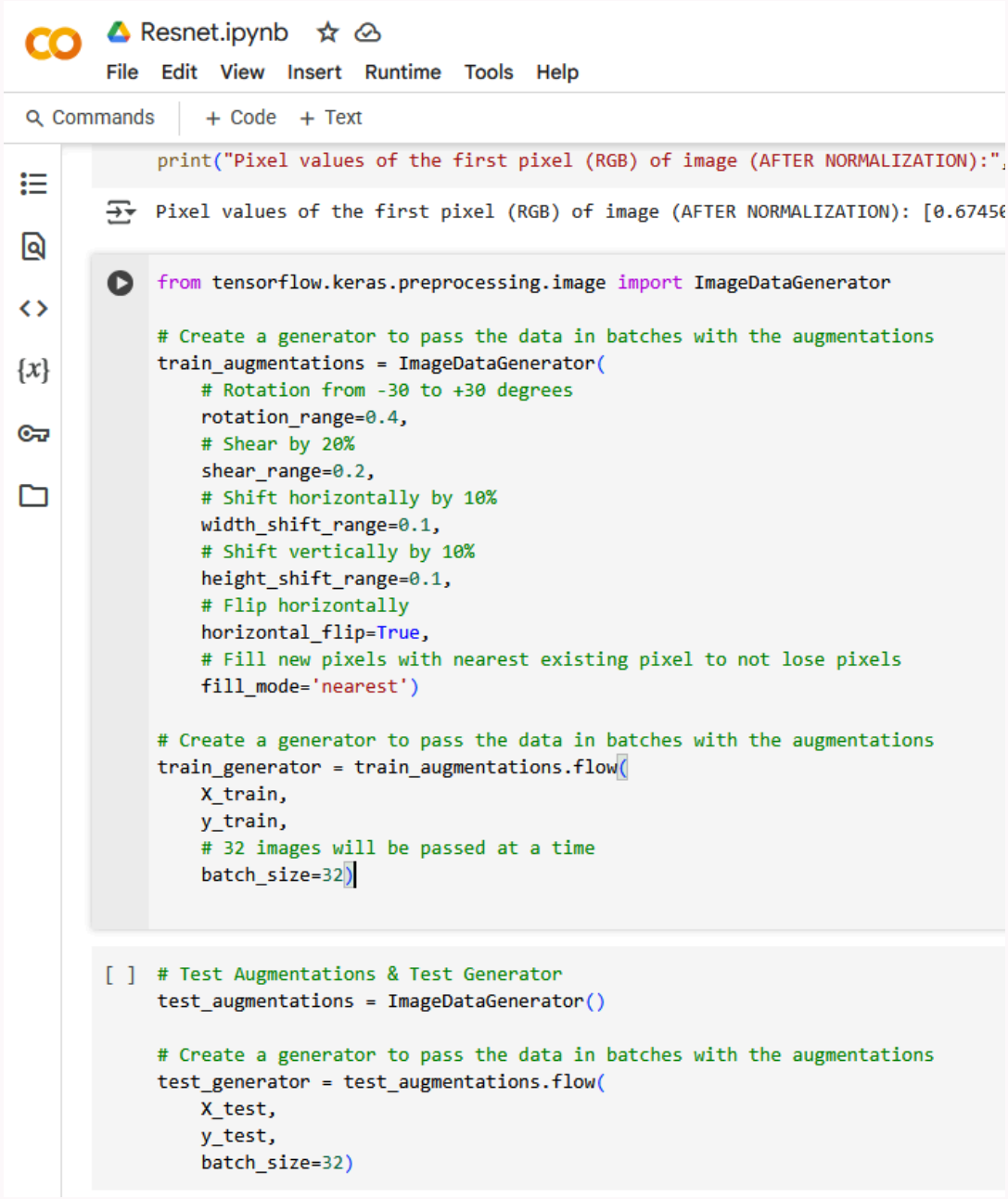
```
(64, 64, 3)
```



- **Normalization**: Pixel values were divided by 255 to scale them between [0,1].
- **One-Hot Encoding**: Labels were converted into one-hot encoded vectors for classification tasks.

2.3 Data Augmentation

- **Applied transformations**: Rotation, zoom, horizontal shift, and vertical shift to increase dataset variability and prevent overfitting.



---

3. Model Architecture

3.1 CNN for Image Classification

- **Input Layer**: 64×64×3 images.
- **Convolutional Layers**: Series of **Conv2D** layers with increasing numbers of filters.
- **MaxPooling**: Used for downsampling the feature maps.
- **Batch Normalization**: Used after convolutions to stabilize the learning process.
- **Dropout**: Used to prevent overfitting during training.
- **Fully Connected Layer**: Dense layer for final classification.

Model: "sequential_1"

Layer (type)	Output Shape	Param #
conv2d_2 (Conv2D)	(None, 62, 62, 32)	896
max_pooling2d_2 (MaxPooling2D)	(None, 31, 31, 32)	0
conv2d_3 (Conv2D)	(None, 29, 29, 64)	18,496
max_pooling2d_3 (MaxPooling2D)	(None, 14, 14, 64)	0
flatten_1 (Flatten)	(None, 12544)	0
dense_2 (Dense)	(None, 64)	802,880
dense_3 (Dense)	(None, 15)	975

Total params: 823,247 (3.14 MB)
Trainable params: 823,247 (3.14 MB)
Non-trainable params: 0 (0.00 B)

Img. Demo From :- Model 1

3.2 Self-Attention CNN

- **Self-Attention Mechanism**: Applied to allow the model to focus on important parts of the image.
- **Skip Connections**: Used to allow the model to learn both low-level and high-level features.

3.3 Cross-Attention CNN

- **Cross-Attention Mechanism**: Uses separate query and key-value projections to focus on different image regions.
- **Global Average Pooling**: Used to reduce dimensionality after convolution layers.

3.4 Regularized CNN

- **Dropout**: To avoid overfitting.
- **Batch Normalization**: To improve stability during training.

---

4. Hyperparameter Tuning

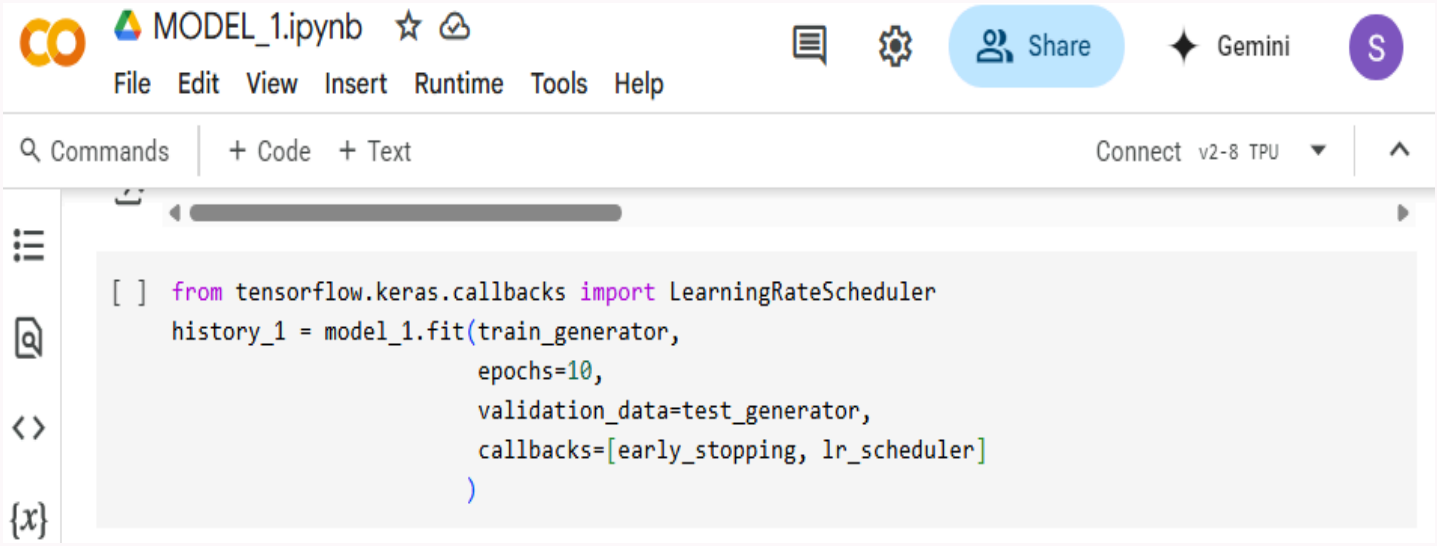
4.1 Hyperparameter Search with Keras Tuner

Keras Tuner was used to optimize the number of filters in Conv2D layers, units in dense layers, and optimizer choices. The **Hyperband** method was employed for efficient hyperparameter search.

---

5. Model Training

5.1 Training the Model



Img. Demo From :- Model 1

The models were trained using **Keras** and **TensorFlow**. We monitored validation accuracy during training and used **EarlyStopping** to prevent overfitting. The final models were saved for future inference.

---

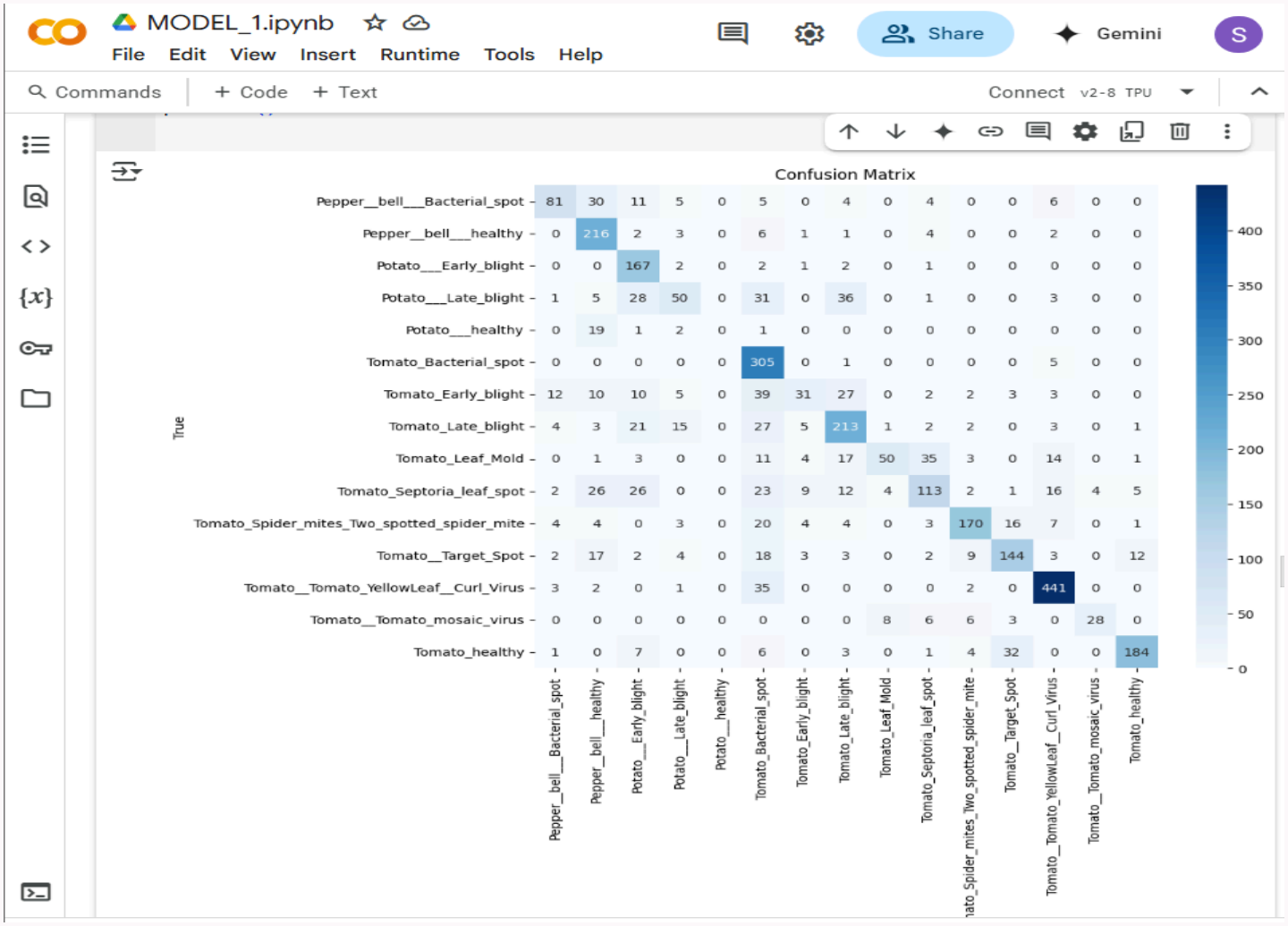
6. Model Evaluation

6.1 Evaluate Model Performance

97/97 ━━━━━━━━━━━━━━━━━━━━ 3s 32ms/step

	precision	recall	f1-score	support
Pepper__bell___Bacterial_spot	0.74	0.55	0.63	146
Pepper__bell___healthy	0.65	0.92	0.76	235
Potato___Early_blight	0.60	0.95	0.74	175
Potato___Late_blight	0.56	0.32	0.41	155
Potato___healthy	0.00	0.00	0.00	23
Tomato_Bacterial_spot	0.58	0.98	0.73	311
Tomato_Early_blight	0.53	0.22	0.31	144
Tomato_Late_blight	0.66	0.72	0.69	297
Tomato_Leaf_Mold	0.79	0.36	0.50	139
Tomato_Septoria_leaf_spot	0.65	0.47	0.54	243
Tomato_Spider_mites_Two_spotted_spider_mite	0.85	0.72	0.78	236
Tomato__Target_Spot	0.72	0.66	0.69	219
Tomato__Tomato_YellowLeaf__Curl_Virus	0.88	0.91	0.89	484
Tomato__Tomato_mosaic_virus	0.88	0.55	0.67	51
Tomato_healthy	0.90	0.77	0.83	238
accuracy			0.71	3096
macro avg	0.67	0.61	0.61	3096
weighted avg	0.72	0.71	0.69	3096

Img. Demo From :- Model 1



Img. Demo From :- Model 1 (Confusion Matrix)

After training, the models were evaluated on the **test data** to check for their accuracy, precision, recall, and F1-score for each plant disease class.

6.2 Model Evaluation

Model	Description	Key Features	Training Accuracy
ResNet50V2	Pre-trained ResNet50V2 for feature extraction with custom dense layers for classification.	Transfer learning, Feature extraction, Efficient architecture	**89%**
Custom CNN (Scratch)	Custom CNN built from scratch with two Conv2D+MaxPooling layers, followed by Dense layers.	Lightweight, No pre-training, Faster training	**71%**
Self-Attention CNN	Custom CNN with multi-head self-attention and skip connections.	Self-attention mechanism, Skip connections, Better feature learning	**91%**
Cross-Attention CNN	Custom CNN with cross-attention mechanism, separate query and key-value projections.	Cross-attention, Separate projections for query/key-value, Improved performance	**88%**
Regularized CNN	CNN with three convolutional layers, regularization, batch normalization, dropout, and softmax output.	Regularization, Batch normalization, Dropout	**92%**

Model	Description	Key Features	Training Accuracy
Simple CNN (Gray-Scale)	Basic CNN using gray-scale input images, three Conv2D layers.	Simplified architecture, Grayscale input	**66.57%**

---

7. Conclusion

7.1 Summary

This project successfully implemented CNN models for plant disease classification, using several architectures, including **ResNet50V2**, **Custom CNN (Scratch)**, **Self-Attention**, **Cross-Attention**, and **Regularized CNN**. The **Regularized CNN** performed the best with an accuracy of **92%**, while the **Simple CNN (Gray-Scale)** model had the lowest performance at **66.57%**.

7.2 Future Work

- **Transfer Learning**: Implement models like **VGG16** or **ResNet** for improved results.
- **Model Deployment**: Deploy the models in a **cloud environment** or as a **mobile app** for real-time plant disease detection.
- **Improving Dataset**: Expand the dataset with more plant species or diverse plant images to improve model accuracy.
- **Multimodal Learning**: Integrate **sensor data** (e.g., temperature, humidity) alongside images for more robust predictions.

---

8. References and Resources

- **PlantVillage Dataset** (for classification tasks).
- **TensorFlow** and **Keras** for building and training the models.
- **Keras Tuner** for hyperparameter optimization.

---

9. Appendix

9.1 Dependencies

- **TensorFlow**: For building and training the deep learning models.
- **Keras**: For high-level API to define neural networks.
- **OpenCV**: For real-time video processing and image manipulation.
- **Keras Tuner**: For hyperparameter optimization.