Group 7: Coffee Plant Disease Detection

Hammad Ansari Manik Shekhar Faizal Khan
Pulkit Sharma Mohan Ram
22111025, 22111038, 22111022, 22111048, 22111043
{hammadns22, maniksh22, faizalk22, pulkits22,mohanram22}@iitk.ac.in
Indian Institute of Technology Kanpur (IIT Kanpur)

Abstract

Every year, the crop output of India is significantly affected due to delayed detection of diseases in crops. This research is a contribution to the farmers in their battle against coffee plants diseases. It will help in timely detection of diseases, resulting in increased coffee production output of India. Many coffee plant diseases like Leaf Rust, Cercospora Spots have clear visual symptoms and thus can be extracted and their classification can be done. Convolutional Neural Networks (CNNs) has proved its efficiency and accuracy in the field of image classification and pattern recognition. Hence it can act as a powerful tool in the diagnosis of coffee leaves diseases since these symptoms have clearly distinguishable patterns. Thus, a Convolution Neural Network model is proposed which utilizes the technique of Transfer Learning, reducing the training time taken by the model significantly..

1 Introduction

Training of model for disease detection in coffee plant, model such as inception v3 and then converting inception v3 to tensorflow lite and then optimization is applied using quantization techniques such as dynamic quantization, float 16 quantization in tensorflow and then comparing the footprints of all, inception v3, converted model to tensorflow and then optimized models using quantization.

2 Related Work

- An IoT-Based Smart Solution for Leaf Disease Detection The existing farm monitoring method is manual; farmers check parameters like soil moisture, humidity, and leaf disease. In the farm, various sensors are deployed, like soil moisture sensors, Temperature - Humidity sensors, and cameras for detecting diseases on a leaf. Data is collected from sensors and sent to Raspberry PI through wired or wireless devices. On the server side, data is verified and matched with outstanding values of data like temperature value, humidity value, and soil moisture value. If a difference occurs concerning a predefined threshold value, a notification is sent to the farmer on his mobile or website. The output of sensors is generated on the web-page and the farmer. Get detailed information about his farm's crop and atmosphere from anywhere. Crop disease detection is done by using Image Processing. The camera is placed near the crop, taking an image of a leaf. The captured image is sent to the server, and using Image processing techniques, leaf disease is detected; the status of a leaf is sent back to the farmer on the web page mobile phone on the app. [1]

- IoT Enabled Efficient Detection and Classification of Plant Diseases for Agricultural Applications A simple and efficient disease detection and classification system have been proposed in this paper. The images of the plants are captured periodically using camera sensors that are interfaced with the Raspberry Pi3 hardware model. From the acquired images, requisite GLCM features

are extracted and transferred to the cloud through Wi-Fi. For efficient extraction of features, the acquired images must be segmented efficiently, which is carried out using the k-means clustering segmentation technique. At the monitoring site, the features that have been extracted are retrieved by agriculture experts. Here, the classification of diseases is done by Random Forest Classification technique. The experts provide solutions to the farmers in case of any plant disease infections. One of the essential tasks must be carried out in agriculture. Diseases are to be detected at an early stage to avoid the spread of the conditions to other plants. Classification of diseases is equally essential to apply the solutions and treat the diseases as soon as they are detected. The proposed system detects plant diseases earlier utilizing image processing techniques and IoT. On average, the proposed method has achieved 24% improved classification accuracy compared to the existing K-NN classifier. This system enables early detection of diseases in hill bananas and remote monitoring of the hill banana field. [2]

- An IoT based System with Edge Intelligence for Rice Leaf Disease Detection using Machine Learning, the referred data-set was not well sorted. So they re-separated the data-set in different classes. They have used 190 healthy leaf images and 310 infected leaf images for training data-set. For testing data-set they have used 30 healthy and 50 infected leaf images. For reduced computation time, they have resized the images to 256 x 256 pixels. Also changed the color format of images from RGB color format to HSV color format, as it is easier to separate colors in HSV color format. Afterwards they also removed the background from images by making two masks for extracting green and brown colors from the image and merged them. Thet have used six machine learning algorithms to train the model and all these models are validated using 10-Fold Cross-Validation technique. The used algorithms are K Nearest Neighbors, Random Forest, Support Vector Machine, Naive Bayes, Logistic Regression and Decision trees.[3]

- Disease Detection in Coffee Plants Using Convolutional Neural Network In these paper,the dataset used for training the model includes large variety of photos of coffee leaves for disease detection can be found in the pre-processed dataset.Preprocessing was done on the data to get rid of irrelevant information and highlight the symptoms.The images include both healthy and diseased leaves that have been impacted by various biotic stressors. The proposed network classifies data using a sequential model. The Keras library has been used in Python to construct this sequential model. There are about 21 million parameters in the model. The optimization algorithm used to update the parameters is Mini-Batch Gradient Descent (MBGD). The generalizability of the model is improved through the use of data augmentation. The use of picture modifications, contrast adjustments, and image blurring without changing the labels on the original images results in a set of new images that are changed versions of input data. The Inception v3 model is the one that was utilised in this case for transfer learning. With the help of the TensorFlow Hub library, They performed transfer learning on more than a million images from the ImageNet Database that belonged to approximately 1000 different classes. They then added a flatten and dense layer to their network. It modifies the dimensionality of the data and avoids the overfitting issue. The proposed convolution neural network has obtained an overall testing accuracy of 97.61% with a loss value of 0.35 using data-augmentation, transfer learning, and MBGD as an optimizer. The coffee plant farmer could be helped in the early detection of infections with the aid of this technology.[4]
- Plant Disease Detection Using IOT. In this research paper model focuses on using sensor to detect early disease detection sensors like temperature, humidity, and color sensors for collecting data from plant leaves. The research paper discussed about not only using the image processing but also using sensors. The changes that a plant undergoes are captured by the different sensors and then analyzed with Arduino software The data collected from temperature, humidity and color sensors are given to Arduino UNO kit from which the information is communicated to the farmers. The paper discussed different methods to achieve this such as Data acquisition, and sensors use to detect different sensor values such as, Temperature sensors to sense the surrounding temperatures paper suggested the use of DHT11 sensor to sense the temperature on the surface to determine whether the leaf is healthy or not based on the whether the temperature detected using sensor is based on certain values. Humidity sensor (DHT11) to sense humidity of surface of leaf to determine whether it is healthy or not based on humidity values are within certain range

or not Color sensor for image processing. The paper discussed that with the proposed model by using several sensors integrated together to detect disease we can detect the disease much early because it not only relies on the image processing but other sensor values. These sensors values are periodically monitored whether they are within certain range or not if some of values are not within certain range then with combination of image processing further clarification whether disease is there or not, or may be the disease is going to happen. The paper uses 100 samples out which 50 samples are normal and 50 samples are diseased. The images of the leaves collected are presented in the report. Initially the standard values of healthy leaves are stored in the database. Then we took the test samples of leaves to test. But the paper also propose the limitation that is here only disease is present or not is detected but it can be further enhanced to detect what kind of disease is present or not. And also here it is discussed about using few sensor like temperature, humidity, color sensor but with increment of more sensors accuracy of disease detection can be increased. link

- Forecasting disease spread to reduce crop losses. This paper discusses about using satellite imagery, machine learning and forecasting models to reduce crop loss. By applying machine learning model on satellite images here we want to identify patterns in environmental conditions (i.e. temperature, rainfall, humidity etc.) . The paper suggested that data can be collected through crowd sourcing via a user-friendly web-based or a mobile application. These data points can be sent to the tool and then eventually to a data storage using HTTP or REST API. This paper is not relying on sensor data rather it is focusing on satellite data example data from satellites are Sentinel satellites. The paper discusses about list of parameters to predict the spread of diseases like wind speed, wind direction, precipitation, Air temperature, Soil Temperature, Humidity, Solar Radiation, Pressure, Land Use/cover, Surface Runoff. Basically, data is collected from Crowd Sourcing, Agriculture department, Satellite imagery, Third party open-source climate records. For predicting the spread, paper proposed the classification algorithms for making the ML model. Among various algorithm paper suggested to use logistic regression, Decision Tree Random Forest because they do best for classification on continuous and categorical data. The paper also discusses about the limitations such as datasets collection because the availability of historical record for occurrence of the specific disease is still a major challenge, it also a challenge that prediction is based on validated input but there can be error associated which can be analyzed and cancelled only after the real testing in field.[5]
- In these research paper, a remote monitoring system based on the detection of leaf disease was taken into consideration. For measuring moisture, temperature, and humidity, they made use of sensor networks. They placed sensors around the farms, and the Raspberry PI was used to control all these sensors. By integrating a camera and Raspberry PI, leaf disease can be identified. Farmers are immediately informed of the status of a farm, such as a leaf disease or crop-affecting factors like humidity, temperature, and moisture, via a WIFI server and Raspberry PI.[6]
- Deep Neural Networks Based Recognition of Plant Diseases by Leaf Image Classification. In these paper, they aimed to develop a three-level framework for a robotized mechanism that locates the location of disease in plants using IoT. The framework is created using sensors, including those for temperature, moisture, and shading, which are related to plant leaf health. They sent the data to the cloud for analysis using a WiFi shield. This cloud-based data is then compared to the entire dataset to determine whether the leaf is affected, allowing the method to be applied in a various fields of industries. Although determining the quality of a leaf and determining its health might be done with greater efficiency and accuracy using picture handling methods. [7]

3 Proposed Idea

We have trained inceptionV3 model and converted it into TFLite model and the post training quantization is done on TFLite models and compared the footprints of all the models.

Post training Quantization:
Post-training quantization is a conversion technique that can reduce model size while also improving CPU and hardware accelerator latency, with little degradation in model accuracy.

Two types of quantization used are:

1. **Dynamic Range Quantization:**
   Dynamic range quantization is a recommended starting point because it provides reduced memory usage and faster computation without you having to provide a representative dataset for calibration. This type of quantization statically quantizes only the weights from floating point to integer at conversion time, which provides 8-bits of precision.

2. **Float16 Quantization:**
   We can reduce the size of a floating point model by quantizing the weights to float16, the IEEE standard for 16-bit floating point numbers. It reduces model size by half and causes minimal loss in accuracy.

4 Methodology

Dataset Description:
The dataset we used is the dataset which was prepared in [8] using the three datasets of coffee leaves namely RoCoLe, BrACoL and LiCoLe. The link to the dataset – dataset.
The dataset is having three stages. Stage 1 splits the images into two classes 'Healthy' and 'Unhealthy'. Stage 2 splits the images into three classes namely 'Rust', 'Brown Spots' and 'Sooty Molds'. Stage 3 splits the images into four classes namely 'Cecospora', 'Phoma', 'Leaf Miner' and 'Red Spider Mite'
We have used stage 1 dataset because we are making prediction model to run on microcontrollers so it will be efficient to do binary classification i.e. 'Healthy' or 'Unhealthy'.
Description of dataset is given below:

	Healthy (No of images)	Unhealthy (No of images)
Training	3000	3000
Validation	308	325
Testing	155	313

Pre-processing
The pre-processing done is resizing the image to (224, 224) and re-scaling all the pixel values from 0 to 1 by dividing all the pixel values by 255. This is done by adding a pre-processing layer in final model.

ML models used
Pre-trained inceptionV3 model is used as the feature extraction layer in final model which is trained using given dataset.

Details about the final model is given below:

```
Model: "sequential_2"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 sequential (Sequential)     (None, 224, 224, 3)       0

 keras_layer (KerasLayer)    (None, 2048)              21802784

 dense (Dense)               (None, 2)                 4098

=================================================================
Total params: 21,806,882
Trainable params: 4,098
Non-trainable params: 21,802,784
_________________________________________________________________
```

- First layer is the pre-processing layer.
- Second layer is the feature extraction layer taken from the pre-trained inceptionV3 model.
- Third layer is the output layer having two outputs

Optimizer: Adam
Loss: Sparse Categorical Cross Entropy
Epochs: 20
Batch Size: 32
Activation function on final output layer: Softmax

Final trained model is then converted to TFLite model. Two types of optimizations are done on TFLite model namely Dynamic Range Quantization and Float16 Quantization.
Finally, footprints (like Model Size, Accuracy, Precision, Recall, Avg CPU utilization, Avg inference time per data pt.) of different models are compared.

Different models used for comparison:
1. InceptionV3
2. TFLite(No Quantization)
3. TFLite(Dynamic Range Quantization)
4. TFLite(Float16 Quantization)

Implementation:

1. Training and Validation data is loaded using the below code.

```
1
2  dataset_dir = "/content/gdrive/MyDrive/iot/data"
3  train_dir = os.path.join(dataset_dir, 'train')
4  val_dir = os.path.join(dataset_dir, 'validation')
5  train_data = tf.keras.utils.image_dataset_from_directory(train_dir,
6                                       labels = "inferred",
7                                       color_mode='rgb',
8                                       batch_size=BATCH_SIZE,
9                                       image_size=(IMG_SHAPE,
10                                      IMG_SHAPE))
11
12 val_data = tf.keras.utils.image_dataset_from_directory(val_dir,
13                                     labels = "inferred",
14                                     color_mode='rgb',
15                                     batch_size=BATCH_SIZE,
```

```
                                        image_size=(IMG_SHAPE,
                                        IMG_SHAPE))
```

2. Pre-processing layer is created using below code.

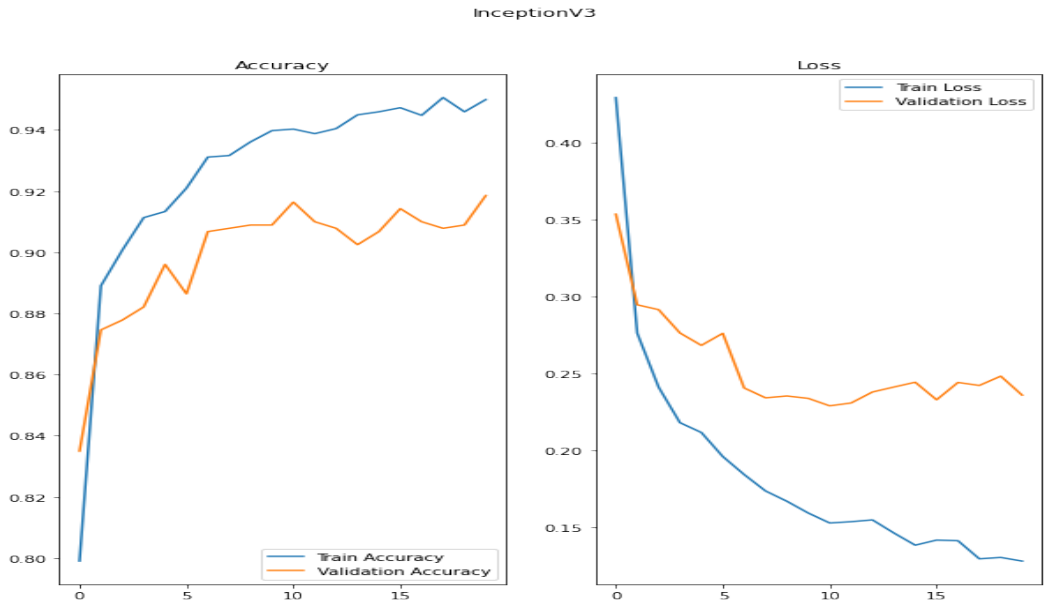
```
resize_and_rescale = tf.keras.Sequential([
  tf.keras.layers.Resizing(IMG_SHAPE, IMG_SHAPE),
  tf.keras.layers.Rescaling(1./255)
])
import tensorflow_hub as hub
inception_v3 = "https://tfhub.dev/google/tf2-preview/inception_v3/
    feature_vector/4"

feature_extractor_model = inception_v3

feature_extractor_layer = hub.KerasLayer(
    feature_extractor_model,
    input_shape=(IMG_SHAPE, IMG_SHAPE, 3),
    trainable=False)
```

3. Final model is created in which the first layer is pre-processing layer, second layer is inceptionV3 feature extraction layer and final output layer is dense layer with two output points and activation function as softmax.

```
model = tf.keras.Sequential([
    resize_and_rescale,
  feature_extractor_layer,
  tf.keras.layers.Dense(num_classes, activation = 'softmax')
])

model.build(input_shape=(None, IMG_SHAPE, IMG_SHAPE, 3))
model.summary()
```

4. Model is compiled and trained using below code and got the following output and convergence curve for loss and accuracy:

```
model.compile(
    optimizer=tf.keras.optimizers.Adam(),
    loss="sparse_categorical_crossentropy",
    metrics=['acc'])

NUM_EPOCHS = 20

history = model.fit(train_data,
                    validation_data=val_data,
                    epochs=NUM_EPOCHS)
```



5. Finally trained model is converted to TFLite model without doing any quantization and saved on the disk.

```
converter = tf.lite.TFLiteConverter.from_saved_model('coffee1')
tflite_model = converter.convert()
with open('model.tflite', 'wb') as f:
  f.write(tflite_model)
tflite_model_file = pathlib.Path('coffee1_noquant.tflite')
tflite_model_file.write_bytes(tflite_model)
```

6. Dynamic range quantization is done and saved the TFLite (Dynamic Range Quantization) model.

```
converter = tf.lite.TFLiteConverter.from_saved_model('coffee1')
converter.optimizations = [tf.lite.Optimize.DEFAULT]
tflite_model = converter.convert()
with open('model.tflite', 'wb') as f:
  f.write(tflite_model)
tflite_model_file = pathlib.Path('coffee1_dyn_quant.tflite')
tflite_model_file.write_bytes(tflite_model)
```

7. Float16 quantization is done and saved the TFLite (Float16 Quantization) model.

```
converter = tf.lite.TFLiteConverter.from_saved_model('coffee1')
converter.optimizations = [tf.lite.Optimize.DEFAULT]
converter.target_spec.supported_types = [tf.float16]
tflite_model = converter.convert()
with open('model.tflite', 'wb') as f:
  f.write(tflite_model)
tflite_model_file = pathlib.Path('coffee1_f16_quant.tflite')
tflite_model_file.write_bytes(tflite_model)
```

8. The below function is used to compute the total prediction time of all test data points and average prediction time of a single test data point using inceptionV3 model.

```
def compute_pred_time(model, classes):
    parent_dir = "/content/gdrive/MyDrive/iot/data/test"
    total_time = 0
    data_points = 0
    for cl in classes:
        class_dir = os.path.join(parent_dir, cl)
        pbar = ProgressBar()
        for file in pbar(os.listdir(class_dir)):
```

```
            image_path = os.path.join(class_dir, file)
            img = image.load_img(image_path, target_size=(224, 224))
            img_array = image.img_to_array(img)
            img_batch = np.expand_dims(img_array, axis=0)
            start = perf_counter()
            prediction = model.predict(img_batch, verbose=0)
            end = perf_counter()
            total_time += end-start
            data_points += 1

    return total_time, total_time/data_points
```

9. The below function is used to compute accuracy, average CPU utilization, precision and recall using inceptionV3 mode.

```
def compute_other(model, classes):
    parent_dir = "/content/gdrive/MyDrive/iot/data/test"
    tp, fp, tn, fn = 0, 0, 0, 0
    total_cpu_util = 0
    accuracy = []
    my_process = psutil.Process(os.getpid())
    initial = my_process.memory_percent()

    for cl in classes:
        class_dir = os.path.join(parent_dir, cl)
        pbar = ProgressBar()
        for file in pbar(os.listdir(class_dir)):
            image_path = os.path.join(class_dir, file)
            img = image.load_img(image_path, target_size=(224, 224))
            img_array = image.img_to_array(img)
            img_batch = np.expand_dims(img_array, axis=0)
            # total_cpu_util += (my_process.cpu_percent(1)/psutil.
    cpu_count())
            prediction = model.predict(img_batch, verbose=0)
            total_cpu_util += (my_process.cpu_percent()/psutil.cpu_count()
    )
            predicted_label = np.argmax(prediction[0])

            if cl ==  '0_Healthy':
                if predicted_label == 0:
                    tp += 1
                else:
                    fp += 1

            else:
                if predicted_label == 0:
                    fn += 1
                else:
                    tn += 1

    final = my_process.memory_percent()

    avg_mem_perc = final - initial
    avg_acc = (tp + tn)/(tp + tn + fp + fn)
    avg_cpu_util = total_cpu_util/(tp + tn + fp + fn)
    precision = tp/(tp + fp)
    recall = tp/(tp + fn)

    return avg_acc, avg_cpu_util, precision, recall
```

10. The below function is used to compute the total prediction time of all test data points and average prediction time of a single test data point using TFLite models.

```
def compute_pred_time_tflite(interpreter, classes):
    total_time = 0
    data_points = 0
    interpreter.allocate_tensors()
    input_index = interpreter.get_input_details()[0]["index"]
    output_index = interpreter.get_output_details()[0]["index"]
    input_format = interpreter.get_output_details()[0]['dtype']

    parent_dir = "/content/gdrive/MyDrive/iot/data/test"


    for cl in classes:
        class_dir = os.path.join(parent_dir, cl)
        pbar = ProgressBar()
        for file in pbar(os.listdir(class_dir)):
            image_path = os.path.join(class_dir, file)
            img = cv.imread(image_path)
            img = cv.resize(img,(224, 224))
            input_tensor= np.expand_dims(img, axis=0).astype(input_format)
            interpreter.set_tensor(input_index, input_tensor)

            start = perf_counter()
            interpreter.invoke()
            end = perf_counter()

            total_time += end-start
            data_points += 1

    return total_time, total_time/data_points
```

11. The below function is used to compute accuracy, average CPU utilization, precision and recall using TFLite models.

```
def compute_other_tflite(interpreter, classes):
    total_cpu_util = 0
    accuracy = []
    tp, fp, tn, fn = 0, 0, 0, 0
    my_process = psutil.Process(os.getpid())
    interpreter.allocate_tensors()
    input_index = interpreter.get_input_details()[0]["index"]
    output_index = interpreter.get_output_details()[0]["index"]
    input_format = interpreter.get_output_details()[0]['dtype']


    parent_dir = "/content/gdrive/MyDrive/iot/data/test"


    for cl in classes:
        class_dir = os.path.join(parent_dir, cl)
        pbar = ProgressBar()
        for file in pbar(os.listdir(class_dir)):
            image_path = os.path.join(class_dir, file)
            img = cv.imread(image_path)
            img = cv.resize(img,(224, 224))
            img = cv.resize(img,(224, 224))
            input_tensor= np.expand_dims(img, axis=0).astype(input_format)
            interpreter.set_tensor(input_index, input_tensor)
            # total_cpu_util += (my_process.cpu_percent(1)/psutil.
    cpu_count())
            interpreter.invoke()
            total_cpu_util += (my_process.cpu_percent()/psutil.cpu_count()
    )
```

```
            output = interpreter.tensor(output_index)
            predicted_label = np.argmax(output()[0])
            # prediction.append(predicted_label)

            if cl ==  '0_Healthy':
                if predicted_label == 0:
                    tp += 1
                else:
                    fp += 1

            else:
                if predicted_label == 0:
                    fn += 1
                else:
                    tn += 1

    avg_acc = (tp + tn)/(tp + tn + fp + fn)
    avg_cpu_util = total_cpu_util/(tp + tn + fp + fn)
    precision = tp/(tp + fp)
    recall = tp/(tp + fn)

    return avg_acc, avg_cpu_util, precision, recall
```

5 Results and Discussion

Model	Model Size (MB)	Avg CPU util (%)	Avg Inference time per data pt(ms)
Inception V3	86.4	68.841026	143.008881
TFLite (No Quantization)	83.1	49.055342	174.036952
TFLite (Dynamic Quantization)	21.1	48.831731	134.416044
TFLite (Float16 Quantization)	41.6	48.986859	173.961960

Model	Accuracy (%)	Precision (%)	Recall (%)
Inception V3	91.880342	85.161290	89.795918
TFLite (No Quantization)	87.606838	90.967742	76.216216
TFLite (Dynamic Quantization)	87.179487	90.322581	75.675676
TFLite (Float16 Quantization)	87.606838	90.967742	76.216216

As from the table we concluded that in InceptionV3 model size is highest with approximately 86.4 MB but when converting it no TFlite the model size is reduced but accuracy is also decreased by 4% while precision is increased and averaged cpu utilization is decreased by 30% this shows that in regard with CPU utilization TFlite is performing well.
And on applying further optimization using quantization techniques such as Dynamic quantization the model size is reduced drastically by 76% and accuracy is also decreased but accuracy is reduced by only 5% approximately relative to InceptionV3 and also with this quantization precision is increased and also average CPU utilization has also decreased by approximately 30%.
With Float16 Quantization also model size has decreased by 53 approximately with slight decrease in Accuracy of 4% approx with increase in precision by 5% and here also CPU utilization has been decreased by nearly 30%.

6 Future Work

- In this project further improvement of accuracy can made by using some more sensor data like humidity sensor, temperature sensor to detect humidity and temperature of plant leaves so that using this data more features we can get and more accurately disease can be predicted.

- lso, if satellite dataset and dataset from agriculture department is integrated then forecasting of disease can also be predicted.
- Also, with integration of more sensors like thermal image sensor for thermal imaging of leaves we can even have future prediction of disease in plant i.e., what are the chances of having disease in future and in what location.

7 Conclusion

After training model of inceptionV3 and then converting inception V3 to TensorflowLite along with different quantization techniques we finally concluded that TensorflowLite with dynamic quantization has performed well as compared to original InceptionV3 trained model with, because TFlite(Dynamic Quantization) is performing well with respect to model size(decreased by 76%) and CPU utilization(decreased by 30%) and Average inference time per data pt(ms) has also been decreased by 5%, but there is some disadvantage that accuracy has been decreased but that accuracy is not decreased by much percentage(5%), while CPU utilization and model size has been optimized very well.

8 Individual Contributions

Name	Contribution
Faizal Khan	20%
Hammad Ansari	20%
Manik Shekhar	20%
Mohan Ram	20%
Pulkit Sharma	20%

References

[1] A. Thorat, S. Kumari, and N. D. Valakunde, "An iot based smart solution for leaf disease detection," in *2017 International Conference on Big Data, IoT and Data Science (BID)*, pp. 193–198, 2017.

[2] R. D. Devi, S. A. Nandhini, R. Hemalatha, and S. Radha, "Iot enabled efficient detection and classification of plant diseases for agricultural applications," in *2019 International Conference on Wireless Communications Signal Processing and Networking (WiSPNET)*, pp. 447–451, 2019.

[3] S. M. Shahidur Harun Rumy, M. I. Arefin Hossain, F. Jahan, and T. Tanvin, "An iot based system with edge intelligence for rice leaf disease detection using machine learning," in *2021 IEEE International IOT, Electronics and Mechatronics Conference (IEMTRONICS)*, pp. 1–6, April 2021.

[4] M. Kumar, P. Gupta, P. Madhav, and Sachin, "Disease detection in coffee plants using convolutional neural network," in *2020 5th International Conference on Communication and Electronics Systems (ICCES)*, pp. 755–760, 2020.

[5] A. Patil, S. A. Sivakumar, and R. Witt, "Forecasting disease spread to reduce crop losses," 2018.

[6] M. A. Nawaz, T. khan, R. M. Rasool, M. Kausar, A. Usman, T. F. N. Bukht, R. Ahmad, and J. Ahmad, "Plant disease detection using internet of thing (iot)," *International Journal of Advanced Computer Science and Applications*, vol. 11, no. 1, 2020.

[7] S. Sladojevic, M. Arsenovic, A. Anderla, and D. Stefanović, "Deep neural networks based recognition of plant diseases by leaf image classification," *Computational Intelligence and Neuroscience*, vol. 2016, pp. 1–11, 06 2016.

[8] F. J. Montalbo, "Automated diagnosis of diverse coffee leaf images through a stage-wise aggregated triple deep convolutional neural network," *Machine Vision and Applications*, vol. 33, 01 2022.