Introduction

- **Overview of the project**

This application is designed to help users identify plant diseases through uploaded images of plant leaves. It uses pre-trained machine learning models to analyze the images and classify any diseases present within its scope.

Before using the application, users must ensure they have installed the required software, such as Python, and specific Python packages like TensorFlow and Flask.

- **Importance of disease detection in plants**

Detection of diseases in plants is crucial for various reasons. Firstly, it helps in maintaining crop yield and quality by allowing timely intervention to minimize losses. Early detection also reduces economic impacts on farmers and agricultural industries, ensuring stability. Moreover, it contributes to food security by safeguarding production against threats that could compromise quantity and quality.

- **Brief explanation of Convolutional Neural Networks (CNNs)**

Convolutional Neural Networks (CNNs) are a class of deep learning models designed specifically for processing structured grid-like data such as images. They consist of multiple layers, including convolutional layers that extract features from input images by applying filters. (Keita, 2023)

- **Objectives of the project**

The project's goal is to create a web-based application that uses convolutional neural networks (CNNs) to identify and categorize plant diseases. This would help farmers identify diseases early on and make treatment options.

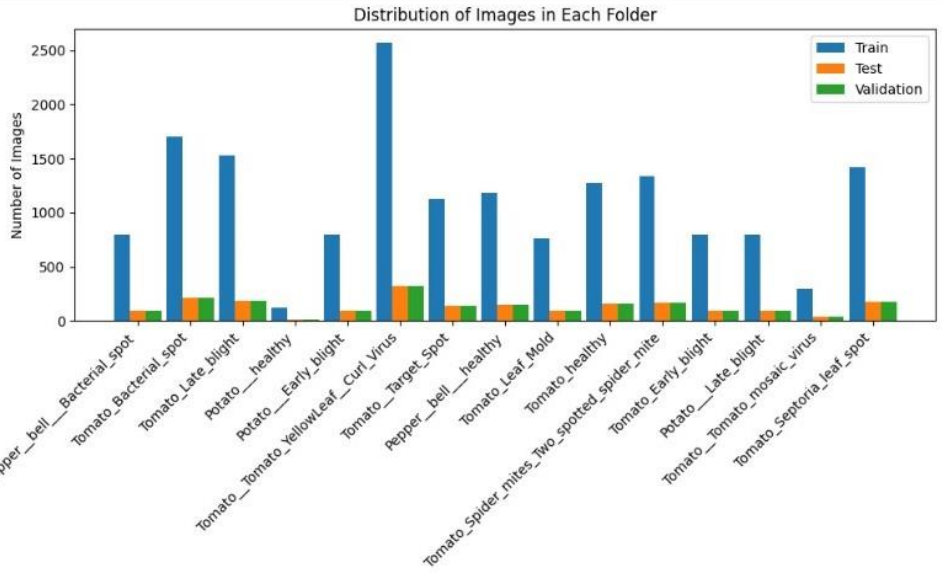
- **Methodology**

- **Data collection and preprocessing**

The dataset was sourced online and underwent preprocessing to ensure consistency and quality. This involved tasks such as resizing images, normalization, and data augmentation to enhance model validity. The preprocessed data was utilized to train the machine learning models for disease detection and classification.

+ Code + Text | Commands Saving...

[6]




```
import matplotlib.image as mpimg

# Create a grid to display the images
num_cols = 3  # Number of columns in the grid
num_rows = 5  # Calculate number of rows
```

+ Code + Text | Commands Saving...

RAM Disk

```
[2] !mkdir -p ~/.kaggle
    !cp kaggle.json ~/.kaggle/kaggle.json
    !chmod 600 ~/.kaggle/kaggle.json
```

Loading Dataset

```
[ ] !pwd

    /content
```

```
[3] !kaggle datasets download -d moazeldsokyx/plantvillage -p /content/


    Downloading plantvillage.zip to /content
     98% 321M/329M [00:03<00:00, 91.3MB/s]
    100% 329M/329M [00:03<00:00, 97.0MB/s]
```

```
!unzip plantvillage.zip -d /content/

  inflating: /content/dataset/validation/Tomato_healthy/991e23a3-fcbf-492b-8972-625779855b0e___RS_HL 0486.JPG
  inflating: /content/dataset/validation/Tomato_healthy/99330465-d609-4f9b-b1cf-da0410f6122c___RS_HL 0388.JPG
  inflating: /content/dataset/validation/Tomato_healthy/9b668abb-fc96-412e-948f-442af45ec580___RS_HL 9972.JPG
  inflating: /content/dataset/validation/Tomato_healthy/9d06e189-6e39-4fb9-853b-e2ce180469a5___RS_HL 0407.JPG
  inflating: /content/dataset/validation/Tomato_healthy/9dab9ebd-6fbf-4565-99c3-06138ca51050___RS_HL 0500.JPG
  inflating: /content/dataset/validation/Tomato_healthy/9f1704fc-28ae-4e27-a6f3-885811ea80e2___GH_HL Leaf 260.JPG
  inflating: /content/dataset/validation/Tomato_healthy/a0e83b60-008d-40b1-b0f4-c3c2e02d8eb3___RS_HL 0092.JPG
```

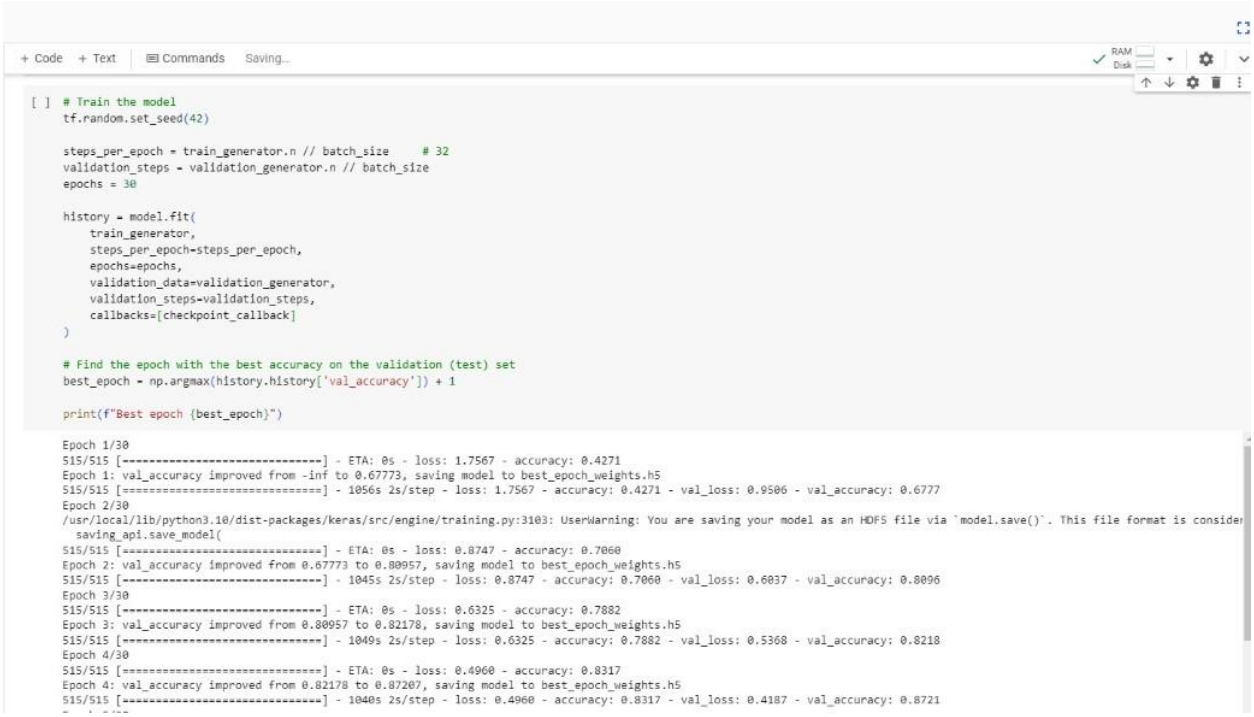
- **Architecture of the CNN model**

The CNN model architecture comprises convolutional layers for feature extraction, pooling layers for dimensionality reduction, and fully connected layers for classification. (Gurucharan, 2022)

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 254, 254, 32)      896

 max_pooling2d (MaxPooling2  (None, 127, 127, 32)      0
 D)

 conv2d_1 (Conv2D)           (None, 125, 125, 64)      18496

 max_pooling2d_1 (MaxPoolin  (None, 62, 62, 64)        0
 g2D)

 conv2d_2 (Conv2D)           (None, 60, 60, 64)        36928

 max_pooling2d_2 (MaxPoolin  (None, 30, 30, 64)        0
 g2D)

 conv2d_3 (Conv2D)           (None, 28, 28, 64)        36928

 max_pooling2d_3 (MaxPoolin  (None, 14, 14, 64)        0
 g2D)

 flatten (Flatten)           (None, 12544)             0

 dense (Dense)               (None, 512)               6423040

 dropout (Dropout)           (None, 512)               0

 dense_1 (Dense)             (None, 128)               65664

 dense_2 (Dense)             (None, 15)                1935

=================================================================
Total params: 6583887 (25.12 MB)
Trainable params: 6583887 (25.12 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

- **Training process**

The process involves feeding batches of training data through the network, computing gradients, and updating weights to improve performance. Training continues until the model converges to a satisfactory level of accuracy or until a predefined number of epochs is reached. (Zhuo, 2019)



```
+ Code + Text | Commands Saving...

[ ] # Train the model
    tf.random.set_seed(42)

    steps_per_epoch = train_generator.n // batch_size     # 32
    validation_steps = validation_generator.n // batch_size
    epochs = 30

    history = model.fit(
        train_generator,
        steps_per_epoch=steps_per_epoch,
        epochs=epochs,
        validation_data=validation_generator,
        validation_steps=validation_steps,
        callbacks=[checkpoint_callback]
    )

    # Find the epoch with the best accuracy on the validation (test) set
    best_epoch = np.argmax(history.history['val_accuracy']) + 1

    print(f"Best epoch {best_epoch}")

    Epoch 1/30
    515/515 [==============================] - ETA: 0s - loss: 1.7567 - accuracy: 0.4271
    Epoch 1: val_accuracy improved from -inf to 0.67773, saving model to best_epoch_weights.h5
    515/515 [==============================] - 1056s 2s/step - loss: 1.7567 - accuracy: 0.4271 - val_loss: 0.9506 - val_accuracy: 0.6777
    Epoch 2/30
    /usr/local/lib/python3.10/dist-packages/keras/src/engine/training.py:3103: UserWarning: You are saving your model as an HDF5 file via `model.save()`. This file format is consider
      saving_api.save_model(
    515/515 [==============================] - ETA: 0s - loss: 0.8747 - accuracy: 0.7060
    Epoch 2: val_accuracy improved from 0.67773 to 0.80957, saving model to best_epoch_weights.h5
    515/515 [==============================] - 1045s 2s/step - loss: 0.8747 - accuracy: 0.7060 - val_loss: 0.6037 - val_accuracy: 0.8096
    Epoch 3/30
    515/515 [==============================] - ETA: 0s - loss: 0.6325 - accuracy: 0.7882
    Epoch 3: val_accuracy improved from 0.80957 to 0.82178, saving model to best_epoch_weights.h5
    515/515 [==============================] - 1049s 2s/step - loss: 0.6325 - accuracy: 0.7882 - val_loss: 0.5368 - val_accuracy: 0.8218
    Epoch 4/30
    515/515 [==============================] - ETA: 0s - loss: 0.4960 - accuracy: 0.8317
    Epoch 4: val_accuracy improved from 0.82178 to 0.87207, saving model to best_epoch_weights.h5
    515/515 [==============================] - 1040s 2s/step - loss: 0.4960 - accuracy: 0.8317 - val_loss: 0.4187 - val_accuracy: 0.8721
```

- **Implementation details**

This implemented CNN model is built and trained using the TensorFlow library in conjunction with the Python programming language. The Flask framework is used to build an interactive web interface that allows users to upload images and view results. During application runtime, the trained models are loaded for real-time inference on uploaded photos.

- **Model Architecture:**

1. The CNN model uses a Sequential architecture, comprising multiple layers for feature extraction and classification.
2. It consists of four Convolutional layers with increasing filter sizes (32, 64, 64, 64) and ReLU activation functions, followed by MaxPooling layers to downsample feature maps.
3. The model includes two fully connected Dense layers with ReLU activation (512 neurons, 128 neurons), along with a Dropout layer with a dropout rate of 0.2 to reduce overfitting.
4. The final Dense layer consists of 15 neurons with a softmax activation function for multiclass classification.

- **Model Compilation:**

1. The model is compiled using the Adam optimizer with default parameters and categorical cross-entropy loss function for multiclass classification.
2. Accuracy is chosen as the evaluation metric to monitor the model's performance during training.

- **Model Checkpoint Callback:**

1. A ModelCheckpoint callback is defined to save the best weights of the model based on the validation accuracy.
2. The callback monitors the validation accuracy and saves the weights to "best_epoch_weights.h5" file whenever an improvement is observed.
3. Only the weights yielding the highest validation accuracy are saved, ensuring the preservation of the best performing model configuration.

These specifications summarize the architecture, compilation, and checkpointing configuration of the CNN model for training.

- **Data Visualization**

- **Data Collection:**

1. The code iterates through each folder ('train', 'test', 'validation') and its subfolders.
2. It calculates the number of images in each subfolder and stores this information in the subfolder_lengths dictionary.

- **Bar Chart Creation:**

1. Using the collected data, the code creates a grouped bar chart.
2. Three sets of bars represent the image distribution in the 'train', 'test', and 'validation' folders.
3. Each group of bars corresponds to a specific subfolder, and the height of the bars indicates the number of images in that subfolder.

- **Visualization Parameters:**

1. The x-axis displays the subfolder names, providing a clear distinction between different categories.
2. Colors are assigned to differentiate between the 'Train', 'Test', and 'Validation' folders, aiding in easy comparison.

- **Insights:**

1. The resulting visualization offers insights into the dataset's composition and distribution.
2. Users can easily interpret data adequacy across folders, identifying potential imbalances or biases in the dataset.

- **Training/validation split**

```
[ ]     target_size = image_shape,
        batch_size = batch_size,
        class_mode = "categorical"

    )

    Found 16504 images belonging to 15 classes.

[ ] validation_datagen = ImageDataGenerator(
        rescale=1/255,
    )

    validation_generator = validation_datagen.flow_from_directory(
        validation_path,
        target_size = image_shape,
        batch_size = batch_size,
        class_mode = "categorical"

    )

    Found 2070 images belonging to 15 classes.

[ ] test_datagen = ImageDataGenerator(
        rescale=1/255,
    )

    test_generator = test_datagen.flow_from_directory(
        test_path,
        target_size = image_shape,
        batch_size = batch_size,
        class_mode = "categorical",
        shuffle = False

    )

    Found 2064 images belonging to 15 classes.
```

Training Data:

The train_datagen is configured to generate batches of augmented image data from the train_path, which corresponds to a directory containing subfolders representing different classes of training images.

Validation Data:

Similarly, the validation_datagen is set up to generate batches of image data from the validation_path, which contains a separate set of subfolders representing validation images.

- **Results**

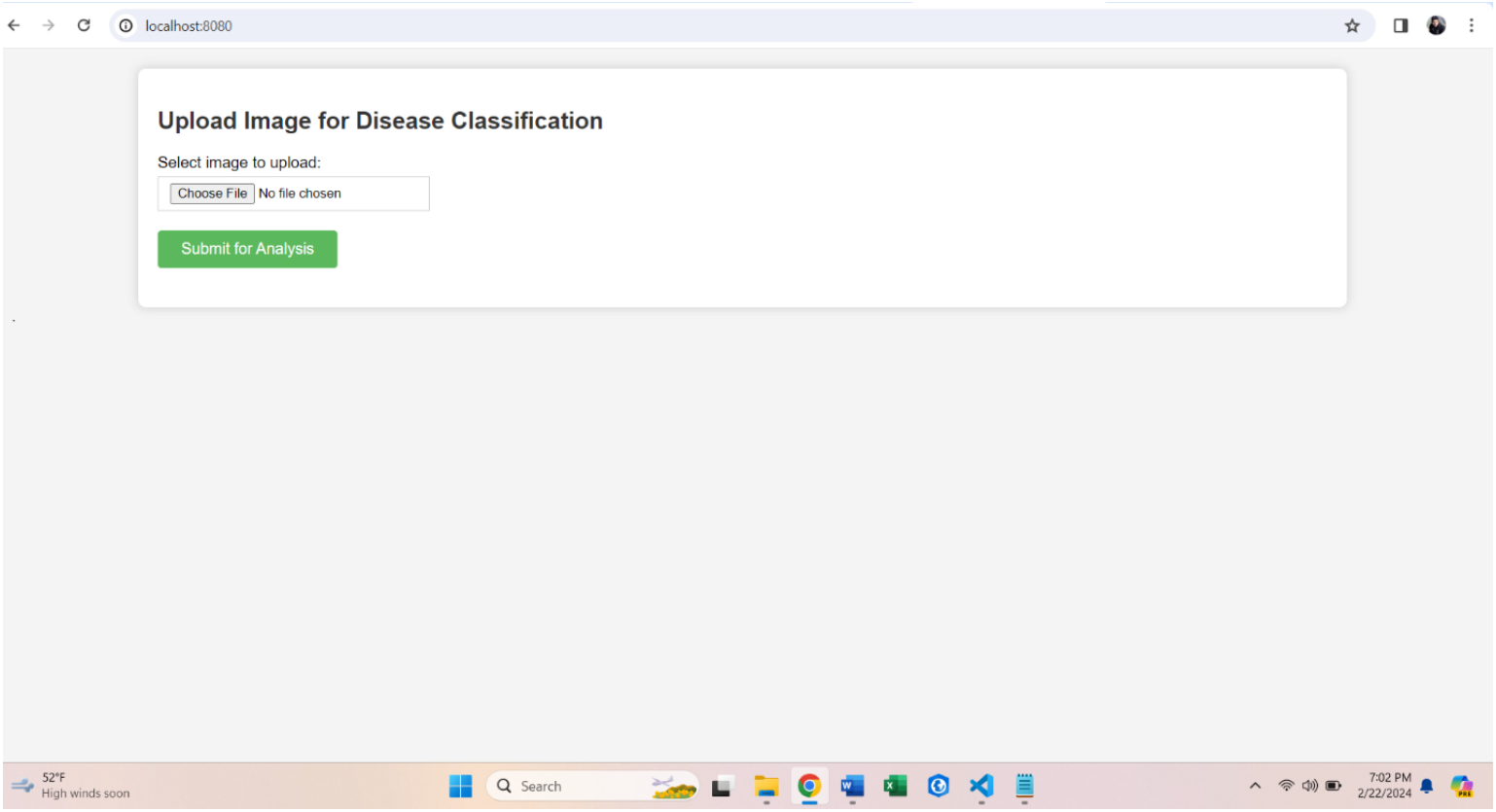
According to the training logs, the best epoch is epoch 30, as indicated by the highest validation accuracy achieved during that epoch. Here are the results generated from the training:

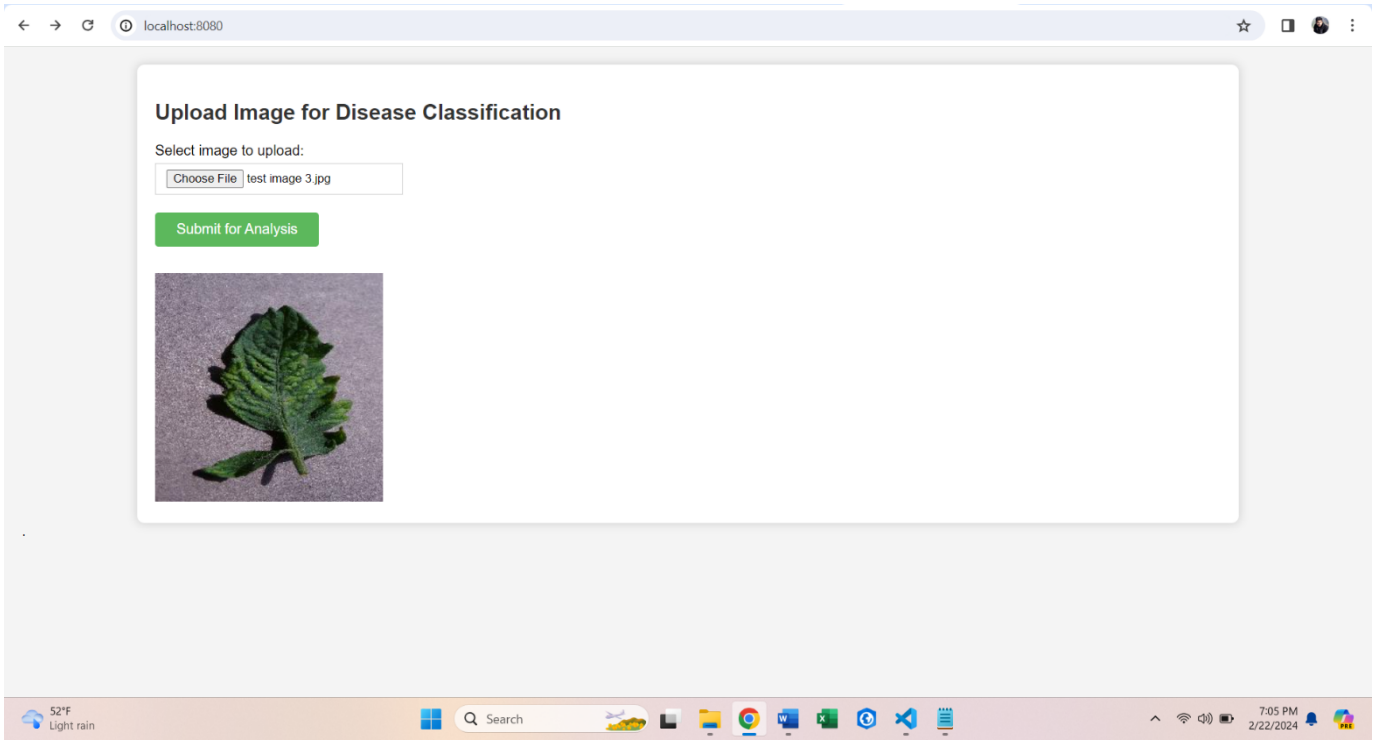
- Best Epoch: Epoch 30
- Validation Accuracy: 96.97%
- Validation Loss: 0.1162

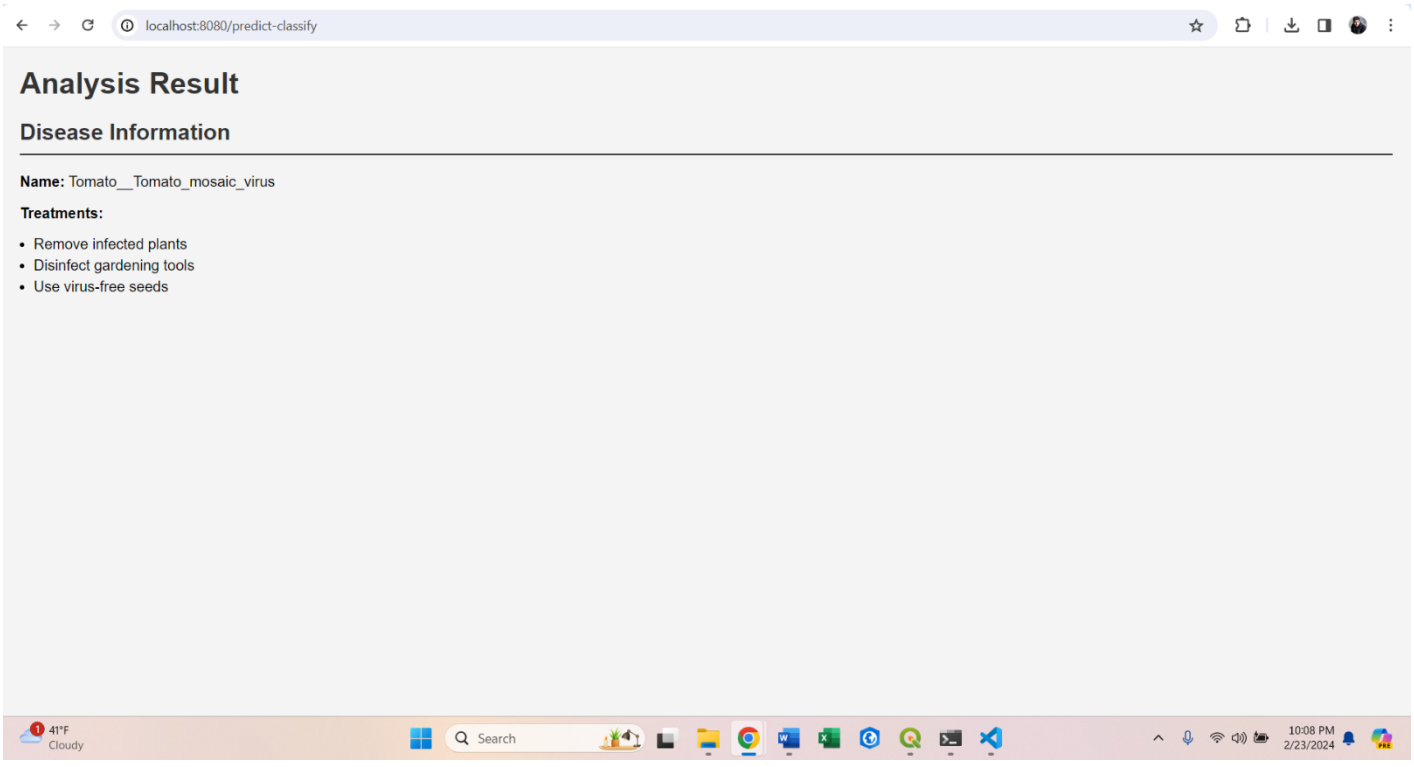
These results suggest that the model achieved its highest performance on the validation dataset during epoch 30, with an accuracy of 96.97% and a loss of 0.1162. This indicates that the model generalized well to unseen data and performed effectively in classifying the validation images.

- **Application**
- **User interface design**

The user interface design is simple and efficient, featuring basic functionality that allows users to upload images effortlessly. Once uploaded, you can press the button “Submit for Analysis” which provides users with the corresponding results, ensuring a seamless experience.




localhost:8080
Upload Image for Disease Classification
Select image to upload:
Choose File
test image 3.jpg
Submit for Analysis
52°F
Light rain
Search
7:05 PM
2/22/2024


localhost:8080/predict-classify
Analysis Result
Disease Information
Name: Tomato__Tomato_mosaic_virus
Treatments:
Remove infected plants
Disinfect gardening tools
Use virus-free seeds
41°F
Cloudy
Search
10:08 PM
2/23/2024

- **Functionality of the app**

To use the app, take the following fundamental actions.

1. Upload Image: Using the app's UI, users can upload pictures of leaves.
2. Processing: The application uses its disease detection algorithm to examine the provided photos.
3. Result Display: Once processed, the app displays the detected diseases and provides recommended treatments related to them.

- **Potential impact on agriculture and farmers**

The app's disease detection abilities can greatly impact agriculture by enabling early identification of plant illnesses, resulting in timely interventions and reduced crop losses.

Farmers can benefit from improved disease management practices, resulting in increased yields and profitability,

- **Conclusion**

  The app provides a valuable solution for detecting plant illnesses through advanced technology, which has the potential to revolutionize agricultural methods. The app can provide farmers with timely insights and boost crop health and productivity. This app can manage agricultural issues and advance sustainable farming methods for more secure food chains.

References

Gurucharan, M., 2022. *Basic CNN Architecture: Explaining 5 Layers of Convolutional Neural Network*. [Online]
Available at: https://www.upgrad.com/blog/basic-cnn-architecture/
[Accessed 2024].

Keita, Z., 2023. *An Introduction to Convolutional Neural Networks (CNNs)*. [Online]
Available at: https://www.datacamp.com/tutorial/introduction-to-convolutional-neural-networks-cnns
[Accessed 2024].

Zhuo, V., 2019. *Training a Convolutional Neural Network from scratch*. [Online]
Available at: https://towardsdatascience.com/training-a-convolutional-neural-network-from-scratch-2235c2a25754
[Accessed 2024].