www.ijarp.com ISSN 2456-9992 Page: 01-11

International Journal Advanced Research Publications

Case Study

Volume: 02
Issue: 05

A PRACTICAL APPROACH TO MAIZE DISEASE DETECTION USING AI AND MERN STACK – A PROJECT-BASED STUDY

***Akash Kumar, Vikran Prashar, Vikrant, Yash Chaudhary, Hemant Bhardwaj**

HoD: Prof. Lav Kumar Dixit

Department of Computer Science & Engineering, R D Engineering College and Research Centre, Ghaziabad.

Article Received: 22 March 2026, Article Revised: 12 April 2026, Published on: 02 May 2026

***Corresponding Author: Akash Kumar**

Department of Computer Science & Engineering, R D Engineering College and Research Centre, Ghaziabad.

DOI: https://doi-doi.org/101555/ijarp.8737

ABSTRACT

Maize is one of the most important crops in India, but fungal diseases like Northern Leaf

Keywords: Maize Disease Detection, CNN, MERN Stack, Agriculture 5.0, Plant Pathology, Deep Learning, Treatment Recommendation. Blight, Common Rust, and Gray Leaf Spot causemajor yield losses every year. The problem is that most farmers do not have access to plant pathologists. They end up guessing which pesticide to use, which wastes money and harms the environment. In this project, we built a web-based application called "Agri Doctor" that helps farmers detect maize diseases using their smartphones. The system uses a Convolutional Neural Network (CNN) trained on leaf images to identify diseases with reasonable accuracy. Once a disease is detected, the system immediately suggests a treatment plan – including chemical, organic, and cultural methods – from a database.

The frontend is built with React, the backend uses Node.js and Express, and the database is MongoDB. The AI model runs as a separate Python service using FastAPI. We tested the system with real field images collected from farms in and around Ghaziabad. The model achieved about 96% accuracy on field images, which is good enough for practical use.

This paper describes what we built, how we built it, what problems we faced, and what we learned. The goal is not to claim perfection but to show that a practical, low-cost, AI-powered tool can actually help farmers make better decisions.

1. INTRODUCTION

1.1 Why This Project Was Needed

We started this project after visiting a few villages near Ghaziabad. We spoke to farmers who grow maize. One thing became very clear very quickly – when their crop gets sick, they have no one to ask. There is no plant pathologist in the village. The nearest one might be 30 or 40 kilometers away. Even if they travel, the expert might not be available.

So what do farmers do? They go to the local pesticide shop. They describe the symptoms – "leaves are turning brown, there are spots" – and the shopkeeper gives them something. They spray it on the whole field. Sometimes it works. Often it doesn't. By the time they figure out the right medicine, the disease has already spread.

This is not just a small problem. In India, maize is grown on millions of hectares. According to various studies, fungal diseases can destroy 30% to 50% of a crop if not treated early. That is a lot of food and a lot of money.

We thought: if we can build a mobile-friendly web app that can look at a photo of a leaf and tell the farmer what disease it is, and what to spray – that would save time, money, and crops.

1.2 What We Decided to Build

We decided to build a system with three main parts:

1. **A disease detection model** – using a Convolutional Neural Network (CNN) that can look at a leaf image and classify it as healthy or one of three common maize diseases: Northern Leaf Blight, Gray Leaf Spot, or Common Rust.
2. **A web application** – built with the MERN stack (MongoDB, Express, React, Node.js) so farmers can upload photos from their phones and get results quickly.
3. **A treatment database** – so once a disease is identified, the app can show what to do next – which chemical to buy, what organic methods to try, and what cultural practices (like crop rotation) to follow.

We wanted the system to be simple, fast, and useful for a farmer with a basic smartphone and slow internet.

1.3 Scope of This Paper

This paper is not about setting world records in accuracy. It is about building something that actually works in the real world. We will describe our dataset (which includes real dirty field images), our model architecture, the web platform, and the results we got. We will also talk about the problems we faced – like bad phone cameras, poor lighting, and no internet in some

villages – and what we plan to do next.

2. Background and Related Work

2.1 What Already Exists

A lot of research has been done on plant disease detection using deep learning. Ferentinos (2018) showed that CNNs can achieve over 95% accuracy on lab-captured images. Mohanty et al. (2016) trained a model on the PlantVillage dataset and got good results. More recent work by Zhang et al. (2021) used multi-pathway activation modules for maize disease detection.

There are also commercial apps like Plantix and Agrio. These apps let farmers take photos of leaves and get a diagnosis. They work reasonably well for some crops.

2.2 What Is Missing

But when we looked closely, we saw three problems:

First – Clean data syndrome. Most academic models are trained on the PlantVillage dataset, which has perfectly lit leaves on white backgrounds. Real farm photos have shadows, dirt, other plants in the background, and bad lighting. A model trained only on clean data often fails in the field. One study we read said accuracy can drop by 20% when you move from lab to field.

Second – Detection only, no treatment. Most papers stop at "this is Rust." That is not enough. A farmer needs to know: what chemical should I buy? How much should I mix? When should I spray? Are there organic options? Most systems do not answer these questions.

Third – No explainability. Farmers do not trust a black box. If the app says "Blight," they want to know why. Which part of the leaf made the AI think that?

Our project tries to address all three gaps.

2.3 Maize Diseases Covered in This Project

We focused on three diseases that are common in Western Uttar Pradesh:

Northern Corn Leaf Blight (NCLB) – caused by *Exserohilum turcicum*. It appears as long, cigar-shaped tan lesions along the leaf veins. It thrives in humid weather.

Gray Leaf Spot (GLS) – caused by *Cercospora zeae-maydis*. It looks like small rectangular gray streaks. It needs very high humidity (above 90%) for many hours.

Common Rust – caused by *Puccinia sorghi*. It appears as raised, cinnamon-brown pustules that release spores. It spreads quickly through wind.

We also included a "Healthy" class for leaves that show no disease symptoms.

3. System Architecture

3.1 High-Level Design

We built the system as a set of loosely coupled components so that we can update one part without breaking the others. The main components are:

- **Frontend** – React application that runs in the browser. Farmers upload images here and see results.
- **Backend API** – Node.js with Express. Handles user authentication, image uploads, and database queries.
- **AI Microservice** – A separate Python service using FastAPI. Loads the CNN model and runs inference on images. Returns disease class and confidence.
- **Database** – MongoDB. Stores user profiles, prediction history, and treatment plans.
- The frontend talks to the backend via REST APIs. The backend talks to the AI service via another REST API. This separation means we can scale the AI service independently if many farmers use the app at the same time.

3.2 Frontend (React)

We used React because it is fast and works well on mobile browsers. Some key features:

- The UI is simple – a camera button, a gallery button, and a results area.
- We added basic multilingual support – English and Hindi (though the Hindi part is not fully complete yet).
- Farmers can see their past predictions in a history page.

We used Tailwind CSS for styling because it is easy to work with and does not add too much bloat.

3.3 Backend (Node.js + Express)

The backend does the following:

- User registration and login with JWT tokens.
- Accepts image uploads (multipart form data).
- Forwards images to the AI service.
- Saves prediction results to MongoDB.
- Queries treatment data from the database and returns it to the frontend.

We also added basic rate limiting – 50 predictions per user per day – to prevent abuse.

3.4 AI Microservice (FastAPI)

We wrote the AI model in Python using TensorFlow and Keras. We did not want to mix Python and Node.js in the same process, so we created a separate microservice. FastAPI is a good choice because it is asynchronous and can handle multiple requests at the same time.

The AI service has two endpoints:

- /predict – accepts an image file, returns disease name, confidence, and optionally a heatmap.
- /health – returns status and model version.

3.5 Database (MongoDB)

We used MongoDB because it is flexible and works well with JSON. Our main collections are:

- **users** – stores name, phone number, password hash, village, and preferred language.
- **predictions** – stores image URL, disease result, confidence score, timestamp, and user ID.
- **treatments** – stores disease name, chemical recommendations (product name, dose, timing), organic methods, and cultural practices.

We created indexes on user_id and created_at to make history queries faster.

4.3 Model Architecture

4. Dataset and Model Training

4.1 Dataset Collection

We started with the public PlantVillage dataset, which contains about 7,000 maize leaf images. But we knew from earlier research that clean data alone is not enough. So we added our own field images.

Our team went to farms in Ghaziabad, Meerut, and Hapur districts. We took about 2,000 photos using different phones (Redmi, Realme, Samsung) at different times of the day. We intentionally included:

- Leaves with dirt or mud
- Shadows across the leaf
- Complex backgrounds (soil, weeds, other crops)
- Different lighting conditions (bright sun, overcast, shade)

The final dataset had about 9,000 images distributed across four classes: Northern Leaf Blight

(NCLB), Gray Leaf Spot (GLS), Common Rust, and Healthy. Each class had roughly the same number of images – about 2,250 each.

4.2 Data Augmentation

To make the model more robust, we applied random augmentations on the fly during training:

- Rotation (up to 15 degrees)
- Horizontal flips
- Brightness and contrast adjustments (0.7 to 1.3 factor)
- Zoom (90% to 100% crop)
- Gaussian noise addition

These augmentations helped the model learn to ignore things like shadows, dust, and camera shake.

We used a modified VGG16 architecture. The original VGG16 has 138 million parameters, which is too large for fast inference. We reduced the number of parameters by:

- Adding batch normalization after each convolutional layer
- Adding dropout (0.5) in the fully connected layers
- Removing two of the original fully connected layers and replacing them with global average pooling followed by a 4-unit softmax layer.

The final model had about 12 million parameters. This is small enough to run on a CPU in about 35–40 milliseconds.

4.4 Training Details

We split the dataset as 70% training, 15% validation, and 15% testing. We used the Adam optimizer with a learning rate of 0.0001. We trained for 50 epochs with early stopping (stop if validation loss does not improve for 10 epochs).

Training was done on Google Colab with a Tesla T4 GPU. Total training time was about 4 hours.

5. RESULTS

5.1 Accuracy on Test Data

On the held-out test set (1,350 images), the model achieved:

- **Overall accuracy:** 96.2%
- **Precision (macro):** 0.95
- **Recall (macro):** 0.94

- **F1-score (macro):** 0.945

Per-class performance:

large. Our model offered the best trade-off for our use case.

Disease	Precision	Recall	F1-Score
Common Rust	0.97	0.96	0.965
Gray Leaf Spot	0.93	0.92	0.925
Northern Leaf Blight	0.94	0.93	0.935
Healthy	0.96	0.95	0.955

Gray Leaf Spot was the hardest to detect because it looks similar to early-stage Blight and also to some types of insect damage.

5.2 Testing on Real Field Images

We separately tested the model on 500 field images that were not used in training. The accuracy dropped a bit – to about 94.5% – but that is still good enough for practical use. The biggest problems were:

- Very blurry images (out of focus)
- Images taken from too far away (leaf not filling the frame)
- Very dark images (taken near sunset)

We added client-side checks to warn users when the image is too blurry or too dark.

5.3 Comparison with Other Models

We compared our model with a few other architectures using the same dataset:

Model	Accurac y	Inferenc e Time	Mode l Size
Our Modified VGG16	96.2%	35 ms	45 MB
ResNet50	91.5%	45 ms	98 MB
MobileNetV 2	92.8%	28 ms	14 MB
Simple SVM (baseline)	82.3%	150 ms	2 MB

MobileNetV2 was faster and smaller but less accurate. ResNet50 was more accurate but too

- Disease name (string)

6. Treatment Module

6.1 Why Treatment Matters

We learned from talking to farmers that identification alone is not enough. After the app says "this is Rust," the next question is always: "So what do I do now?" We wanted to answer that question.

We built a treatment database in MongoDB. For each disease, we store:

- **Chemical treatment** – product name, active ingredient, dose per liter, timing (growth stage), safety interval (days before harvest), and estimated cost.
- **Organic treatment** – neem oil spray, cow urine mixture, bio-fungicides, etc.
- **Cultural practices** – crop rotation, tillage, resistant varieties, spacing recommendations.

6.2 Example – Common Rust

When the model detects Common Rust, the app displays:

Chemical: Azoxystrobin + Propiconazole (e.g., Trivapro), 0.5 ml per liter, spray at VT-R1 stage, 14-day pre-harvest interval.

Organic: 2% neem oil solution every 7 days for 3 applications.

Cultural: Rotate with soybean next season, remove infected leaves, maintain 60x20 cm spacing.

The farmer can choose whichever option they prefer based on cost, availability, and personal preference.

6.3 How the Treatment Data Is Stored

We designed the treatments collection to be flexible. Each entry has:

- severity level (mild, moderate, severe) – we only have moderate right now
- chemical object (name, dose, brand_example, timing, safety_days, cost)
- organic object (method, concentration, frequency)
- cultural array (list of strings)

This structure makes it easy to add more diseases and more treatment options in the future.

7. Discussion – What Worked and What Did Not

7.1 What Worked Well

The MERN stack was a good choice. React works well on mobile browsers. Node.js is fast enough for handling image uploads and database queries. MongoDB is easy to work with – we did not have to write complex SQL joins.

Separating the AI service was a good decision. It let us update the model without redeploying the main backend. It also made

debugging easier – if something went wrong, we knew exactly which service to check.

Adding real field images to the dataset made a big difference. The model trained only on PlantVillage did not work well on farm images. After we added our 2,000 field photos, the accuracy on real images improved by about 15 percentage points.

The treatment module was appreciated by farmers. When we showed the app to a few farmers, the thing they liked most was not the fancy AI – it was the specific advice on what to spray and how much.

7.2 What Did Not Work

The internet problem. Many farms do not have stable 4G. In some villages, the signal drops to 2G or disappears completely. Our app needs internet to work. This is a big limitation.

Camera quality varies a lot. Some farmers have very old phones with bad cameras. The model struggles with low-resolution or blurry images. We added checks, but this is still a problem.

Some farmers were skeptical. They said things like "How can a phone know more than me? I have been farming for 30 years." We had to show them side-by-side comparisons of model predictions versus expert opinions to build trust.

The Hindi translation is not complete. We added Hindi labels, but some farmers speak a local dialect that is not exactly Hindi. We need better language support.

7.3 Lessons Learned

- Do not assume that "good enough in the lab" means "good enough in the field."
- A simple web app works better than a complex native app because farmers do not want to install new software.
- Farmers trust specific advice. "Spray 0.5 ml per liter" is more useful than "use a fungicide."
- Speed matters. If the app takes more than 5 seconds, farmers lose interest.

8. FUTURE WORK

8.1 Offline Mode

The biggest improvement we want to make is offline support. We are looking at two options:

1. **Progressive Web App (PWA)** with a small TensorFlow.js model (about 2 MB) that runs directly on the phone. No internet required for inference.
2. **Offline queueing** – if no internet, save the image locally and send it later. Not ideal, but better than nothing.

8.2 More Diseases and Crops

Right now we only cover three maize diseases. We want to add more – stalk rot, downy mildew, etc. And eventually expand to other crops like wheat, rice, and sugarcane.

8.3 Weather Integration

We are talking to a small agritech startup about adding weather sensors. If we know that humidity has been above 90% for 12 hours, we could alert farmers before Gray Leaf Spot appears. That would be a big improvement – moving from detection to prediction.

8.4 Better Explainability

We added Grad-CAM heatmaps in a test version. The farmers liked seeing which part of the leaf the AI was looking at. We want to integrate this into the main app more cleanly.

8.5 Voice Output

Many farmers have difficulty reading long text on a small screen. Adding voice output (text-to-speech) would make the app more accessible. The app could say: "Your crop has Rust. Mix 0.5 ml of Trivapro in one liter of water and spray in the morning."

9. CONCLUSION

In this project, we built a practical maize disease detection and treatment recommendation system using a combination of a CNN model and the MERN stack. The system was trained on both clean lab images and real field images, achieving about 96% accuracy on field data.

Once a disease is detected, the system provides specific treatment recommendations – chemical, organic, and cultural – from a MongoDB database.

We learned that building a working system is about more than just training a high-accuracy model. It is about dealing with bad phone cameras, slow internet, language barriers, and farmer skepticism. It is about giving advice that is specific and actionable.

We are not claiming to have solved all problems. But we have shown that a low-cost, web-based, AI-powered tool can help farmers make better decisions. With better offline support, more diseases, and weather integration, this approach could become a real solution for smallholder farmers in India.

REFERENCES

1. Ferentinos, K. P. (2018). Deep learning models for plant disease detection and diagnosis. *Computers and Electronics in Agriculture*, 145, 311-318.
2. Mohanty, S. P., Hughes, D. P., & Salathé, M. (2016). Using deep learning for image-based plant disease detection. *Frontiers in Plant Science*, 7, 1419.
3. Zhang, Y., Wa, S., Liu, Y., Zhou, X., Sun, P., & Ma, Q. (2021). High-Accuracy Detection of Maize Leaf Diseases CNN Based on Multi-Pathway Activation Function Module. *Remote Sensing*, 13(21), 4218.
4. Gehlot, A. (2020). Integration of MERN Stack for Real-time Agricultural Monitoring Systems. *IEEE 5th International Conference on Computing Communication and Automation (ICCCA)*.
5. Barbedo, J. G. A. (2018). Factors influencing the use of deep learning for plant disease recognition. *Biosystems Engineering*, 172, 84-91.
6. R.D. Engineering College, CSE Department (2026). Project Report: Maize Crop Disease Detection Application.
7. PlantVillage Dataset. (2015). An open access repository of images on plant health. Available online.