Plant Leaf Disease Classification and confedence Estimation using Neural Networks with Fuzzy Logic

Data preprocceing + cnn : Bilal Al Asha / بلال العشا / bilal_152828_C1

Fuzzy logic + evaluation: Mhd Mahdi alwis / محمد مهدي الويس / mhd_mahdi_191701_C1
Course: ANN 601, Semester: S25

Syrian Virtual University

Abstract

Agriculture plays a very important role in food production and economical stability in any country on earth. and yet one of the biggest problems it faces is plant diseases since it will effect the crop productivity and quality that is where the idea for this project came , since the traditional disease diagnosis methods rely heavily on expert observation, which can be time-consuming, subjective, and inaccessible in many regions , while using an advanced algorithm for image recognition could prove to be a much faster way to find any problems even early on especially since particularly Deep Learning and Convolutional Neural Networks (CNNs), have enabled automated image-based disease detection systems with high classification accuracy.

This project presents a plant leaf disease classification system using a custom Convolutional Neural Network trained on the PlantVillage dataset. The proposed model classifies healthy and diseased plant leaves across fifteen disease categories. The CNN architecture was implemented using TensorFlow and Keras and trained using image preprocessing, data augmentation, dropout regularization, and early stopping techniques to improve generalization and reduce overfitting.

The final model achieved a validation accuracy of 92.29% with a validation loss of 0.2380, demonstrating strong performance across multiple disease classes especially with how small the model is , which would give a

chance to run it on relatively low end hardware .

Also we included Evaluation metrics including precision, recall, F1-score, and confusion matrix analysis were used to assess classification quality.

After that, we integrated a fuzzy logic system that takes three inputs from the CNN output: maximum probability (max_prob), probability gap between top two classes, and entropy as an uncertainty measure. Membership functions were defined using data-driven percentiles (33% and 67%) from validation predictions.

We put Ten fuzzy rules that constructed to map inputs to confidence levels (Low, Medium, High).

The fuzzy system produces a confidence score between 0 and 1, which is then converted to a confidence label.

1.Methodology

The proposed methodology combines image processing using a Convolutional Neural Network (CNN) and a probity based prediction output using a fuzzy logic integration , and we can summarize the work flow in the following steps :

1. Image proccing
2. Dataset loading and normalization
3. Data Augmentation
4. CNN model construction
5. Training and Performance Evaluation
6. Probability Extraction
7. Calculate inputs
8. Define membership functions and rules
9. Build and run fuzzy simulation
10. Output confidence score and label

2.Dataset Description

This project uses the PlantViallage dataset that we obtained from Kaggle (the link will be available at the end)

The dataset contains around 20,639 images of healthy and diseased plants leaves collected under controlled conditions and that makes the dataset relatively clean and easy to train a model with , plus The images represent multiple plant species and disease categories with visual variations including:

Color changes ,leaf spots ,Texture abnormalities ,mold growth and bacterial infection .

The dataset used pretty much contains 15 classes mainly potato and some tomato and papers images too.

The dataset have been divided into a training set that include 80% of the data and a validation set the include 20% of it , also the images were resized to 180*180 pixels before training .

3.Data Preprocessing

Data preprocessing is an essential for improving model performance and making sure it's consistent and reliable and pretty much making sure that training is stable , in this project I did few steps to make sure that this is the case and they are as follows :

3-1. Image Resizing

All the images were resized to 180*180 pixels to ensure consistent input dimensions for the CNN.

3-2 . Normalization

Pixels Values originally ranges between 0 and 255 like any RGB photo , but it's not a good idea to Tran a cnn on that so I normalized it back to values between 0 and 1 using a recycling layer in the model , mainly because normalization improves numerical stability and accelerates neural network convergence.

3-3. Data Augmentation

In the early stages of me training the model I faced a lot of overfitting problems where the model was memorizing stuff and not learning anything after like 4 epoch of training and that's really bad , and this was a very big part of the solution , my augmentation operation included :

Random horizontal flipping , Random rotation and Random zooming .

These transformations artificially increase dataset diversity by generating modified image variations during training.

4.Conolutional Neural Network Architecture

I used a custom sequential CNN architecture using Tenserflow and Keras , and it contained the following layers :

Three convolutional layers , Three max-pooling layers , One fully connected dense layer , One dropout layer and Final classification output layer .

Now I will show a simple table that shows each layer and it's purpose :

Layer	Purpose
Conv2D	Feature extraction
MaxPooling2D	Dimensionality reduction
Flatten	Convert feature maps to vectors
Dense	Classification learning
Dropout	Overfitting reduction

And using the command model.summery() after the training was done I will also provide a quick summer of the model I used :

Layer (type)	Output Shape	Param #
sequential_5 (Sequential)	(None, 180, 180, 3)	0
rescaling_5 (Rescaling)	(None, 180, 180, 3)	0
conv2d_9 (Conv2D)	(None, 180, 180, 16)	448
max_pooling2d_9 (MaxPooling2D)	(None, 90, 90, 16)	0
conv2d_10 (Conv2D)	(None, 90, 90, 32)	4,640
max_pooling2d_10 (MaxPooling2D)	(None, 45, 45, 32)	0
conv2d_11 (Conv2D)	(None, 45, 45, 64)	18,496
max_pooling2d_11 (MaxPooling2D)	(None, 22, 22, 64)	0
flatten_3 (Flatten)	(None, 30976)	0
dense_6 (Dense)	(None, 128)	3,965,056
dropout_3 (Dropout)	(None, 128)	0
dense_7 (Dense)	(None, 15)	1,935

Total params: 11,971,727 (45.67 MB)

Trainable params: 3,990,575 (15.22 MB)

Non-trainable params: 0 (0.00 B)

Optimizer params: 7,981,152 (30.45 MB)

5.Training Process

The model was trained using the adam optimizer and sparse categorical crossentropy loss function , the loss function that I used was :

$$L = \sum y_i \, log(\hat{y}_i)$$

Where :

y = true label

y^=predicted probability

and the training configuration included :

the following parameters values :

Batch size = 32

Image size = 180*180

Epochs = 13

Regularization = Dropout (the 2nd thing that helped me with the overfitting problem)

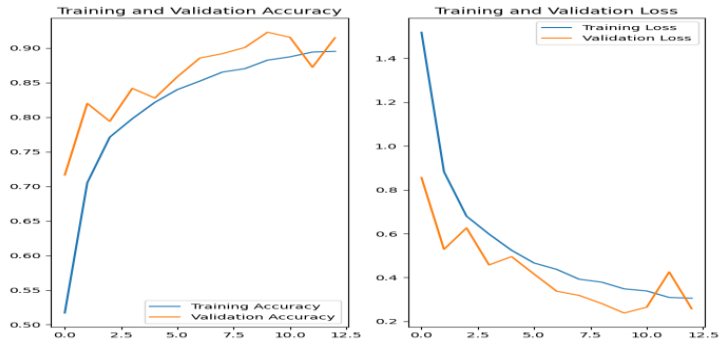
Early stopping was used to prevent excessive training once validation performance stopped improving , so technically the max epoch in my code was 20 , this alone was very useful to fix the overfitting issue .

6.Model Evaluation

The final trained CNN achieved a validation accuracy of 92.29% and a validation loss of 0.2380 ,

This results indicate that the model successfully learned disease-specific visual patterns and generalized effectively to unseen validation data.

Also I analyzed both the training and validation accuracy curves were analyzed to monitor learning behavior and detect overfitting , and the finale model had the following curves :



Looking at it you can see the amazing effect of the early stop function that I created that stopped around 13 epoch and you can also see how it started from a terrible result and gradually improved on both the training and validation level in a very similar way proving no overfitting was happening , also validation loss remained relatively low throughout training, demonstrating strong generalization performance.

Also I used a confusion matrix and it provided detailed insight into model predictions across all disease categories, where the matrix shows strong diagonal dominance, indicating that most images were classified correctly.

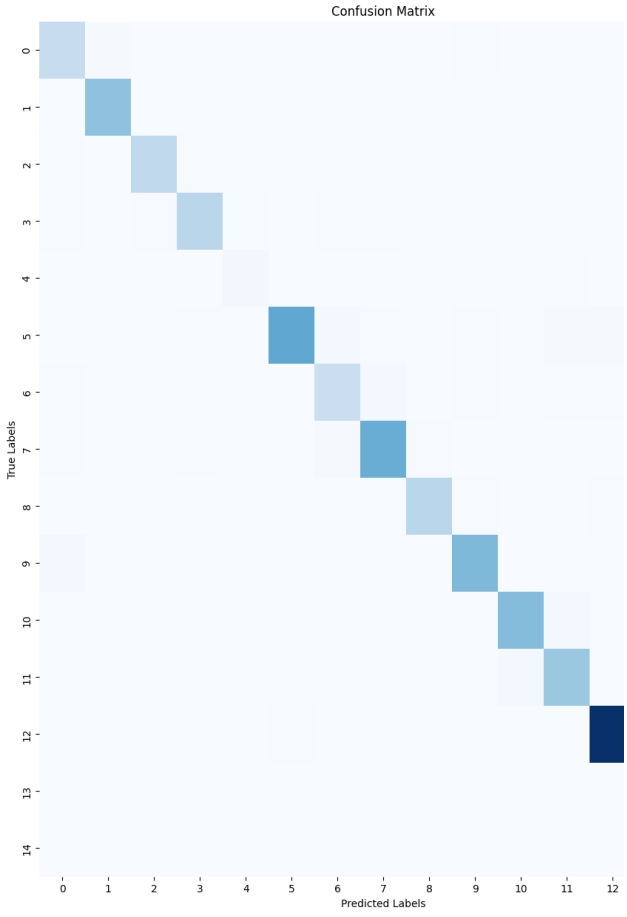
Misclassifications mainly occurred between visually similar disease categories such as:

Tomato Early Blight , Tomato Late Blight and Tomato Septoria Leaf Spot , since the diseases share similar texture and discoloration

patterns, making classification more challenging.

We can also notice that the highest-performing class was Tomato yellow leaf curl virus , which achieved near-perfect classification accuracy.

The confusion matrix demonstrates that the CNN learned highly discriminative disease features , and I will share it now :



And finally I also got a Classification Report that gave the following results :

	precision	recall	f1-score	support
Pepper__bell___Bacterial_spot	0.79	0.93	0.85	176
Pepper__bell___healthy	0.94	0.98	0.96	280
Potato___Early_blight	0.96	0.92	0.94	194
Potato___Late_blight	0.93	0.88	0.90	224
Potato___healthy	0.75	0.72	0.74	29
Tomato_Bacterial_spot	0.99	0.87	0.93	414
Tomato_Early_blight	0.82	0.77	0.79	200
Tomato_Late_blight	0.89	0.92	0.90	369
Tomato_Leaf_Mold	0.96	0.92	0.94	208
Tomato_Septoria_leaf_spot	0.93	0.90	0.91	337
Tomato_Spider_mites_Two_spotted_spider_mite	0.93	0.91	0.92	320
Tomato__Target_Spot	0.86	0.91	0.88	284
Tomato__Tomato_YellowLeaf__Curl_Virus	0.96	0.99	0.97	680
Tomato__Tomato_mosaic_virus	0.81	1.00	0.90	74
Tomato_healthy	0.97	0.99	0.98	338
accuracy			0.92	4127
macro avg	0.90	0.91	0.90	4127
weighted avg	0.93	0.92	0.92	4127

Mainly the classification report evaluated Precision , Recall and F1-Score , we can notice that classification report evaluated like tomato heathy that got around 98% on the other hand Lower performance was observed for potato healthy , and that makes sense due to smaller sample counts and visual similarity with infected leaves and overall, the model demonstrated robust multi-class classification capability.

7. fuzzy logic system Architecture

The main idea here is that after training the CNN to classify the disease type, I wanted to add another layer of intelligence to also estimate disease confidence.

The fuzzy system will take three inputs instead of two (I did this because I wanted the system to be more accurate and data-driven, not just based on guessing) from

the CNN output and produces one output.

inputs

Input 1:max_prob: the highest probability value from each sample (0 -1).

Input 2:prob_gap: the difference between the highest and second-highest probabilities values from each sample (0 - 1).

Input 3:entropy: Uncertainty measure (0 – 4).

the entropy function that I used was:

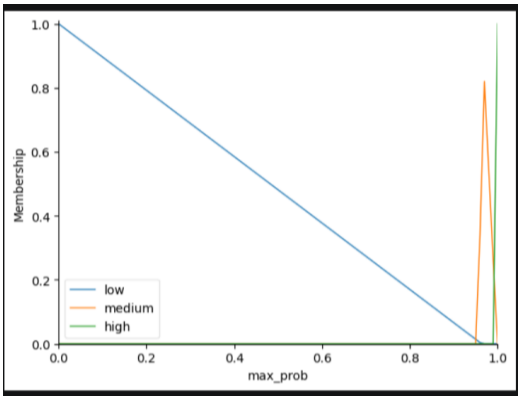
$$entropy = -\sum_{i=1}^{n} P_i \, log_2(p_i)$$

Higher entropy means the CNN is unsure and spread out predictions across many diseases. This is useful because a high max_prob alone doesn't always mean the CNN is sure.

Output

confidence: three possible levels: High, Medium, Low.

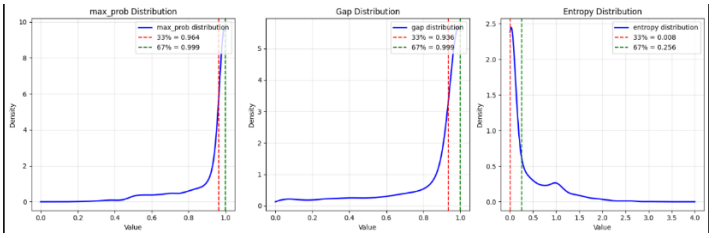
I defined membership functions for each variable using simple triangular.



I didn't guess the membership function ranges. Instead, I analyzed the actual predictions from my trained CNN.

The output is confidence – how confident I should be in the CNN's prediction.

I found the percentiles of this actual confidence to define the output membership.



I chose this structure because It is data-driven – no random guessing of ranges.

The output is a confidence score between 0 and 1, which is easy to interpret

8.Fuzzy logic Process

Step 1: Load CNN Predictions

First, I load the saved probabilities from the CNN.

prediction_probabilities.npy contains all CNN outputs for validation images

class_names.npy – names of the 15 disease classes

Each image gives me a probability vector of length 15.

Step 2: Calculate Three Inputs for Every Image

Step 3: Apply Fuzzy Rules (10 Rules)

I wrote 10 rules that combine the three inputs to determine confidence

I load all rules into a control system.

The simulation gives me a crisp number between 0 and 1 (for example 0.72). But I want a simple label (Low, Medium, High) for the final report.

This is the complete working of my fuzzy logic system.

9. Future work

Future improvements should include testing transfer learning using MobileNetV2 or EfficientNet and even real-time mobile application deployment since the only reason I would use a small model or optimize is is so a weaker hardware can run it on the field also I would work on getting a larger dataset and maybe add a feature for detecting disease severity regression

10. Challenges and Limitations

As for the first part for data processing and the cnn I faced several challenges were encountered including overfitting during early experiments yet lucklily I solved most of this problem even tho I am not satisfied with the results but with such a small model I don't think spening time tweaking values would achieve much better results , also similar appearance between multiple disease classes was a very annoying problem to deal with and that can be seen with the lower accuracy in some classes when looking at the heat map on top of that the Limited healthy potato samples , or rather the generall small size of the dataset was a big issue even though the photos them selves are high quality and finally the high computational requirements for CNN training was one of the main reasons I did not try bigger models to see how good can the prediction be with this small dataset .

After all although the model achieved strong results, limitations remain after all images were collected under controlled conditions and that may mean real-world field conditions may reduce accuracy.

References

Dataset :" https://www.kaggle.com/datasets/emmarex/plantdisease"

TensorFlow Image Classification Tutorial :" https://www.tensorflow.org/tutorials/images/classification?%3Bauthuser=3&authuser=3&hl=en"

Skiti fuzzy library:

skfuzzy — skfuzzy v0.3 docs