Deep Learning Potato Disease Classification

Review - 2

Group Members

RA1811003030563 - Rijul Singh Chauhan

RA1811003030328 - Vishal Kumar

RA1811003030301 - Gaurav Kumar

Supervised By:

Sunil Kumar

Designation

Department of Computer Science & Engineering

Faculty of Engineering & Technology SRM
Institute of Science & Technology

March 26, 2022

Objective and Motivation

We will develop an End-to-End project which will be based on Pure Deep Learning. I'm using Pure term here, the reason you'll come to know in my further articles. The reason behind building this project is to detect or identify potato leaf diseases, having a variety of illnesses. Because our naked eyes can't classify them, but Convolutional Neural Network can easily. You won't believe it when I tell you that the error of some pre-trained Neural Network Architectures is approximately 3%, which is even less than the top 5% error of human vision. On large-scale images, the human top-5 error has been reported to be 5.1%, which is higher than pre-trained networks.

Literature Survey

Problem Statement for potato leaf Disease Prediction

Farmers who grow potatoes suffer from serious financial standpoint losses each year which cause several diseases that affect potato plants. The diseases Early Blight and Late Blight are the most frequent. Early blight is caused by fungus and late blight is caused by specific micro-organisms and if farmers detect this disease early and apply appropriate treatment then it can save a lot of waste and prevent economical loss. The treatments for early blight and late blight are a little different so it's important that you accurately identify what kind of disease is there in that potato plant. Behind the scenes, we are going to use Convolutional Neural Network – Deep Learning to diagnose plant diseases.

Potato Leaf Disease Prediction Project Description

Here, we'll develop an end-to-end Deep Learning project in the field of agriculture. We will create a simple Image Classification Model that will categorize Potato Leaf Disease using a simple and classic Convolutional Neural Network Architecture. We'll start with collecting the data, then model building, and finally, we'll use Streamlit to build a web-based application and deploy it on Heroku.

Potato Leaf Disease Prediction Data Collection

Any Data Science project starts with the process of acquiring the data. First, we need to collect data. We have 3 options for collecting data first we can use readymade data we can either buy it from a third-party vendor or get it from Kaggle etc. The second option is we can have a team of Data Annotator whose job is to collect these images from farmers

and annotate those images either healthy potato leaves or having early or late blight diseases. So, this team of annotators works with farmers, go to the fields and they can ask farmers to take a photograph of leaves or they can take photographs themselves and they can classify them with the help of experts from agriculture field. So, they can manually collect the data. But this process will be time-consuming. The third option is writing a web-scraping script to go through different websites which has potato images and collect those images and use different tools to annotate the data. In this project, I am using readymade data that I got from Kaggle.

Dataset Link: https://www.kaggle.com/abdallahalidev/plantvillage-dataset

Platform utilized

The machine over which this research has been accomplished is having an NVIDIA GeForce N16V-GMR1 graphic card. The feature extraction and classification work are done using an orange data mining tool using its image analytics add-on. Orange is an open-source python library us ed as a data visualization and analysis tool. which consists of a canvas interface onto which the us er places the widgets and creates a data analysis workflow. Users can interactively explore visualization using the basic functionalities like showing the data table, feature selection, training predictors, visualizing data elements , etc.

Architectural Design of the Proposed System

Potato Leaf Disease Prediction Data Load Collection

Our dataset must be in the following format.

Potato Leaf Dataset –> main folder
—-| train
—-| Potato_Healthy
—-| img1.jpg
—-| img2.jpg
—-| img3.jpg
—-| Potato_Early_Blight
—-| img1.jpg
—-| img2.jpg
—-| img3.jpg
—-| Potato_Late_Blight

—-| img1.jpg
—-| img2.jpg
—-| img3.jpg
—-| test
—-| Potato_Healthy
—-| img1.jpg
—-| img2.jpg
—-| img3.jpg
—-| Potato_Early_Blight
—-| img1.jpg
—-| img2.jpg
—-| img3.jpg
—-| Potato_Late_Blight
—-| img1.jpg
—-| img2.jpg
—-| img3.jpg

—-| valid
—-| Potato_Healthy
—-| img1.jpg
—-| img2.jpg
—-| img3.jpg
—-| Potato_Early_Blight
—-| img1.jpg
—-| img2.jpg
—-| img3.jpg
—-| Potato_Late_Blight
—-| img1.jpg
—-| img2.jpg
—-| img3.jpg

In this project, we are going to use only 900 images to train our model and 300 images for validation. As we all know, training a deep learning model requires a lot of data. To overcome

this problem, we will use one of the simple and effective methods, called **Data Augmentation**. Let's first see what data augmentation is.

Data Augmentation: Data Augmentation is a process that generates several realistic variants of each training sample, to artificially expand the size of the training dataset. This aids in the reduction of overfitting. In data augmentation, we will slightly shift, rotate, and resize each image in the training set by different percentages, and then add all of the resulting photos to the training set. This allows the model to be more forgiving of changes in the object's orientation, position, and size in the image. The contrast and lighting settings of the photographs can be changed. The images can be flipped horizontally and vertically. We may expand the size of our training set by merging all of the modifications.

Methodology/Technique/Algorithm used in project

The process of detection of diseases from plant leaf images involves various steps and each of those steps can be discussed as follows.

- Let's begin the coding section
- Installing the necessary libraries

```
import numpy as np
import matplotlib.pyplot as plt
import glob
import cv2
import os
import matplotlib.image as mpimg
import random
from sklearn import preprocessing
import tensorflow.keras as keras
```

```
import tensorflow as tf
from keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.utils import to_categorical
```

Let's now add some static variables to aid us in our progress.

```
SIZE = 256
SEED_TRAINING = 121
SEED_TESTING = 197
```

```
SEED_VALIDATION = 164

CHANNELS = 3

n_classes = 3

EPOCHS = 50

BATCH_SIZE = 16

input_shape = (SIZE, SIZE, CHANNELS)
```

To begin, we must first establish the setup for augmentation that we will use on our training dataset.

```
train_datagen = ImageDataGenerator(
        rescale = 1./255,
        rotation_range = 30,
        shear_range = 0.2,
        zoom_range = 0.2,
        width_shift_range=0.05,
        height_shift_range=0.05,
        horizontal_flip = True,
        fill_mode = 'nearest')
```

Here is one thing we need to care that, on validation and test dataset we will not use the same augmentation that we have used on the training dataset, because the validation and testing dataset will only test the performance of our model, and based on it, our model parameters or weights will get tunned. Our objective is to create a generalized and robust model, which we can achieve by training our model on a very large amount of dataset. That's why here we are only applying data augmentation on the training dataset and artificially increasing the size of the training dataset.

```
validation_datagen = ImageDataGenerator(rescale=1./255)

test_datagen = ImageDataGenerator(rescale = 1./255)
```

Let's now load the training, testing, and validation datasets from the directory and perform data augmentation on them.

Note that our data must be in the above-mentioned format. Otherwise, we might be able to get an error or our model's performance may suffer. It will load our dataset from the directory first, then resize all of the images to the same dimension, make batches, and then choose RGB as the color mode.

```
train_generator = train_datagen.flow_from_directory(
        directory = '/content/Potato/Train/',  # this is the input directory
        target_size = (256, 256),  # all images will be resized to 64x64
        batch_size = BATCH_SIZE,
```

```
    class_mode = 'categorical',
    color_mode="rgb")


validation_generator = validation_datagen.flow_from_directory(
        '/content/Potato/Valid/',
        target_size = (256, 256),
        batch_size = BATCH_SIZE,
        class_mode='categorical',
        color_mode="rgb")
```

Let's build a simple and classical Convolutional Neural Network Architecture now.

Our dataset is pre-processed and now we are ready to build our model. We are going to use a Convolutional Neural Network which is one of the famous types of Neural Network Architecture if you are solving Image Classification Problems. Here we are creating simple and classical Neural Network Architecture. Here we are using Keras Sequential API to create our model architecture, it only contains a stack of convolutional and pooling layers. There is approx n number of layers and then at the end, there is a dense layer where we just flatten our feature maps. In the end, we are using a dense layer with a softmax activation function, which will return the likelihood of each class.

```
model = keras.models.Sequential([

    keras.layers.Conv2D(32, (3,3), activation = 'relu', input_shape = input_shape),
```

```
keras.layers.MaxPooling2D((2, 2)),
keras.layers.Dropout(0.5),
keras.layers.Conv2D(64, (3,3), activation = 'relu', padding = 'same'),
keras.layers.MaxPooling2D((2,2)),
keras.layers.Dropout(0.5),
keras.layers.Conv2D(64, (3,3), activation = 'relu', padding = 'same'),
keras.layers.MaxPooling2D((2,2)),
keras.layers.Conv2D(64, (3,3), activation = 'relu', padding = 'same'),
keras.layers.MaxPooling2D((2,2)),
```

```
    keras.layers.Conv2D(64, (3,3), activation = 'relu', padding = 'same'),
    keras.layers.MaxPooling2D((2,2)),
  keras.layers.Conv2D(64, (3,3), activation = 'relu', padding = 'same'),
  keras.layers.MaxPooling2D((2,2)),
  keras.layers.Flatten(),
  keras.layers.Dense(32, activation ='relu'),
  keras.layers.Dense(n_classes, activation='softmax')
 ])
```

The next step is to investigate model architecture.

Let's have a look at the brief summary of our model. We have a total of 185,667 trainable parameters. These are the weights we'll be working with.

```
model.summary()

#---------------Output---------------

Model: "sequential"

_________________________________________________________________

 Layer (type)                 Output Shape              Param #

=================================================================
```

```
conv2d (Conv2D)              (None, 254, 254, 32)      896
max_pooling2d (MaxPooling2D  (None, 127, 127, 32)      0

dropout (Dropout)            (None, 127, 127, 32)      0
conv2d_1 (Conv2D)            (None, 127, 127, 64)      18496
max_pooling2d_1 (MaxPooling 2d)  (None, 63, 63, 64)        0
dropout_1 (Dropout)          (None, 63, 63, 64)        0
conv2d_2 (Conv2D)            (None, 63, 63, 64)        36928
max_pooling2d_2 (MaxPooling 2D) (None, 31, 31, 64)       0
conv2d_3 (Conv2D)            (None, 31, 31, 64)        36928
```

```
max_pooling2d_3 (MaxPooling 2D)  (None, 15, 15, 64)        0
conv2d_4 (Conv2D)                (None, 15, 15, 64)        36928

max_pooling2d_4 (MaxPooling 2D)  (None, 7, 7, 64)          0
conv2d_5 (Conv2D)                (None, 7, 7, 64)          36928
max_pooling2d_5 (MaxPooling 2D) (None, 3, 3, 64)           0
flatten (Flatten)                (None, 576)     0
dense (Dense)                    (None, 32)      18464
dense_1 (Dense)                  (None, 3)       99
=================================================
```

```
Total params: 185,667

Trainable params: 185,667

Non-trainable params: 0
```

Compile the Potato Leaf Disease Prediction model

```
model.compile(optimizer = 'adam',

loss = tf.keras.losses.CategoricalCrossentropy(),

metrics  = ['accuracy'])
```

```
history = model.fit_generator(train_generator,

steps_per_epoch = train_generator.n // train_generator.batch_size,
#The 2 slashes division return rounded integer

epochs = EPOCHS,

validation_data = validation_generator,

validation_steps = validation_generator.n //
validation_generator.batch_size)
```

Let's see how well the model performs on the test data.

Let’s test our model on the test dataset. Before putting our model into production, we must first test it on a test dataset to see how it performs.

```
score = model.evaluate_generator(test_generator)
print('Test loss : ', score[0])
print('Test accuracy : ', score[1])
```

```
------OUTPUT-------
Test Loss : 0.10339429974555969
Test Accuracy : 0.9733333587646484
```

On the test set, we have a 97% accuracy rate, which is rather good. Let's start with some performance graphs.

Let's have a look at the history parameter.

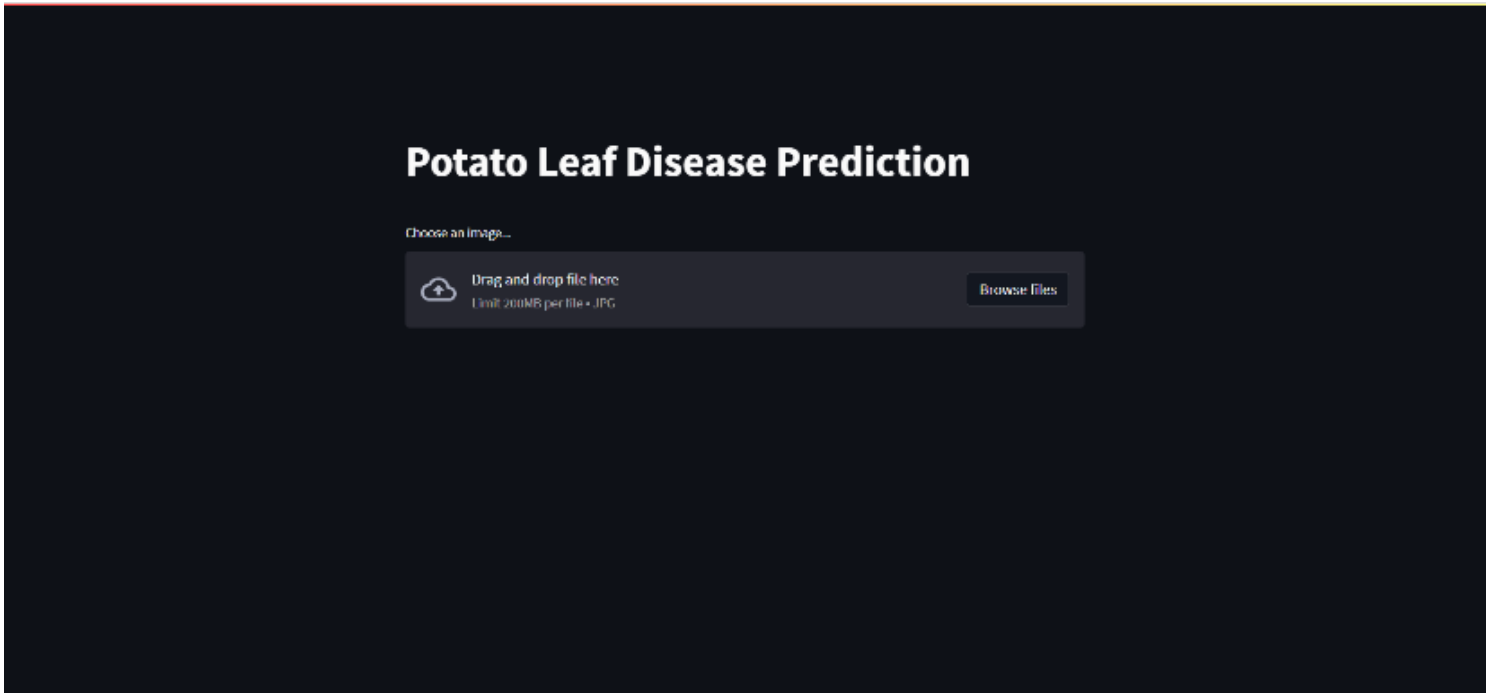
Actually, history is a Keras callback that keeps all epoch history as a list; let's utilize it to plot some intriguing plots. Let's start by putting all of these parameters into variables.

```
acc = history.history['accuracy']

val_acc = history.history['val_accuracy']

loss = history.history['loss']

val_loss = history.history['val_loss']
```

```
plt.figure(figsize=(8, 8))
plt.subplot(1, 2, 1)
plt.plot(range(EPOCHS), acc, label='Training Accuracy')
plt.plot(range(EPOCHS), val_acc, label='Validation Accuracy')
plt.legend(loc='lower right')
plt.title('Training and Validation Accuracy')
plt.subplot(1, 2, 2)
```

```
plt.plot(range(EPOCHS), loss, label='Training Loss')
plt.plot(range(EPOCHS), val_loss, label='Validation Loss')
plt.legend(loc='upper right')
plt.title('Training and Validation Loss')
plt.show()
Let's save our model
# it will save the model
model.save('final_model.h5')
```

now we are making web application and output will be look like this



1. Make a GitHub repository and add your model, web application python file, and model building source code to it.
2. Once we've completed that, create a requirement.txt file in which we'll list any libraries, packages, or modules that we'll be utilizing in this project.
3. Create a Procfile, fill it with the code below, and save it to our project's GitHub repository.

```
web: sh setup.sh && streamlit run your_webapp_name.py
```

4. Create a setup.sh file and paste the code below into it.

```
mkdir -p ~/.streamlit/

echo "

[general]n

email = "mention_your_mailid_here"n
```

```
" > ~/.streamlit/credentials.toml

echo "

[server]n

headless = truen

enableCORS=falsen

port = $PORTn

" > ~/.streamlit/config.toml
```

5. After we've completed all of the procedures, we will go to Heroku and build a new app, then select Connect to GitHub.

References I

- Dataset Credit https://www.kaggle.com/arjuntejaswi/plant-village.
- Machine learning, CNN, Panda tool,
- YouTube Channel for dataset study https://www.youtube.com/watch?v=RLwgHorj7cY&list=PLPbgcxheSpE1gl5WkrwtmRiCwiGMM8NdH
- Image and Diagram taken from google.
- TensorFlow,
- Studied from Wikipedia for project model
- GIthub and hareku to build web site