PROJECT TEAM -10

PLANT DISEASE DETECTION FROM LEAF IMAGES

PRESENTED BY,

Beena Kurian

Chitrank Dave

OBJECTIVES:

The primary objective is to develop a plant disease detection using leaf images.

To improve the accuracy of existing models.

To build plant disease detection applications to aid farmers/agricultural experts in the early detection of diseases.

MOTIVATION:

Plants are a significant source of food for the world population.

Proper identification of the disease, disease-causing agents, and disease control measures are necessary to save time and money.

This also enables researchers to implement timely and effective remedial measures.

Plant diseases contribute to production loss, which can be solved with continuous monitoring.

The problem of plant disease detection from leaf images is attractive due to its real-world impact on agriculture.

Detecting affected plants is one of the first steps in diagnosing a plant disease.

Building an application will help not only the farmers but also the agricultural experts to identify new diseases spreading in a particular area

REVIEW OF THE DATA SOURCE

PlantVillage dataset [1]

- It contains 15 classes representing **12 diseases of 3 plants (potatoes, bell peppers, and tomatoes).**
- One class is dedicated to representing healthy leaf images of each type.

New Plant Diseases Dataset [2]

- 38 classes
- To predict more plant diseases associated with other plant leaves like **apples, blueberries, cherries, corn, grapes, oranges, peaches, raspberries, soybeans, squash, tomatoes, and strawberries.**
- This project's model is trained using this 38-class dataset.

DATASET(38 CLASSES) – 14 PLANT LEAVES

- Apple___Apple_scab
- Apple___Black_rot
- Apple___Cedar_apple_rust
- Apple___healthy
- Blueberry___healthy
- Cherry_(including_sour)___Powdery_mildew
- Cherry_(including_sour)___healthy
- Corn_(maize)___Cercospora_leaf_spot Gray_leaf_spot
- Corn_(maize)___Common_rust_
- Corn_(maize)___Northern_Leaf_Blight
- Corn_(maize)___healthy
- Grape___Black_rot
- Grape___Esca_(Black_Measles)
- Grape___Leaf_blight_(Isariopsis_Leaf_Spot)
- Grape___healthy
- Orange___Haunglongbing_(Citrus_greening)
- Peach___Bacterial_spot
- Peach___healthy
- Pepper,_bell___Bacterial_spot
- Pepper,_bell___healthy
- Potato___Early_blight
- Potato___Late_blight
- Potato___healthy
- Raspberry___healthy
- Soybean___healthy
- Squash___Powdery_mildew
- Strawberry___Leaf_scorch
- Strawberry___healthy
- Tomato___Bacterial_spot
- Tomato___Early_blight

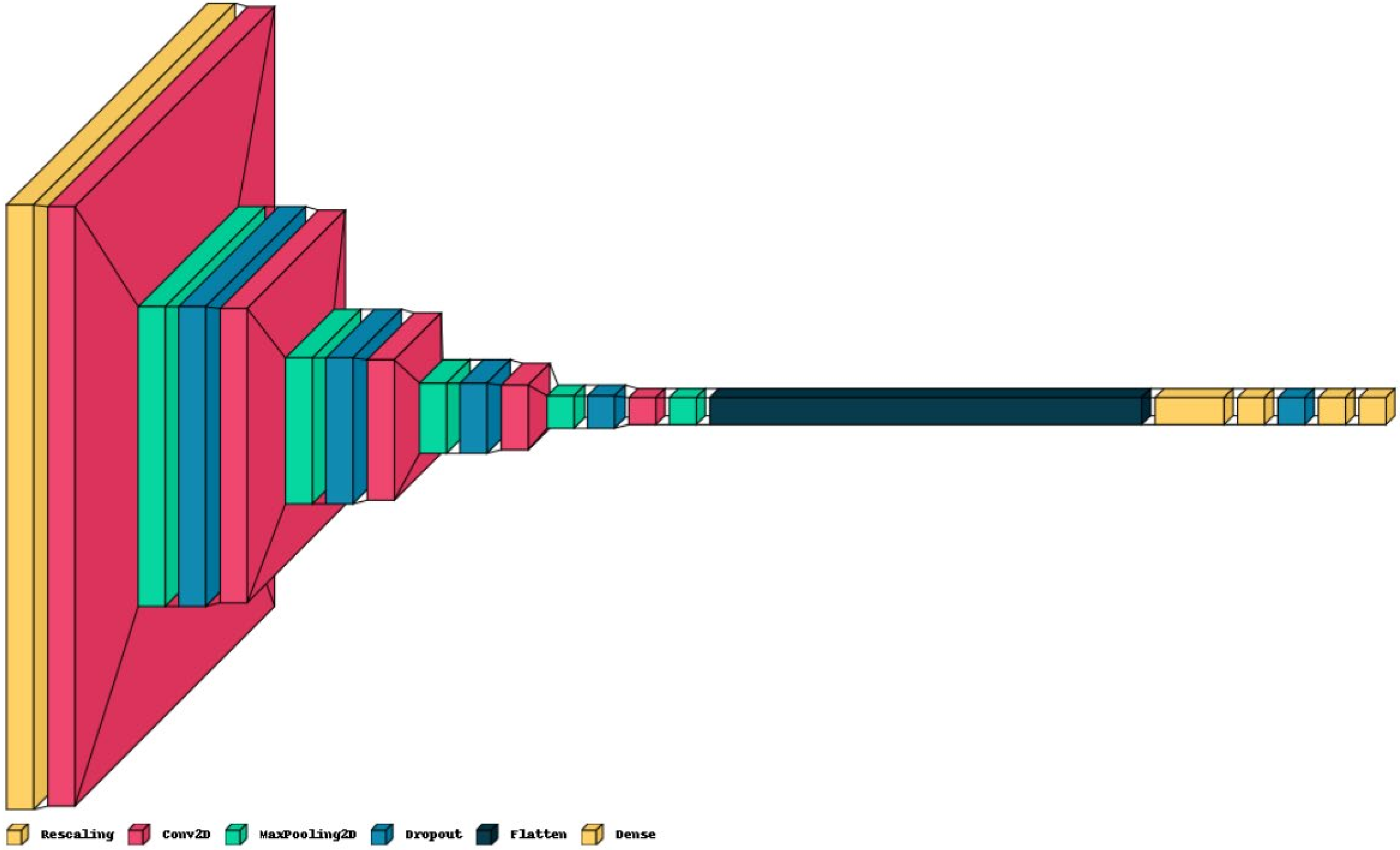
LOADING DATASET :

- Used **TensorFlow's image_dataset_from_directory** utility to create training and testing datasets.
- Training dataset: Setting shuffle=True
  - Reduce overfitting
  - Preventing it from learning spurious patterns
  - Better Generalization

```
# Prefetch the datasets
train_ds = train_ds.prefetch(tf.data.AUTOTUNE)
test_ds = test_ds.prefetch(tf.data.AUTOTUNE)
```

- Prefetching helped with asynchronous data loading.
- While the **GPU is processing the current batch of data**, the **CPU can prefetch the next batch** simultaneously.
- This helped **minimize the idle time of the GPU**, resulting in **better overall training efficiency.**
- This is particularly beneficial when dealing with **large datasets or training deep neural networks** requiring significant computational resources.

MODEL ARCHITECTURE



MODEL

- <u>Data Preprocessing</u>: The first layer, Rescaling, is used for preprocessing the input images. It scales pixel values to the range [0,1] by dividing them by 255.
- **Convolutional Layers:** The model includes several convolutional layers (Conv2D) with ReLU activation.
- **MaxPool2D** : Followed by max-pooling layers (MaxPool2D)
- **Dropout Layers**: Dropout layers (Dropout) prevent overfitting.
- **Flatten Layer**: It converts the 2D output from the convolutional layers into a 1D vector.
  - The flattened vector is fed into a series of fully connected (Dense) layers.
- **Fully Connected Layers**: The last layer uses '**softmax activation**' to convert the network's final outputs into 38 class probabilities.
  - **A dropout layer is added to reduce overfitting.**

MODEL

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 rescaling (Rescaling)       (None, 224, 224, 3)       0

 conv2d (Conv2D)             (None, 222, 222, 32)      896

 max_pooling2d (MaxPooling2D  (None, 111, 111, 32)     0
 )

 dropout (Dropout)           (None, 111, 111, 32)      0

 conv2d_1 (Conv2D)           (None, 109, 109, 64)      18496

 max_pooling2d_1 (MaxPooling  (None, 54, 54, 64)       0
 2D)

 dropout_1 (Dropout)         (None, 54, 54, 64)        0

 conv2d_2 (Conv2D)           (None, 52, 52, 64)        36928

 max_pooling2d_2 (MaxPooling  (None, 26, 26, 64)       0
 2D)

 dropout_2 (Dropout)         (None, 26, 26, 64)        0

 conv2d_3 (Conv2D)           (None, 24, 24, 64)        36928

 max_pooling2d_3 (MaxPooling  (None, 12, 12, 64)       0
 2D)

 dropout_3 (Dropout)         (None, 12, 12, 64)        0

 conv2d_4 (Conv2D)           (None, 10, 10, 128)       73856

 max_pooling2d_4 (MaxPooling  (None, 5, 5, 128)        0
 2D)

 flatten (Flatten)           (None, 3200)              0

 dense (Dense)               (None, 512)               1638912

 dense_1 (Dense)             (None, 128)               65664

 dropout_4 (Dropout)         (None, 128)               0

 dense_2 (Dense)             (None, 64)                8256

 dense_3 (Dense)             (None, 38)                2470

=================================================================
Total params: 1,882,406
Trainable params: 1,882,406
Non-trainable params: 0
_________________________________________________________________
```

```python
model = keras.Sequential([
    keras.layers.Rescaling(scale = 1/255 , input_shape =(224,224,3) )

    keras.layers.Conv2D(32 , (3,3) , activation = 'relu'),
    keras.layers.MaxPool2D((2,2))                          ,
    keras.layers.Dropout(0.2),

    keras.layers.Conv2D(64 , (3,3) , activation = 'relu') ,
    keras.layers.MaxPool2D((2,2)) ,
    keras.layers.Dropout(0.2),

    keras.layers.Conv2D(64 , (3,3) , activation = 'relu') ,
    keras.layers.MaxPool2D((2,2)) ,
    keras.layers.Dropout(0.2),

    keras.layers.Conv2D(64 , (3,3) , activation = 'relu') ,
    keras.layers.MaxPool2D((2,2)) ,
    keras.layers.Dropout(0.2),

    keras.layers.Conv2D(128 , (3,3) , activation = 'relu') ,
    keras.layers.MaxPool2D((2,2)) ,

    # fully connected layers
    keras.layers.Flatten(),
    keras.layers.Dense(512,activation = 'relu'),
    keras.layers.Dense(128,activation = 'relu'),
    keras.layers.Dropout(0.2),
    keras.layers.Dense(64,activation = 'relu'),
    keras.layers.Dense(38,activation ='softmax')

])
```

MODEL TRAINING(WITHOUT VALIDATION)

```
history = model.fit(train_ds , epochs = 15)
```

```
Epoch 1/15
679/679 [==============================] - 123s 160ms/step - loss: 2.0527 - accuracy: 0.4316
Epoch 2/15
679/679 [==============================] - 105s 154ms/step - loss: 0.8411 - accuracy: 0.7426
Epoch 3/15
679/679 [==============================] - 104s 153ms/step - loss: 0.5007 - accuracy: 0.8427
Epoch 4/15
679/679 [==============================] - 103s 151ms/step - loss: 0.3647 - accuracy: 0.8843
Epoch 5/15
679/679 [==============================] - 102s 149ms/step - loss: 0.2853 - accuracy: 0.9084
Epoch 6/15
679/679 [==============================] - 102s 149ms/step - loss: 0.2341 - accuracy: 0.9255
Epoch 7/15
679/679 [==============================] - 101s 149ms/step - loss: 0.2009 - accuracy: 0.9359
Epoch 8/15
679/679 [==============================] - 101s 148ms/step - loss: 0.1774 - accuracy: 0.9440
Epoch 9/15
679/679 [==============================] - 101s 148ms/step - loss: 0.1593 - accuracy: 0.9502
Epoch 10/15
679/679 [==============================] - 100s 147ms/step - loss: 0.1343 - accuracy: 0.9575
Epoch 11/15
679/679 [==============================] - 100s 147ms/step - loss: 0.1263 - accuracy: 0.9600
Epoch 12/15
679/679 [==============================] - 100s 147ms/step - loss: 0.1146 - accuracy: 0.9641
Epoch 13/15
679/679 [==============================] - 100s 146ms/step - loss: 0.1173 - accuracy: 0.9640
Epoch 14/15
679/679 [==============================] - 100s 146ms/step - loss: 0.0978 - accuracy: 0.9701
Epoch 15/15
679/679 [==============================] - 100s 146ms/step - loss: 0.0965 - accuracy: 0.9710
```

TESTING ACCURACY

```
model.evaluate(test_ds)
```

```
170/170 [==============================] - 14s 83ms/step - loss: 0.0773 - accuracy: 0.9818
[0.0772799625992775, 0.9817696213722229]
```

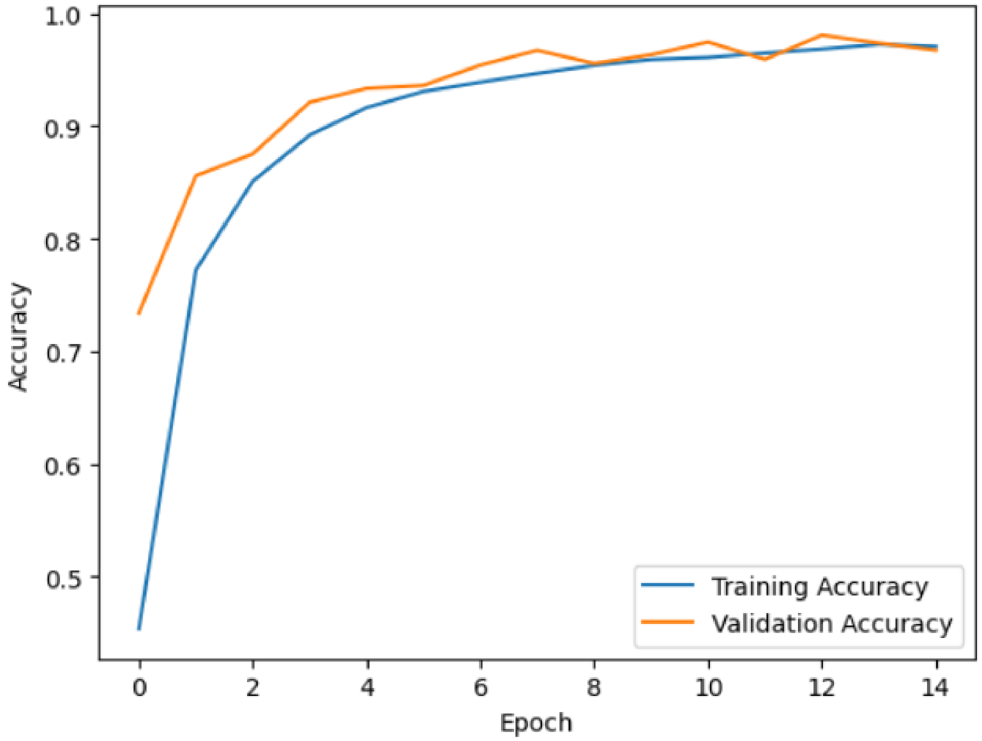
MODEL TRAINING

```
model.compile(
    optimizer = 'adam' ,
    loss = 'sparse_categorical_crossentropy',
    metrics = 'accuracy'
)
```

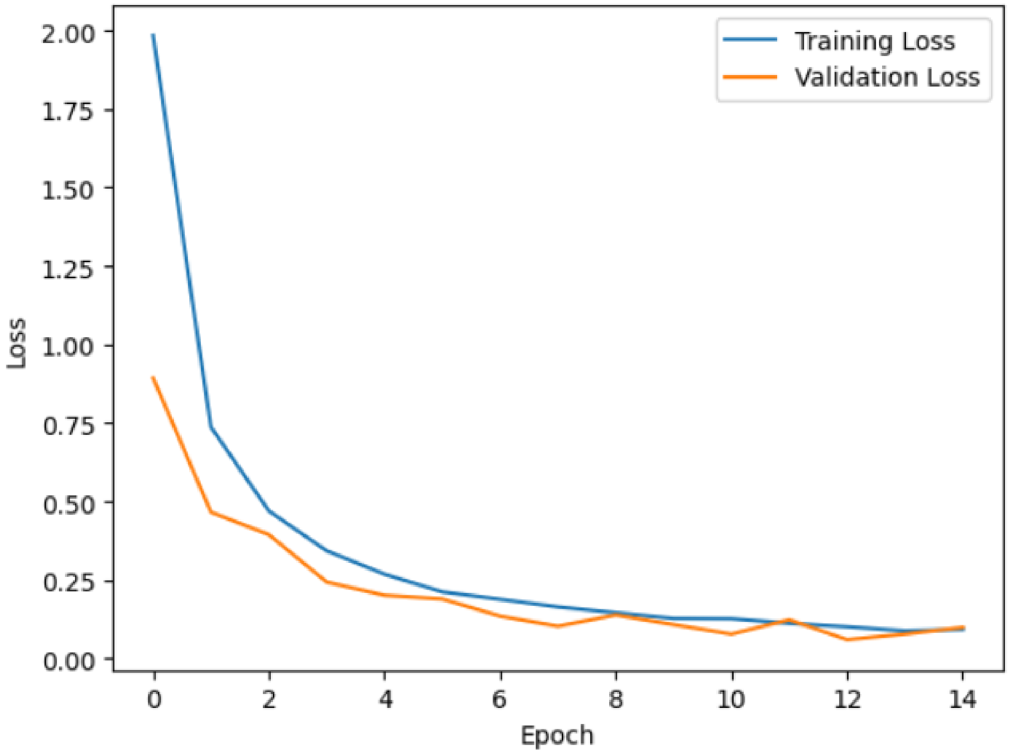
```
history = model.fit(train_ds , epochs = 15, validation_split=0.2, validation_data=train_ds)
```

```
Epoch 1/15
679/679 [==============================] - 137s 202ms/step - loss: 1.9843 - accuracy: 0.4536 - val_loss: 0.8934 - val_accuracy: 0.7339
Epoch 2/15
679/679 [==============================] - 130s 192ms/step - loss: 0.7372 - accuracy: 0.7721 - val_loss: 0.4663 - val_accuracy: 0.8558
Epoch 3/15
679/679 [==============================] - 135s 199ms/step - loss: 0.4704 - accuracy: 0.8508 - val_loss: 0.3950 - val_accuracy: 0.8751
Epoch 4/15
679/679 [==============================] - 135s 199ms/step - loss: 0.3433 - accuracy: 0.8918 - val_loss: 0.2439 - val_accuracy: 0.9210
Epoch 5/15
679/679 [==============================] - 136s 199ms/step - loss: 0.2688 - accuracy: 0.9162 - val_loss: 0.2017 - val_accuracy: 0.9334
Epoch 6/15
679/679 [==============================] - 134s 198ms/step - loss: 0.2126 - accuracy: 0.9306 - val_loss: 0.1900 - val_accuracy: 0.9360
Epoch 7/15
679/679 [==============================] - 136s 200ms/step - loss: 0.1890 - accuracy: 0.9388 - val_loss: 0.1351 - val_accuracy: 0.9540
Epoch 8/15
679/679 [==============================] - 134s 197ms/step - loss: 0.1648 - accuracy: 0.9466 - val_loss: 0.1031 - val_accuracy: 0.9671
Epoch 9/15
679/679 [==============================] - 139s 204ms/step - loss: 0.1460 - accuracy: 0.9538 - val_loss: 0.1388 - val_accuracy: 0.9555
Epoch 10/15
679/679 [==============================] - 133s 196ms/step - loss: 0.1278 - accuracy: 0.9589 - val_loss: 0.1081 - val_accuracy: 0.9633
Epoch 11/15
679/679 [==============================] - 135s 198ms/step - loss: 0.1270 - accuracy: 0.9608 - val_loss: 0.0782 - val_accuracy: 0.9744
Epoch 12/15
679/679 [==============================] - 134s 197ms/step - loss: 0.1124 - accuracy: 0.9647 - val_loss: 0.1235 - val_accuracy: 0.9591
Epoch 13/15
679/679 [==============================] - 133s 196ms/step - loss: 0.1013 - accuracy: 0.9682 - val_loss: 0.0603 - val_accuracy: 0.9806
Epoch 14/15
679/679 [==============================] - 134s 197ms/step - loss: 0.0884 - accuracy: 0.9724 - val_loss: 0.0782 - val_accuracy: 0.9735
Epoch 15/15
679/679 [==============================] - 132s 193ms/step - loss: 0.0927 - accuracy: 0.9705 - val_loss: 0.0992 - val_accuracy: 0.9673
```

TRAINING ACCURACY AND VALIDATION ACCURACY



TRAINING LOSS AND VALIDATION LOSS



TESTING ACCURACY

```
# Evaluate the model on the test dataset
evaluation_result = model.evaluate(test_ds)

# The evaluation_result will contain the loss and metrics specified during model compilation
print("Test Loss:", evaluation_result[0])
print("Test Accuracy:", evaluation_result[1])
```

```
170/170 [==============================] - 9s 54ms/step - loss: 0.1839 - accuracy: 0.9398
Test Loss: 0.1839224249124527
Test Accuracy: 0.9397845268249512
```

COMPARISON WITH OTHER MODELS

COMPARISON-1

https://www.kaggle.com/code/emmarex/plant-disease-detection-using-keras/notebook [3]

- **Handling 15 classes**
- **Total params: 58,102,671**
- **Trainable params: 58,099,791**
- **Non-trainable params: 2,880**
- **Batch Size:32**
- **Epoch: 25**
- **Test Accuracy:96.77 %(without validation)**

New Model

- **Handling 38 classes**
- **Total params: 1,882,406**
- **Trainable params: 1,882,406**
- **Non-trainable params: 0**
- **Batch Size:64**
- **Epoch:15**
- **Test Accuracy:98.18 %(without validation)**

```
flatten (Flatten)            (None, 3200)          0

dense (Dense)                (None, 128)           409728

dense_1 (Dense)              (None, 64)            8256

dense_2 (Dense)              (None, 38)            2470

=================================================================
Total params: 587,558
Trainable params: 587,558
Non-trainable params: 0
```

COMPARISON-2

```
flatten_1 (Flatten)          (None, 3200)          0

dense_4 (Dense)              (None, 512)           1638912

dense_5 (Dense)              (None, 128)           65664

dropout_9 (Dropout)          (None, 128)           0

dense_6 (Dense)              (None, 64)            8256

dense_7 (Dense)              (None, 38)            2470

=================================================================
Total params: 1,882,406
Trainable params: 1,882,406
Non-trainable params: 0
_________________________________________________________________
```

https://www.kaggle.com/code/abdulrahmankhaled1/plant-disease-detection/notebook [4]

- **Handling 38 classes**
- **Total params: 587,558**
- **Trainable params: 587,558**
- **Non-trainable params:0**
- **Batch Size:32**
- **Epoch: 25**

when validation is not used

- **Training Accuracy : 96.92%**
- **Test Accuracy: 90.86 %(without validation)**

New Model

- **Handling 38 classes (Same Base Layers used, modified fully connected layers)**
- **Total params: 1,882,406 , Trainable params: 1,882,406, Non-trainable params: 0**
- **Batch Size:64**
- **Epoch:15**
- **use of prefetch in TensorFlow datasets**

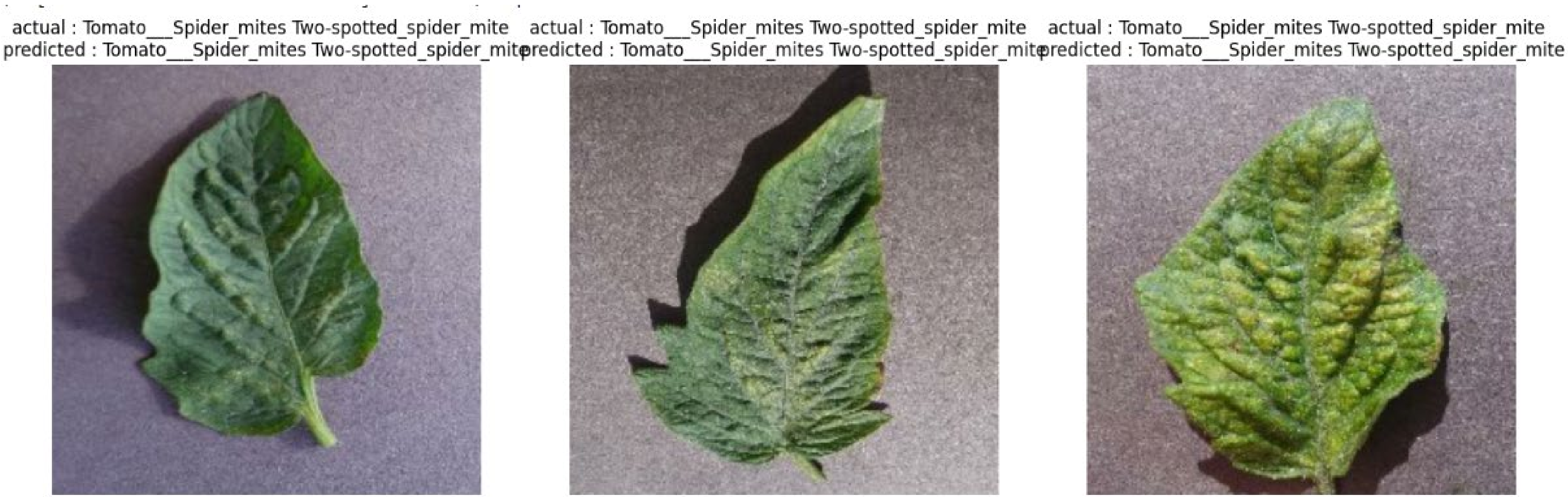
when validation is not used

- **Training Accuracy : 97.10%**
- **Test Accuracy:98.18%**

when validation is used

- **Train accuracy : 97.05%**
- **Validation Accuracy : 96.73**
- **Test Accuracy:93.98 %**

PREDICTION VISUALISATION



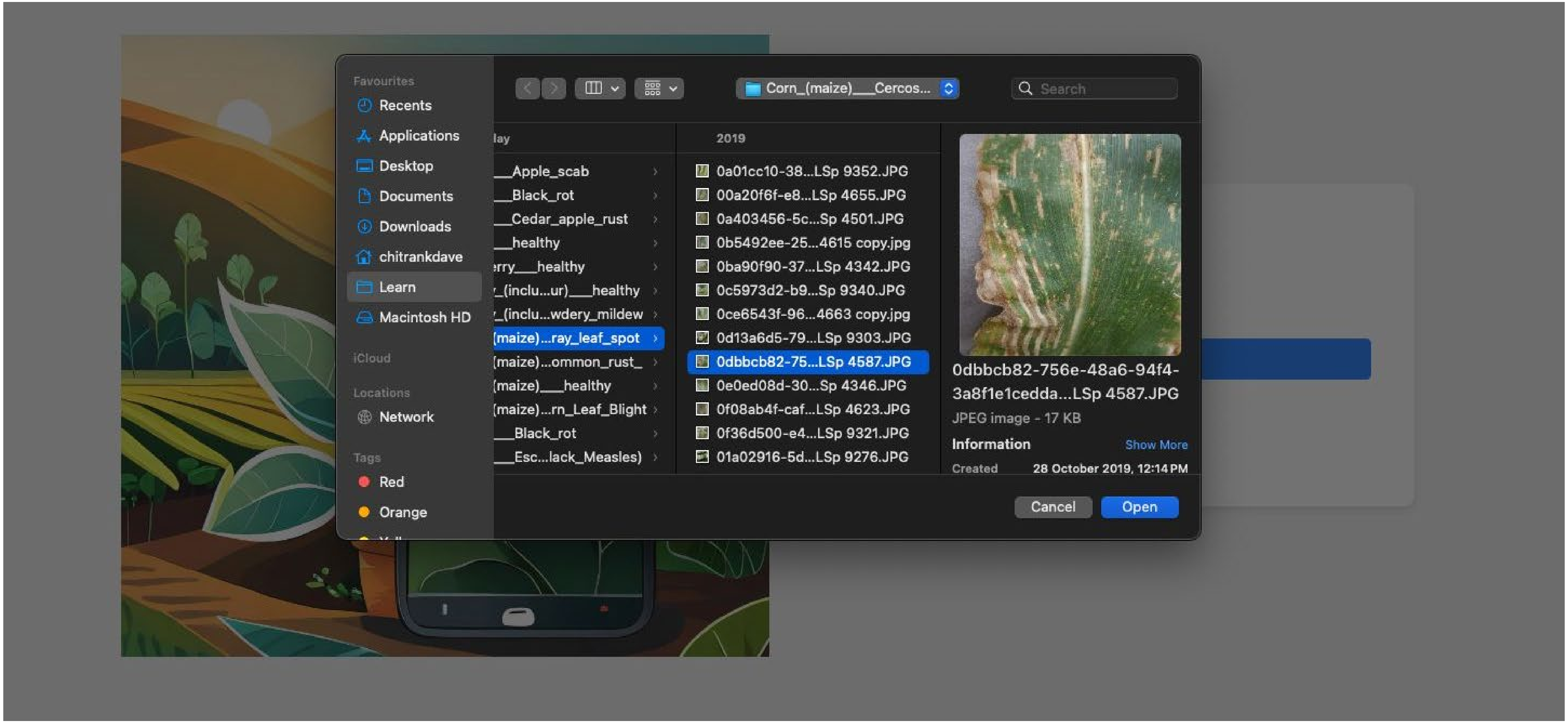
PROJECT DEMO

Plant Disease Detection

Choose file No file chosen

Upload your image

Detect


Favourites
Recents
Applications
Desktop
Documents
Downloads
chitrankdave
Learn
Macintosh HD
iCloud
Locations
Network
Tags
Red
Orange
Corn_(maize)___Cercos...
Search
2019
___Apple_scab
___Black_rot
___Cedar_apple_rust
___healthy
rry___healthy
_(inclu...ur)___healthy
_(inclu...wdery_mildew
(maize)...ray_leaf_spot
(maize)...ommon_rust_
(maize)___healthy
(maize)...rn_Leaf_Blight
___Black_rot
___Esc...lack_Measles)
0a01cc10-38...LSp 9352.JPG
00a20f6f-e8...LSp 4655.JPG
0a403456-5c...Sp 4501.JPG
0b5492ee-25...4615 copy.jpg
0ba90f90-37...LSp 4342.JPG
0c5973d2-b9...Sp 9340.JPG
0ce6543f-96...4663 copy.jpg
0d13a6d5-79...LSp 9303.JPG
0dbbcb82-75...LSp 4587.JPG
0e0ed08d-30...Sp 4346.JPG
0f08ab4f-caf...LSp 4623.JPG
0f36d500-e4...LSp 9321.JPG
01a02916-5d...LSp 9276.JPG
0dbbcb82-756e-48a6-94f4-
3a8f1e1cedda...LSp 4587.JPG
JPEG image - 17 KB
Information
Show More
Created
28 October 2019, 12:14 PM
Cancel
Open

Prediction: Corn_(maize)___Cercospora_leaf_spot Gray_leaf_spot

Try again

CHALLENGES FACED

- Dataset Download and Extraction
- Training Time was high because of 3 GB data
  - Waiting time for changing hyperparameter tuning.

TAKE AWAYS..

- Setting the shuffle attribute during training has proven essential, preventing the model from memorizing patterns and promoting generalization across diverse datasets.
- The introduction of prefetching has significantly improved the overall training speed and efficiency.
- Iterative adjustments to hyperparameters have proven effective in finding the right balance, resulting in a more robust and accurate plant disease detection model.
- The careful analysis of accuracy and loss during training and testing has allowed us to strike a delicate balance.
- Utilizing Kaggle as a collaborative platform has greatly facilitated our project. The ability to edit existing notebooks, save versions, and compare performance with other models has streamlined collaboration and allowed for continuous improvement.
- It's been a journey of learning, overcoming challenges, and ultimately creating a tool that holds promise in addressing real-world problems.

REFERENCES:

1. https://www.kaggle.com/datasets/emmarex/plantdisease/data
2. https://www.kaggle.com/datasets/vipoooool/new-plant-diseases-dataset
3. https://www.kaggle.com/code/emmarex/plant-disease-detection-using-keras
4. https://www.kaggle.com/code/abdulrahmankhaled1/plant-disease-detection/notebook

THANK YOU