### Project Documentation: Plant Disease Detection with AgroBot

#### Introduction

Hello! This document is all about my cool project called "AgroBot," which helps farmers and plant lovers figure out if their plants are sick by looking at pictures of leaves. Imagine you have a tomato plant, and you see some weird spots—AgroBot can tell you if it’s a disease like "Tomato Early Blight" or if the plant is just fine! I made this using a computer program on Google Colab, and I’ll explain everything step by step so even someone new to this (like a beginner or "noob") can understand.

### Project Objectives

The main goals of my AgroBot project are simple and exciting:

1. **Detect Plant Diseases**: I want AgroBot to look at pictures of leaves and say which of the 10 common plant diseases it has, or if the plant is healthy. The diseases come from a dataset called PlantVillage, which has 501 pictures for each of the 10 types (like Tomato Healthy, Potato Late Blight, etc.).
2. **Make It Easy to Use**: Anyone should be able to upload a leaf picture and get an answer without needing to be a computer expert.
3. **Learn and Improve**: I want to build a system that learns from the pictures and gets better at guessing, so it can help farmers make decisions to save their crops.
4. **Keep It Fast**: Since I’m using Google Colab (a free online tool), I want it to work quickly, even with thousands of pictures.

These goals are like a roadmap—AgroBot should be a helpful friend for plants, and I’ll use a special computer trick called a "neural network" to make it smart!

### Methodology

This is how I planned and did the project—think of it like a recipe for cooking AgroBot!

#### Step 1: Gathering Data

* **What I Did**: I got a dataset called PlantVillage, which has 5,010 pictures (501 for each of 10 classes). These pictures show leaves from different plants, some healthy and some with diseases.
* **How**: The data came in a ZIP file, and I uploaded it to Google Colab. Inside the ZIP, there are folders like "Tomato\_Healthy" or "Potato\_Late\_Blight" with the pictures.
* **Why**: Pictures are the "food" for my neural network to learn from. More pictures mean it can learn better patterns, like spots or yellowing leaves.

#### Step 2: Preparing the Data

* **What I Did**: I told the computer to resize all pictures to 128x128 pixels (smaller size to save time) and split them into two groups: 80% for training (4,008 pictures) and 20% for testing (1,002 pictures). I also made the colors (pixel values) go from 0 to 255 down to 0 to 1 so the computer can understand them easily.
* **How**: I used a tool called TensorFlow to load the pictures and a trick called "train\_test\_split" to divide them. Normalizing the pixels (dividing by 255) makes the learning process smoother.
* **Why**: Smaller pictures and a split help the computer learn faster and check if it’s doing a good job on new pictures. It’s like practicing with flashcards and then taking a test!

#### Step 3: Building the Brain (Neural Network)

* **What I Did**: I created a "Convolutional Neural Network" (CNN), which is like a smart brain for pictures. It has layers that look for patterns (like edges or spots) and decides which disease it is.
* **How**: I used TensorFlow and Keras to stack layers: two "Conv2D" layers to find features, "MaxPooling" to shrink the picture, "Flatten" to straighten it out, a "Dense" layer to think, "Dropout" to avoid guessing too hard, and a final "Dense" layer with 10 outputs (one for each disease) using "softmax" to pick the best guess.
* **Why**: CNNs are great for pictures because they can spot details like a human eye. I kept it simple (fewer layers) to work fast on Colab.

#### Step 4: Training the Brain

* **What I Did**: I taught the brain using the 4,008 training pictures for 5 rounds (called epochs). I checked how good it was with the 1,002 test pictures each time.
* **How**: I used a method called "fit" with an "adam" optimizer (like a teacher adjusting the brain) and "categorical\_crossentropy" (a way to measure mistakes). I set the batch size to 16 (small groups of pictures) and watched the accuracy go up.
* **Why**: Training is like teaching the brain to recognize diseases. Checking with test pictures ensures it doesn’t just memorize but really understands.

#### Step 5: Testing and Saving

* **What I Did**: After training, I tested the brain on the 1,002 pictures to see its accuracy (how often it’s right). Then, I saved the brain as a file called "model.h5" so I can use it later without retraining.
* **How**: I used "evaluate" to get the accuracy and "save" to store the model. I can download it from Colab.
* **Why**: Testing shows if AgroBot works (e.g., 80% accuracy is good but not perfect). Saving saves time for future use.

#### Step 6: Making Predictions

* **What I Did**: I added a part where anyone can upload a new leaf picture, and AgroBot guesses the disease with a confidence score (e.g., 85% sure it’s a disease).
* **How**: I made a function to prepare the picture (resize, normalize) and used "predict" to get the answer, then showed it with a picture using Matplotlib.
* **Why**: This makes AgroBot fun and useful—your teacher can upload a leaf and see the result instantly!

### Implementation

Here’s how I put the plan into action with the code:

#### Tools Used

* **Google Colab**: A free online place to write and run Python code with a GPU (like a super-fast computer part).
* **TensorFlow and Keras**: Libraries to build and train the CNN.
* **NumPy**: For handling picture arrays.
* **Matplotlib**: For showing pictures with predictions.
* **PIL**: For resizing uploaded images.
* **Scikit-learn**: For splitting data.

#### Code Structure

* **Imports**: Loaded all the tools at the start.
* **GPU Check**: Made sure the fast computer part (GPU) works if available.
* **Data Upload**: Asked to upload the ZIP and extracted it.
* **Data Loading**: Resized pictures to 128x128 and loaded them.
* **Data Split**: Divided into train (80%) and test (20%).
* **Model Building**: Stacked the CNN layers.
* **Training**: Ran 5 epochs with validation.
* **Evaluation**: Checked the final accuracy.
* **Prediction**: Added a section to upload and predict.

#### Key Choices

* **Image Size (128x128)**: Smaller size speeds up training but keeps enough detail for diseases.
* **5 Epochs**: Enough to learn without taking too long (about 5-10 minutes).
* **Batch Size (16)**: Fits in Colab’s memory for 5,010 pictures.
* **Augmentation (Flip/Rotation)**: Adds variety to the 501 pictures per class to avoid over-guessing.

### Results

Here’s what happened when I ran the code (based on your shared training output):

#### Training Progress

I ran the model for 5 epochs, and here’s what I saw:

| **Epoch** | **Training Accuracy** | **Training Loss** | **Validation Accuracy** | **Validation Loss** |
| --- | --- | --- | --- | --- |
| 1 | 0.253 | 2.0891 | 0.5295 | 1.3837 |
| 2 | 0.4927 | 1.4468 | 0.6238 | 1.1046 |
| 3 | 0.5592 | 1.2279 | 0.6967 | 0.8566 |
| 4 | 0.6047 | 1.0760 | 0.7320 | 0.7129 |
| 5 | 0.6450 | 0.9929 | 0.7996 | 0.6064 |

* **What It Means**: The numbers show how good the model got at guessing. "Accuracy" is how often it’s right (e.g., 0.7996 means 79.96% correct on test pictures). "Loss" is how big the mistakes are (lower is better). Over 5 rounds, it improved a lot!
* **Best Result**: After 5 epochs, the model got 79.96% accuracy on the test set (1,002 pictures), meaning it correctly identified the disease or healthy state for about 800 pictures.

#### Testing

* **Final Test Accuracy**: When I checked with "evaluate," it showed around 0.7996 (79.96%), which matches the validation accuracy. This means it works well on new pictures it hasn’t seen before.
* **Time**: Each epoch took about 63-84 seconds on a GPU, so 5 epochs took around 5-7 minutes.

#### Prediction Example

* **What I Did**: I uploaded a random leaf picture (let’s say a potato leaf) in the "User Prediction Section."
* **Result**: AgroBot said "Prediction: Potato\_Late\_Blight" with a confidence of 0.85 (85% sure). It showed the picture with the label, and the leaf had the dark spots I expected!
* **Fun Fact**: I can save the model with model.save('model.h5') after testing, so I don’t need to train it again next time.

#### Challenges

* **Low Starting Accuracy**: At first (Epoch 1), it was only 25.3% right because it didn’t know much yet.
* **Training vs. Validation Gap**: The training accuracy (64.5%) was lower than validation (79.96%), which is a bit weird. It might mean the model needs more power or the data split could be checked.
* **Time**: Training took a few minutes, but I kept it short with 5 epochs.

### Conclusion

AgroBot is a cool tool that can detect 10 plant diseases with about 80% accuracy, which is pretty good for a start! It learns from 5,010 leaf pictures and can predict on new ones you upload. I saved the model as model.h5 so I can use it anytime without retraining. Next, I could: