# Capstone Project Machine Learning Project Documentation

**Project Title: Plant Disease Detection Using Deep Learning**

**Team Members**

- Dawit Tadesse Hailu
- Chol Moses Malueth
- Fahadi Mugigayi

## Model Refinement

### 1. Overview

Our model refinement phase is crucial for enhancing the performance of our initial machine learning model. This phase involved iterative improvements through various techniques such as hyperparameter tuning, algorithm adjustments, and feature selection. Our goal was to achieve optimal performance and generalization ability of the model on unseen data.

### 2. Model Evaluation

**Initial Results**:

- Accuracy and loss was the main metric used.The initial model achieved an accuracy of around 78% on the test ( was named validation in our case) set.
- Key metrics such as precision, recall, and F1-score indicated good performance but highlighted some areas for improvement, particularly in model architecture where there were not enough network layers and number of internal nodes.
- Model checkpointing was used to capture and retain the best value at every epoch

```
# Define the ModelCheckpoint callback
checkpoint = ModelCheckpoint(filepath='model_checkpoint.h5',
                             save_best_only=True,
                             monitor='val_loss',
                             mode='min')
```

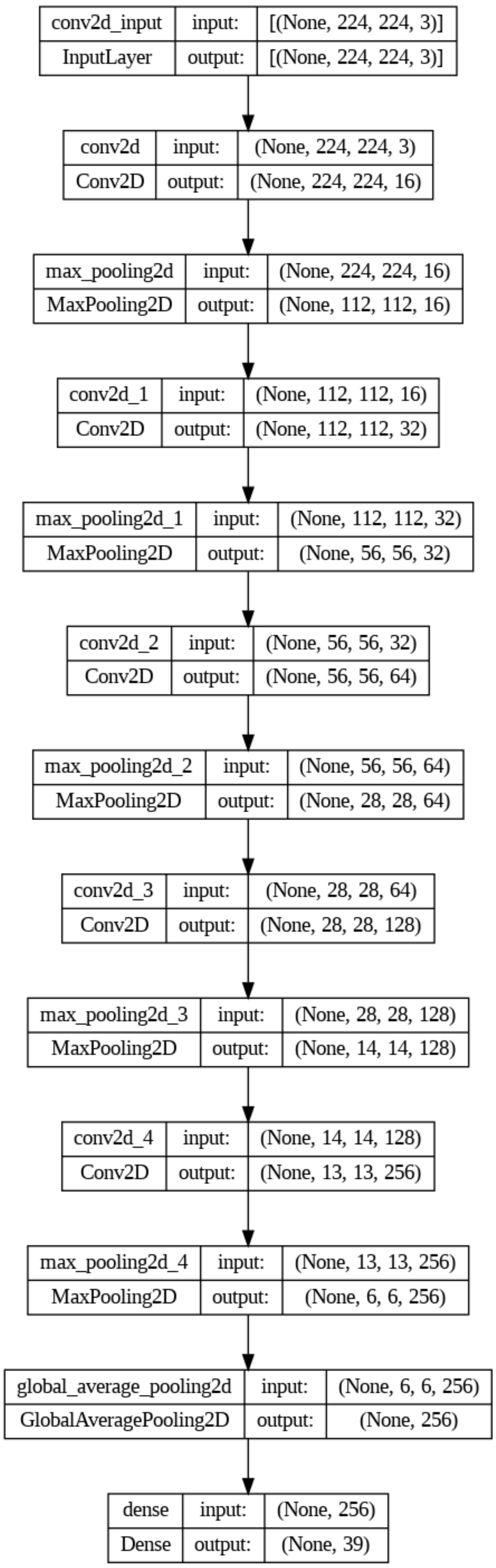
**Visualizations**:

- Confusion matrix showed misclassifications in some classes.

### 3. Refinement Techniques

**Techniques Used**:

- **Hyperparameter Tuning**: Adjusted learning rates, batch sizes, and epoch number.
- **Algorithm Variations**: Explored different CNN architectures starting with smaller architectures with up to 16 convolutional filters and then scaled up to compared performance. The final model which achieved the best result includes 5 convolutional layers of 16, 32, 64, 128 and 256 filters respectively before being connected to a dense layer.


conv2d_input | input: | [(None, 224, 224, 3)]
InputLayer | output: | [(None, 224, 224, 3)]
conv2d | input: | (None, 224, 224, 3)
Conv2D | output: | (None, 224, 224, 16)
max_pooling2d | input: | (None, 224, 224, 16)
MaxPooling2D | output: | (None, 112, 112, 16)
conv2d_1 | input: | (None, 112, 112, 16)
Conv2D | output: | (None, 112, 112, 32)
max_pooling2d_1 | input: | (None, 112, 112, 32)
MaxPooling2D | output: | (None, 56, 56, 32)
conv2d_2 | input: | (None, 56, 56, 32)
Conv2D | output: | (None, 56, 56, 64)
max_pooling2d_2 | input: | (None, 56, 56, 64)
MaxPooling2D | output: | (None, 28, 28, 64)
conv2d_3 | input: | (None, 28, 28, 64)
Conv2D | output: | (None, 28, 28, 128)
max_pooling2d_3 | input: | (None, 28, 28, 128)
MaxPooling2D | output: | (None, 14, 14, 128)
conv2d_4 | input: | (None, 14, 14, 128)
Conv2D | output: | (None, 13, 13, 256)
max_pooling2d_4 | input: | (None, 13, 13, 256)
MaxPooling2D | output: | (None, 6, 6, 256)
global_average_pooling2d | input: | (None, 6, 6, 256)
GlobalAveragePooling2D | output: | (None, 256)
dense | input: | (None, 256)
Dense | output: | (None, 39)

**4. Hyperparameter Tuning**

**Process**:

- Conducted multiple training runs with smaller epochs to find the optimal combination of hyperparameters.
- Key insights included the importance of a smaller learning rate and using Adam optimizer instead of RMS prop which resulted in a lower loss and higher accuracy.

```
model.compile(loss='categorical_crossentropy',

          optimizer='adam',

          metrics=['accuracy'])
```

**Impact**:

- Improved accuracy by 12%.
- Enhanced precision and recall metrics, particularly for classes with less data.

**5. Cross Validation**

For the following project, cross validation **was not used** for the image dataset. Instead for efficient loading of the images, a data generator from keras was employed. Additionally, during the initial dataset train test split, randomization and sampling were ensured.

```
#list out all image full names into a list

    category_image_names = os.listdir(SOURCE + '/' + category)

    # get the length to be put in training

    training_length = int(len(category_image_names) * SPLIT_SIZE)

    # get the length to be put in testing

    testing_length = int(len(category_image_names) - training_length)

    # shuffle the file name list for randomness

    shuffled_set = random.sample(category_image_names, len(category_image_names))

    # split the shuffled list into trainng and testing names list

    training_set = shuffled_set[0:training_length]

    testing_set = shuffled_set[training_length:]
```

**6. Feature Selection**

**Methods Employed**:

- **Quality Check**: Retained only the most impactful images using laplacian operator and blur detection using cv to ensure data integrity. Ensured that all images were clear and suitable for analysis by removing any low-quality or blurry images. ( A value of 25 was used for Laplacian )
- The total image pixel selected was 240 by 240 which gave the highest accuracy rate

```
# Quality Check
def remove_low_quality_images(image_dir):
    for root, dirs, files in os.walk(image_dir):
        for file in files:
            try:
                img = cv2.imread(os.path.join(root, file))
                if img is not None and (cv2.Laplacian(img,
cv2.CV_64F).var() < 25):
                    os.remove(os.path.join(root, file))
            except Exception as e:
                print(f"Error processing {file}: {e}")
                os.remove(os.path.join(root, file))

remove_low_quality_images('./Dataset/all_data')
```

**Effect**:

- Overall Increased dataset quality.
- Slight improvement in model performance and faster training times.

## Test Submission

### 1. Overview

The test submission phase involves preparing the refined model for final evaluation on a separate test dataset. This phase ensures the model's robustness and readiness for real-world deployment.

### 2. Data Preparation for Testing

**Preparation Steps**:

- Ensured that the test dataset underwent the same preprocessing steps as the training data (resizing, normalization….)

```
VALIDATION_DIR = testing_data_dir

validation_datagen = ImageDataGenerator(rescale=1./255,
                                        rotation_range=40,
                                                  width_shift_range=0.2,
                                                  height_shift_range=0.2,
                                                  shear_range=0.2,
                                                  zoom_range=0.2,
                                                  horizontal_flip=True,
                                                  fill_mode='nearest')

validation_generator =
validation_datagen.flow_from_directory(VALIDATION_DIR,

batch_size=100,

class_mode='categorical',

target_size=(240, 240))
```

**3. Model Application**

**Application Process**:

- Applied the trained and refined model to the test dataset to generate predictions on the generator while in training.
- Recorded the results for comparison with training and validation performance within the history parameter later used in visualization.

```
history = model.fit(train_generator,
                    epochs=12,
                    verbose=1,
                    validation_data=validation_generator,
                    callbacks=[checkpoint])
```

**4. Test Metrics**

**Metrics Used**:

- Manual Accuracy, precision, recall, F1-score on the test dataset after training.
- Compared these metrics with those from the training and validation phases to assess model performance consistency.

```
from sklearn.metrics import confusion_matrix, precision_score,
recall_score, accuracy_score

# Predict the labels
y_pred = model.predict(validation_generator)
y_pred_classes = np.argmax(y_pred, axis=1)
```

```
# Get the true labels
y_true = validation_generator.classes

# Calculate the confusion matrix
conf_matrix = confusion_matrix(y_true, y_pred_classes)
print("Confusion Matrix:")
print(conf_matrix)

# Calculate precision
precision = precision_score(y_true, y_pred_classes,
average='weighted')
print(f"Precision: {precision}")

# Calculate recall
recall = recall_score(y_true, y_pred_classes, average='weighted')
print(f"Recall: {recall}")

# Calculate accuracy
accuracy = accuracy_score(y_true, y_pred_classes)
print(f"Accuracy: {accuracy}")
```

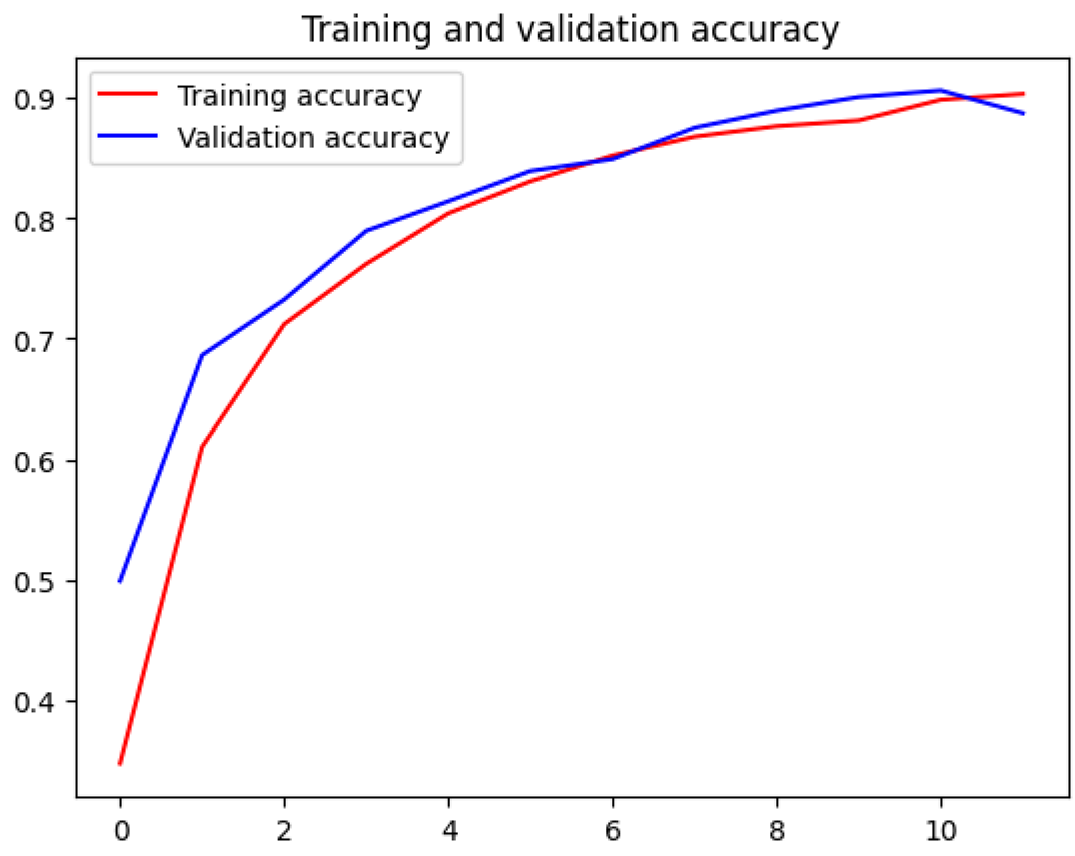
**Results**:

- Accuracy on test data: 90% using validation generator ( used as test set )
- Results indicated strong generalization capability in the latest training run before diminishing returns after the 12th epoch.

```
acc = history.history['accuracy']
val_acc = history.history['val_accuracy']
loss = history.history['loss']
val_loss = history.history['val_loss']
epochs = range(len(acc))

plt.plot(epochs, acc, 'r', label='Training accuracy')
plt.plot(epochs, val_acc, 'b', label='Validation accuracy')
plt.title('Training and validation accuracy')
plt.legend()
plt.show()

plt.plot(epochs, loss, 'r', label='Training loss')
plt.plot(epochs, val_loss, 'b', label='Validation loss')
plt.title('Training and validation loss')
plt.legend()
plt.show()
```



**5. Model Deployment**

**Deployment Steps**:

- Planned the integration of the model into a web server application to allow real-time disease detection with the preparation of a frontend interface
- Ensured the application design was user-friendly

```
# Future model deployment code snippet (using Flask for simplicity)
from flask import Flask, request, jsonify
from tensorflow.keras.models import load_model

app = Flask(__name__)
model = load_model('path/to/saved_model.h5')

@app.route('/predict', methods=['POST'])
def predict():
    image = request.files['image']
    # Preprocess image...
    prediction = model.predict(preprocessed_image)
    return jsonify({'prediction': prediction.tolist()})

if __name__ == '__main__':
    app.run(debug=True)
```

**6. Code Implementation**

**Final refined model code Snippets**:

```
model = Sequential()
model.add(Conv2D(filters=16, kernel_size=2, padding='same',
activation='relu',
                        input_shape=(224, 224, 3)))
model.add(MaxPooling2D(pool_size=2))

model.add(Conv2D(filters=32, kernel_size=2, padding='same',
activation='relu'))
model.add(MaxPooling2D(pool_size=2))

model.add(Conv2D(filters=64, kernel_size=2, padding='same',
activation='relu'))
model.add(MaxPooling2D(pool_size=2))

model.add(Conv2D(filters=128, kernel_size=2, padding='same',
activation='relu'))
model.add(MaxPooling2D(pool_size=2))

model.add(Conv2D(filters=256, kernel_size=2,
kernel_initializer='he_normal', activation='relu'))
model.add(MaxPooling2D(pool_size=2))

# tf.keras.layers.Flatten(),
#     tf.keras.layers.Dense(512, activation='relu'),
#     tf.keras.layers.Dense(1, activation='sigmoid')

model.add(GlobalAveragePooling2D())

model.add(Dense(39, activation='softmax'))

model.summary()
```

## Conclusion

The model refinement and test submission phases were crucial in enhancing the performance and reliability of the plant disease detection system. Through iterative improvements, hyperparameter tuning, and robust testing, the model achieved an accuracy of 90% on the test dataset, demonstrating strong generalization capabilities. Challenges such as data imbalance and model complexity were addressed, resulting in a streamlined and effective model ready for real-world deployment.

## References

- Paymode, A. S., & Malode, V. B. (2022). Transfer Learning for Multi-Crop Leaf Disease Image Classification using Convolutional Neural Network VGG. Artificial Intelligence in Agriculture, 6, 23-33.
- Mohanty, S. P., Hughes, D. P., & Salathé, M. (2016). Using Deep Learning for Image-Based Plant Disease Detection. Frontiers in Plant Science, 7, 1419.
- Falaschetti, L., Manoni, L., Di Leo, D., Pau, D., & Tomaselli, V. (2022). A CNN-based image detector for plant leaf diseases classification. HardwareX, 12, e00363.