AI for AgriTech Hackathon Phase 1: Potato Disease Classification

Nikhil Teja Cheema (MTech IIT Hyderabad)
N S Deepika (PhD IIT Hyderabad)

1 Introduction

This project develops a Convolutional Neural Network (CNN) to classify potato leaf images into three categories: Potato Early Blight, Potato Late Blight, and Potato Healthy. The model is implemented using TensorFlow/Keras and processes a dataset stored on Google Drive, organized into Train, Test, and Valid splits. The goal is to accurately identify potato diseases to support agricultural decision-making.

1.1 Dataset Description

The dataset[1] comprises 1500 images of potato leaves, equally distributed across three classes (500 images each for Early Blight, Late Blight, and Healthy). Images are stored in a Google Drive directory (`/content/drive/MyDrive/Potato`) with subfolders for Train, Test, and Valid splits. Each image is verified for integrity, and the dataset is balanced to ensure equitable class representation.

(a) Potato Early Blight (b) Potato Late Blight (c) Potato Healthy

Figure 1: Sample images from the dataset, representing each class: (a) Potato Early Blight, (b) Potato Late Blight, and (c) Potato Healthy.

2 Model Code and Working

The Python code below outlines the data loading, preprocessing, model architecture, training, and evaluation processes. It uses TensorFlow for model building and Pandas for data management.

```
import os
import pandas as pd
from google.colab import drive
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from sklearn.preprocessing import LabelEncoder
```

```
from sklearn.model˙selection import train˙test˙split
from imblearn.over˙sampling import RandomOverSampler
from PIL import Image
import matplotlib.pyplot as plt
import tensorflow as tf
from tensorflow.keras.layers import Input, Lambda, Conv2D, MaxPooling2D, Flatten, Dense, Reshape, Dot, Softmax, Multiply, Concatenate
from tensorflow.keras.models import Model
from sklearn.metrics import classification˙report, confusion˙matrix

# Mount Google Drive
drive.mount('/content/drive', force˙remount=True)

# Base path to dataset
drive˙path = "/content/drive/MyDrive/Potato"
subfolders = ["Train", "Test", "Valid"]
image˙paths, labels, split˙info = [], [], []

# Traverse each split folder
for split in subfolders:
    split˙path = os.path.join(drive˙path, split)
    if os.path.exists(split˙path):
        for category in os.listdir(split˙path):
            category˙path = os.path.join(split˙path, category)
            if not os.path.isdir(category˙path):
                continue
            for image˙name in os.listdir(category˙path):
                image˙path = os.path.join(category˙path, image˙name)
                image˙paths.append(image˙path)
                labels.append(category)
                split˙info.append(split)
    else:
        print(f"Split folder not found: –split˙path˝")

# Create DataFrame
df = pd.DataFrame(–"image˙path": image˙paths, "label": labels, "split": split˙info˝)

# Remove corrupt images
def verify˙images(df):
    good = []
    for path in df['image˙path']:
        try:
            img = Image.open(path)
            img.verify()
            good.append(path)
        except:
            continue
    return df[df['image˙path'].isin(good)].reset˙index(drop=True)

df = verify˙images(df)

# Encode labels
le = LabelEncoder()
df['category˙encoded'] = le.fit˙transform(df['label'])

# Balance classes
max˙count = df['category˙encoded'].value˙counts().max()
dfs = []
ros = RandomOverSampler(random˙state=42)
for c in df['category˙encoded'].unique():
    class˙df = df[df['category˙encoded'] == c]
    upsampled = ros.fit˙resample(class˙df, class˙df['category˙encoded'])
```

```
    dfs.append(pd.DataFrame(upsampled[0], columns=df.columns))
df˙resampled = pd.concat(dfs)

# Train-test split
train˙df˙new, test˙df˙new = train˙test˙split(df˙resampled, train˙size=0.8, stratify=df˙resampled[’category˙encoded’], random˙state=42)

# Image generators
img˙size = (224, 224)
batch˙size = 16
train˙gen˙new = ImageDataGenerator(rescale=1./255).flow˙from˙dataframe(
    train˙df˙new, x˙col=’image˙path’, y˙col=’label’, target˙size=img˙size,
    class˙mode=’sparse’, color˙mode=’rgb’, shuffle=True, batch˙size=batch˙size)
valid˙gen˙new = ImageDataGenerator(rescale=1./255).flow˙from˙dataframe(
    df[df[’split’] == ’Valid’], x˙col=’image˙path’, y˙col=’label’, target˙size=img˙size,
    class˙mode=’sparse’, color˙mode=’rgb’, shuffle=True, batch˙size=batch˙size)
test˙gen˙new = ImageDataGenerator(rescale=1./255).flow˙from˙dataframe(
    test˙df˙new, x˙col=’image˙path’, y˙col=’label’, target˙size=img˙size,
    class˙mode=’sparse’, color˙mode=’rgb’, shuffle=False, batch˙size=batch˙size)

# Model architecture
input˙layer = Input(shape=(224, 224, 3))
x = Lambda(lambda x: [x[:, :112, :, :], x[:, 112:, :, :]])(input˙layer)
x1, x2 = x[0], x[1]
x2 = Lambda(lambda x: x)(x2)

for filters in [32, 64, 128]:
    x1 = Conv2D(filters, (3, 3), activation=’relu’, padding=’same’)(x1)
    x1 = MaxPooling2D((2, 2))(x1)
    x2 = Conv2D(filters, (3, 3), activation=’relu’, padding=’same’)(x2)
    x2 = MaxPooling2D((2, 2))(x2)

x1 = Flatten()(x1)
x2 = Flatten()(x2)
x1 = Dense(512, activation=’relu’)(x1)
x2 = Dense(512, activation=’relu’)(x2)
x1 = Reshape((1, 512))(x1)
x2 = Reshape((1, 512))(x2)
x = Dot(axes=(2, 2))([x1, x2])
x = Softmax()(x)
x = Multiply()([x, x1])
x = Concatenate()([x, x2])
x = Reshape((1024,))(x)
x = Dense(256, activation=’relu’)(x)
x = Dense(128, activation=’relu’)(x)
output = Dense(3, activation=’softmax’)(x)

model = Model(inputs=input˙layer, outputs=output)
model.compile(optimizer=’adam’, loss=’sparse˙categorical˙crossentropy’, metrics=[’accuracy’])

# Train model
history = model.fit(train˙gen˙new, validation˙data=valid˙gen˙new, epochs=3)

# Plot results
def plot˙history(history):
    plt.figure(figsize=(12, 5))
    plt.subplot(1, 2, 1)
    plt.plot(history.history[’accuracy’], label=’train’)
    plt.plot(history.history[’val˙accuracy’], label=’val’)
    plt.title(’Accuracy’)
    plt.legend()
```

```
    plt.subplot(1, 2, 2)
    plt.plot(history.history['loss'], label='train')
    plt.plot(history.history['val˙loss'], label='val')
    plt.title('Loss')
    plt.legend()
    plt.show()

plot˙history(history)
```

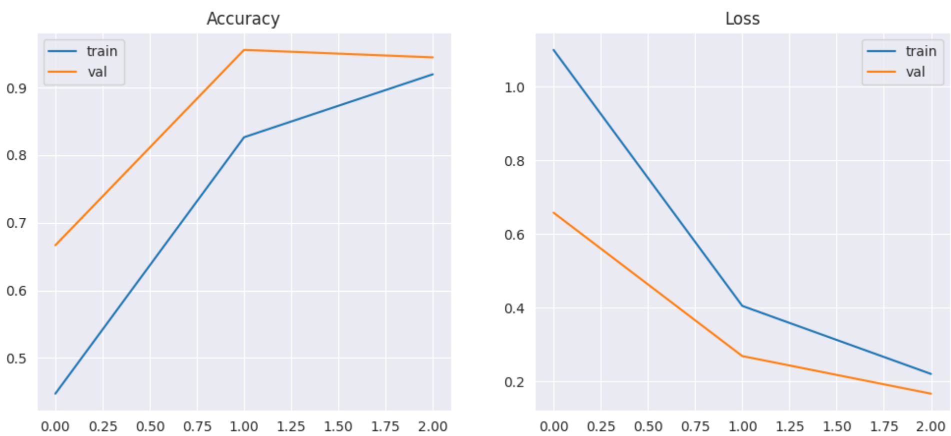

Figure 2: Training and validation accuracy (left) and loss (right) over epochs, as generated by the *plot_history* function.

```
# Test set evaluation
test˙loss, test˙acc = model.evaluate(test˙gen˙new)
print(f"Test Accuracy: -test˙acc:.4f˝ — Test Loss: -test˙loss:.4f˝")

# Confusion matrix and classification report
y˙pred = model.predict(test˙gen˙new)
y˙pred˙classes = y˙pred.argmax(axis=1)
y˙true = test˙gen˙new.classes
labels = list(test˙gen˙new.class˙indices)
print("Classification Report:"n", classification˙report(y˙true, y˙pred˙classes,
    target˙names=labels))
```

3 Model Architecture

The model employs a dual-branch CNN architecture inspired by Siamese networks. The input image (224x224x3) is split into two halves (top and bottom, each 112x224x3). Each branch includes:

- Three convolutional blocks with 32, 64, and 128 filters (3x3 kernels), ReLU activation, and same padding, each followed by 2x2 max-pooling.
- A flatten layer to convert feature maps into vectors.
- A dense layer with 512 units and ReLU activation.

The branch outputs are reshaped to (1, 512), and a dot product is computed to capture feature interactions, followed by a softmax operation. The result is multiplied with one branch's output and concatenated with the other, forming a 1024-dimensional vector. This is processed through dense layers (256 and 128 units with ReLU) and a final softmax layer for three-class classification.

4 Rationale for Architecture Choice

The dual-branch architecture was selected to capture localized disease patterns in potato leaves by processing two image regions independently. This approach, inspired by attention mechanisms, allows the model to focus on spatially distinct features, such as disease spots in different leaf areas. The convolutional layers extract hierarchical features, while the dot product and softmax operations weigh feature importance, enhancing discriminative power. The architecture is lightweight yet effective, achieving 93.33% test accuracy after three epochs, making it suitable for the dataset size and computational constraints.

5 Model Evaluation Report

5.1 Accuracy, Precision, Recall

The model was evaluated on the test set, achieving:

- **Test Accuracy**: 0.9333
- **Test Loss**: 0.4385

The classification report details per-class performance:

Class	Precision	Recall	F1-Score	Support
Potato_Early_blight	0.88	1.00	0.94	30
Potato_Late_blight	1.00	0.90	0.95	30
Potato_healthy	0.94	1.00	0.97	30
Macro Avg	0.94	0.93	0.94	90
Weighted Avg	0.94	0.93	0.94	90

Table 1: Classification Report

5.2 Confusion Matrix

The confusion matrix illustrates classification performance, shown as a table followed by a heatmap for visual clarity.

5.3 IoU, mAP, SSIM, PSNR, MSE

This is an image classification task, so metrics like IoU and mAP (used for object detection/segmentation) and SSIM, PSNR, and MSE (used for image quality or regression) are not applicable. The provided metrics (accuracy, precision, recall, F1-score, and confusion matrix) comprehensively evaluate the model's classification performance.

6 Optimization Report

The following optimization techniques were applied:

	Early Blight	Late Blight	Healthy
Early Blight	30	0	0
Late Blight	4	24	2
Healthy	0	0	30

Table 2: Confusion Matrix

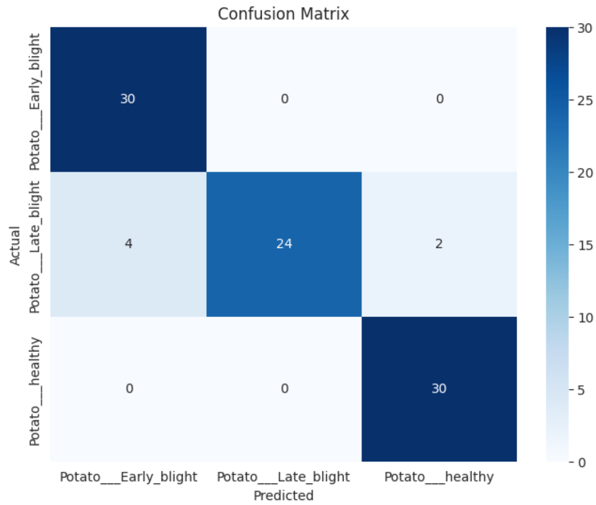

Figure 3: Confusion Matrix Heatmap

- **Data Preprocessing**: Corrupt images were removed using PIL's *verify* method to ensure data integrity.
- **Class Balancing**: Random oversampling (*RandomOverSampler*) was applied, though the dataset was already balanced (500 images per class).
- **Data Normalization**: Pixel values were rescaled to [0, 1] using *ImageDataGenerator*, aiding model convergence.
- **Model Design**: The dual-branch CNN with attention-like mechanisms focused on relevant features, improving classification accuracy.
- **Optimizer**: Adam optimizer was used for efficient gradient descent.
- **Limited Epochs**: Training for three epochs prevented overfitting, as evidenced by high validation accuracy (0.9444).

These techniques resulted in a robust model with 93.33% test accuracy and balanced class performance.

7 Conclusion

This project presents an effective CNN-based solution for potato disease classification, achieving 93.33% test accuracy. The dual-branch architecture and optimization strategies ensure robust performance. Future enhancements could include advanced data augmentation (e.g., rotation, flipping) and transfer learning with models like ResNet to further improve accuracy.

References

[1] Faysal Miah, *Potato Disease Classification Dataset*, Kaggle, 2025. Available at: https://www.kaggle.com/datasets/faysalmiah1721758/potato-dataset.