Automated Crop Disease Classification Using Machine Learning and Computer Vision: A Deep Learning Approach

Group 5
Robin Gould and Madison Duranleau

【Introduction and Background】

Crop diseases pose a significant threat to global food security and agricultural sustainability. The timely detection and management of these diseases are essential for minimizing yield losses, ensuring food safety, and maintaining economic viability for farmers. Traditionally, disease identification has relied heavily on manual inspection by agronomists, which can be time-consuming, subjective, and prone to errors. Moreover, with the increasing scale of agricultural operations and the globalization of food trade, the need for scalable and efficient disease detection solutions has become more pressing than ever.

In recent years, advancements in machine learning and computer vision have opened up new avenues for automating the detection and classification of crop diseases. By leveraging large-scale datasets of labeled crop images and powerful deep learning algorithms, researchers and practitioners can develop intelligent systems capable of accurately identifying diseases in crops with high throughput and accuracy. These automated systems have the potential to revolutionize agriculture by providing farmers with real-time insights into the health status of their crops, enabling timely interventions and resource allocation.

【Project Objectives】

The primary objective of our project is to develop a machine-learning-based approach for automating crop disease classification. Specifically, we aim to build a deep learning model capable of accurately distinguishing between healthy and diseased crop leaves and identifying the specific disease types. By leveraging state-of-the-art convolutional neural networks (CNNs) and large-scale image datasets, we seek to achieve high levels of accuracy and generalization performance across a wide range of crop species and disease types.

【Methodology】

Our methodology involves several key steps:

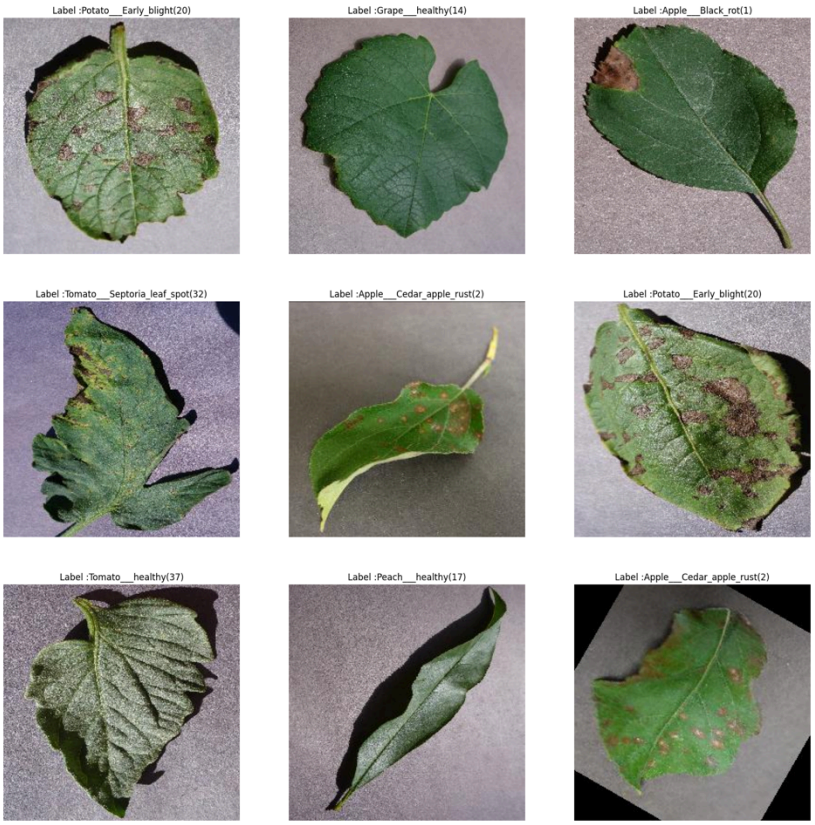
1. Data Acquisition: We obtained a comprehensive dataset of crop images from Kaggle, which includes RGB images of healthy and diseased crop leaves across multiple crop species and disease types.

2. Data Preparation: The dataset was preprocessed and divided into training and validation sets with an 80/20 ratio. Additionally, pixel values of images were normalized to the range [0, 1] to facilitate model training. Data augmentation techniques such as rotation, flipping, and scaling were not explicitly mentioned but could be considered to further enhance the model's ability to generalize.

3. Data Visualization: Prior to model training, we visualized a subset of images from the training dataset to gain insights into the diversity and characteristics

of the data. Visual inspection of the images can provide valuable intuition regarding the challenges associated with crop disease classification.

Below is our subset of images from the training dataset:

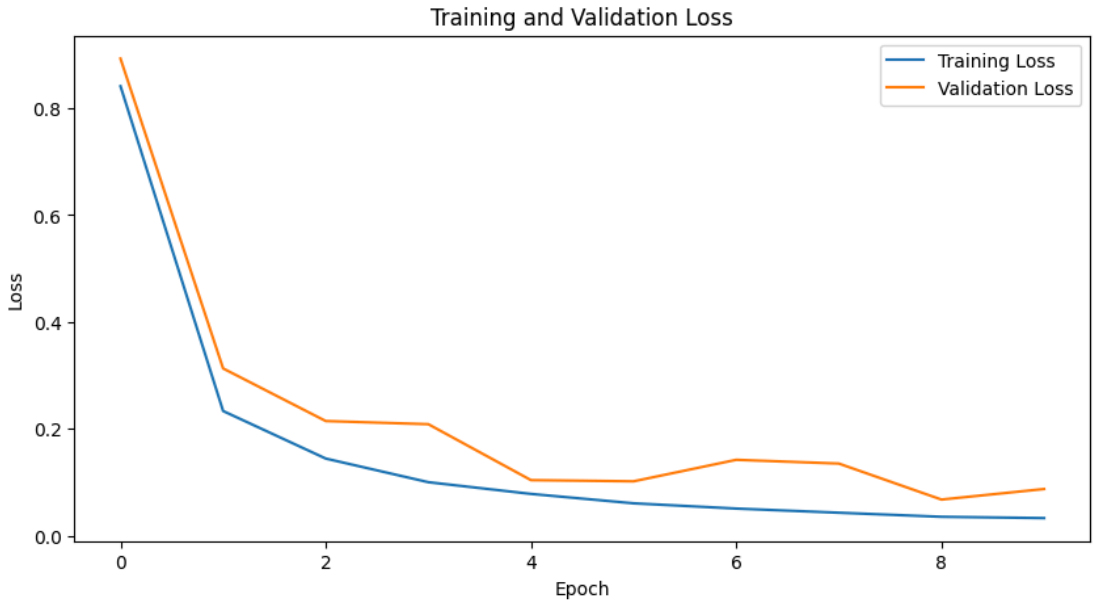


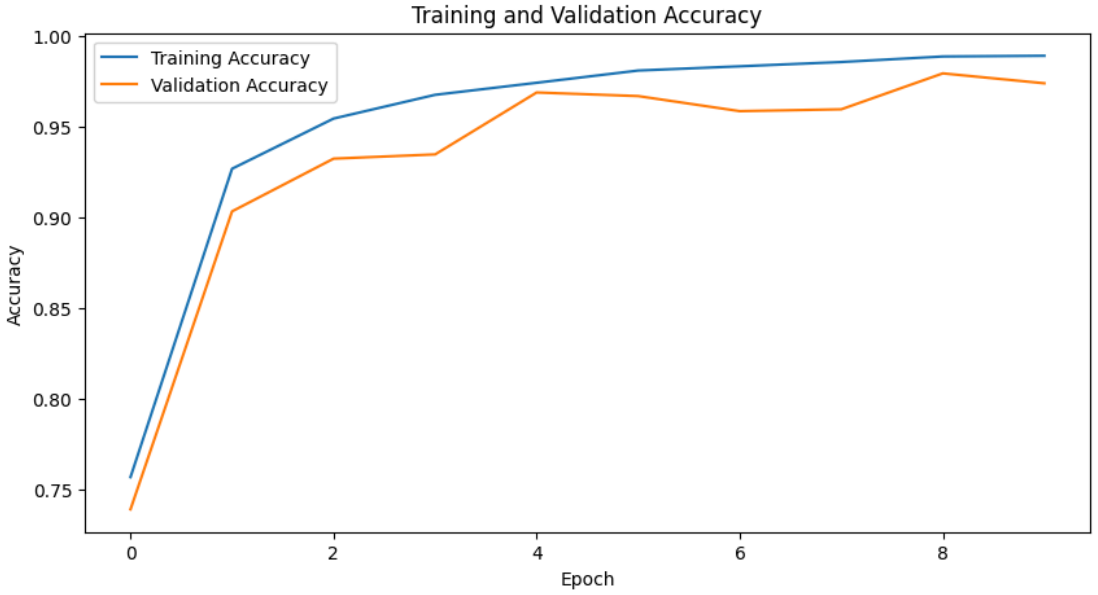
4. Model Development: We developed a deep learning model based on convolutional neural networks (CNNs) to perform crop disease classification. The model architecture consists of multiple convolutional and pooling layers followed by fully connected layers, with batch normalization and dropout regularization to improve performance and prevent overfitting.

5. Model Training and Evaluation: We trained the model using the training dataset and evaluated its performance on the validation set. We monitored key performance metrics such as accuracy, precision, and recall to assess the model's effectiveness in detecting and classifying crop diseases.

【Results】

After training our crop disease classification model for 10 epochs, we evaluated its performance on the validation set. The model achieved an accuracy of 97.42% on the validation set, indicating its ability to generalize well to unseen data. Due to the high accuracy, we did not believe further fine-tuning and optimization would be necessary to improve its performance.





【Future Directions】

While our initial results are promising, there are several avenues for further exploration and improvement. One potential direction is the incorporation of transfer learning techniques, where pre-trained CNN models such as VGG, ResNet, or MobileNet are fine-tuned on our dataset to leverage their learned features. Additionally, ensembling multiple models or employing more advanced architectures such as attention mechanisms could potentially enhance classification performance.

【Conclusion】

In conclusion, our project demonstrates the potential of machine learning and computer vision techniques for automating crop disease classification. By

leveraging deep learning models and large-scale datasets, we can empower farmers with tools to detect and mitigate crop diseases more effectively, ultimately contributing to increased agricultural productivity and food security.

【Limitations and Challenges】

It's important to acknowledge the limitations and challenges encountered during the project. These may include issues related to dataset quality, class imbalance, and computational resources. Addressing these challenges through data augmentation, class balancing techniques, and optimization algorithms can lead to more robust and reliable models.

【Ethical Considerations】

As with any machine learning application, ethical considerations must be taken into account. It's crucial to ensure that the deployment of automated crop disease classification systems does not exacerbate existing inequalities or contribute to environmental harm. Transparency, accountability, and fairness should be prioritized throughout the development and deployment process.

【Recommendations for Further Research】

To build upon our work, future research could explore the integration of remote sensing data and devices for real-time monitoring of crop health. Additionally, collaboration with domain experts such as agronomists and plant pathologists can provide valuable insights into the underlying biological mechanisms of crop diseases, leading to more accurate and interpretable models.

【Reference】

- https://www.kaggle.com/datasets/vipoooool/new-plant-diseases-dataset/data

【Code】

Math 5671 - Group 5 - Final Project

We downloaded our data from: https://www.kaggle.com/datasets/vipoooool/new-plant-diseases-dataset/data

```
# Mount Google Drive to access files
from google.colab import drive
drive.mount('/content/gdrive')

# Unzip the files
!unzip '/content/gdrive/My Drive/models/math5671/archive(1).zip'

import os
import numpy as np
import pandas as pd
import torch
import matplotlib.pyplot as plt
import torch.nn as nn
from torch.utils.data import DataLoader
from PIL import Image
import torch.nn.functional as F
import torchvision.transforms as transforms
from torchvision.utils import make_grid
from torchvision.datasets import ImageFolder
import keras
from keras import layers
from keras.layers import BatchNormalization, Dropout
from tensorflow.keras.utils import image_dataset_from_directory

# This is where we are defining the directories
root_dir = "New Plant Diseases Dataset(Augmented)/New Plant Diseases Dataset(Augmented)"
train_dir = root_dir + "/train"
val_dir = root_dir + "/valid"
test_dir = "test/test"
# Get the list of classes
classes = os.listdir(train_dir)

# Printing the different classes of our dataset
print(classes)

# Printing the number of classes we have
print(len(classes))

# Printing the number of images per class in the training directory
for i in classes:
  print(f"{i}: {len(train_dir + '/' + i)}")
```

```
# Counting the total number of classes in training directory
total_class_train = len(os.listdir(train_dir))
# Counting the total number of classes in validation directory
total_class_val = len(os.listdir(val_dir))
# Counting the total number of classes in test directory
total_test = len(os.listdir(test_dir))

# Printing the total number of classes in training directory
print(total_class_train)
# Printing the total number of classes in the validation directory
print(total_class_val)
# Printing the total number of classes in the test directory
print(total_test)

# Function to count data samples per class
def count_data_per_class(path):
    classes = sorted(os.listdir(path))
    class_counts = {}
    for cls in classes:
        class_path = os.path.join(path, cls)
        class_counts[cls] = len(os.listdir(class_path))
    return class_counts

# Calculate total data of each class
train_class_counts = count_data_per_class(train_dir)
val_class_counts = count_data_per_class(val_dir)

# Create dataframe class count
train_counts_df = pd.DataFrame.from_dict(train_class_counts, orient="index",
columns=["Count"])
val_counts_df = pd.DataFrame.from_dict(val_class_counts, orient="index",
columns=["Count"])

# Displaing the train counts
train_counts_df

# Display the validation counts
val_counts_df

# Load and transform train and validation datasets
train = ImageFolder(train_dir, transform=transforms.ToTensor())
valid = ImageFolder(val_dir, transform=transforms.ToTensor())

# Display sample images from the train dataset
plt.figure(figsize=(20, 20))
for i, (image, label) in enumerate(torch.utils.data.Subset(train,
np.random.choice(len(train), 9, replace=False))):
    ax = plt.subplot(3, 3, i+1)
    plt.title("Label :" + train.classes[label] + "(" + str(label) + ")")
    plt.imshow(image.permute(1, 2, 0))
```

```
    plt.axis("off")

# Create image datasets from directories
train_gen = image_dataset_from_directory(directory=train_dir, image_size=(256, 256))
valid_gen = image_dataset_from_directory(directory=val_dir, image_size=(256, 256))

# Normalize the pixel values of images in the training and validation dataset
train_gen = train_gen.map(lambda image, label: (image / 255.0, label))
valid_gen = valid_gen.map(lambda image, label: (image / 255.0, label))

# Initialize the model
model = keras.Sequential()

# Add Convolution and Pooling Layers to define the model architecture
model.add(keras.layers.Conv2D(32,(3,3),activation="relu",padding="same",input_shape=(256,256,3)))
model.add(keras.layers.Conv2D(32,(3,3),activation="relu",padding="same"))
model.add(keras.layers.MaxPooling2D(3,3))
model.add(BatchNormalization())

model.add(keras.layers.Conv2D(64,(3,3),activation="relu",padding="same"))
model.add(keras.layers.Conv2D(64,(3,3),activation="relu",padding="same"))
model.add(keras.layers.MaxPooling2D(3,3))

model.add(keras.layers.Conv2D(128,(3,3),activation="relu",padding="same"))
model.add(keras.layers.Conv2D(128,(3,3),activation="relu",padding="same"))
model.add(keras.layers.MaxPooling2D(3,3))
model.add(BatchNormalization())

model.add(keras.layers.Conv2D(256,(3,3),activation="relu",padding="same"))
model.add(keras.layers.Conv2D(256,(3,3),activation="relu",padding="same"))

model.add(keras.layers.Conv2D(512,(5,5),activation="relu",padding="same"))
model.add(keras.layers.Conv2D(512,(5,5),activation="relu",padding="same"))

model.add(keras.layers.Flatten())

model.add(keras.layers.Dense(1568,activation="relu"))
model.add(keras.layers.Dropout(0.5))
model.add(BatchNormalization())
model.add(keras.layers.Dense(38,activation="softmax"))

# Compile the model
opt = keras.optimizers.Adam(learning_rate=0.0001)
model.compile(optimizer=opt,loss="sparse_categorical_crossentropy",metrics=['accuracy'])
model.summary()
```

```
# Train the model
history = model.fit(
    train_gen,
    validation_data=valid_gen,
    epochs=10
)

# Model evaluation
# Evaluate the model on the validation set
model.evaluate(valid_gen)

# Plot training loss
plt.figure(figsize=(10, 5))
plt.plot(history.history['loss'], label='Training Loss')
plt.plot(history.history['val_loss'], label='Validation Loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.title('Training and Validation Loss')
plt.legend()
plt.show()

# Plot the training accuracy
plt.figure(figsize=(10, 5))
plt.plot(history.history['accuracy'], label='Training Accuracy')
plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.title('Training and Validation Accuracy')
plt.legend()
plt.show()
```