WWW.IJRPR.COM

International Journal of Research Publication and Reviews

Journal homepage: www.ijrpr.com ISSN 2582-7421

DERMA AI SKIN DISEASE DETECTION AND HEALTHCARE CHATBOT

KAVIN R[1], NITHERSAN C[2], LOGESHWARAN S[3], Project Guide: NISHMITHA R[4]

Department of Information Technology,
Sri Shakthi Institute of Engineering and Technology,
Coimbatore, Tamil Nadu, India
Email ID: logeshwaransathya27@gmail.com

ABSTRACT :

Skin conditions are common and may require timely assessment, but recognizing them from visual symptoms can be difficult for individuals without dermatological knowledge. This project presents DermaAI, a web-based skin disease classification and healthcare information chatbot. The application combines a React and Vite frontend with a Python Flask backend. Users can upload a skin image and receive the model's top-three predicted classes with scores. The image classification component uses a TensorFlow/Keras MobileNetV2 transfer-learning model and supports 22 class labels represented in the project's class-name file. A separate chatbot module, DermaBot, responds to questions related to symptoms, causes, treatment, prevention, and when to seek medical care. The backend exposes API endpoints for health status, class information, image prediction, and chatbot interaction. The included evaluation report records a test accuracy of 25.68% on 1,546 test images, indicating that the current model requires further improvement before practical clinical use. DermaAI is intended as an educational and preliminary information tool, not as a substitute for diagnosis by a qualified healthcare professional. The project demonstrates the integration of computer vision, web development, and conversational interaction in a single application.

Keywords: Skin Disease Classification, DermaAI, MobileNetV2, Transfer Learning, TensorFlow, Flask, React, Healthcare Chatbot, Image Classification.

1. INTRODUCTION

Skin disorders include a wide range of conditions that can differ in appearance, severity, and required care. A person may find it challenging to identify a condition from a photograph because different diseases can share similar visual features. Access to dermatological evaluation may also be limited by location, availability, or cost. These challenges motivate the development of accessible digital tools that can provide preliminary information and encourage appropriate medical consultation.

Recent developments in deep learning have made image classification possible for many visual tasks. Convolutional neural networks can learn patterns from image data, while transfer learning allows a model pretrained on a large image collection to be adapted to a specific classification task. Web technologies can make such a model accessible through a browser-based interface. In this project, DermaAI combines an image prediction feature and a healthcare information chatbot in one web application.

1.1 Existing challenges

Manual identification of skin conditions requires knowledge of dermatological features and may be affected by image quality, lighting, skin tone, and similarities between conditions. A general image classifier may also make mistakes when trained data are limited or imbalanced. Chatbot responses can be misunderstood as medical advice if the application does not clearly communicate its limitations. Therefore, a responsible system should validate uploaded files, present predictions as estimates, and advise users to consult a healthcare professional for diagnosis or treatment.

1.2 Role of deep learning and web technologies

MobileNetV2 is a convolutional neural network architecture designed to provide efficient image feature extraction. In the project, transfer learning is used by loading MobileNetV2 with ImageNet pretrained weights and adding a task-specific classification head. TensorFlow and Keras provide model construction and training functions. Flask handles backend requests, while React and Vite provide the user interface. The frontend communicates with the backend through API calls for prediction, chatbot responses, and server status.

1.3 Problem statement and objectives

The project addresses the need for a convenient educational application that accepts a skin image, produces a preliminary classification, and provides related general information through a chatbot. The system is not designed to confirm a medical diagnosis. Its current model evaluation also demonstrates that prediction performance is limited and further development is necessary.
The objectives of DermaAI are:

i. To develop a browser-based interface for skin image submission and result viewing.
ii. To build a deep-learning image classifier using MobileNetV2 transfer learning.
iii. To classify submitted images into the 22 labels defined by the project.
iv. To display the top-three predicted classes and associated scores.
v. To provide a chatbot interface for general skin-health questions.
vi. To connect the frontend and backend using Flask API endpoints.
vii. To validate uploaded image types and handle invalid or unreadable files.
viii. To evaluate the model using the available test report and identify limitations.

2. MATERIALS AND METHODS

2.1 Development approach

The project was developed through requirement identification, dataset preparation, model training, backend implementation, frontend development, integration, and evaluation. The image-classification component was implemented in Python using TensorFlow/Keras. The backend provides prediction and chatbot services through Flask. The frontend was developed with React and Vite and contains separate prediction and chatbot views. The final application connects these components through HTTP API requests.

2.2 System architecture

DermaAI follows a client-server architecture. The React frontend provides the user interface and sends requests to the Flask backend. For image prediction, the backend validates the uploaded file, temporarily stores it, invokes the prediction service, formats the top-three output, and removes the temporary image. For chatbot interaction, the backend forwards the message and optional latest predicted disease to the chatbot module. The frontend displays the returned response.

Layer / component	Responsibility
Frontend -- React + Vite	Provides navigation, image upload, preview, prediction display, chatbot interface, and backend status presentation.
API module	Calls backend endpoints for health status, prediction, class list, and chatbot responses.
Flask backend	Receives requests, validates inputs, invokes prediction/chatbot logic, and returns JSON responses.
Prediction service	Loads model/class information, preprocesses images, and produces model predictions.
MobileNetV2 model	Extracts image features and classifies an image into the configured skin-condition labels.
Chatbot module	Generates general skin-health information based on the user message and optional predicted class.

Table 1. Responsibilities of the DermaAI system components.

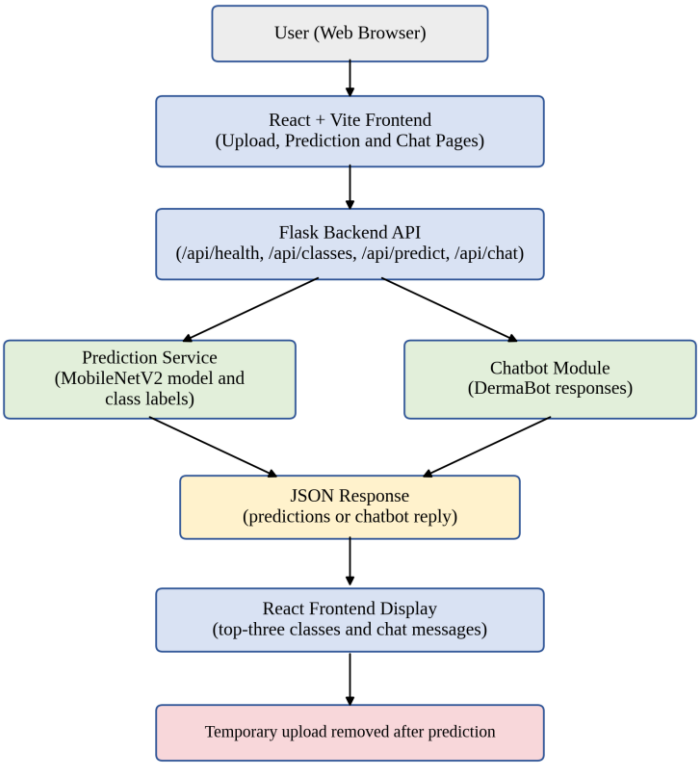


Fig. 1. System Architecture of DermaAI.

2.3 Technology stack and development environment

Technology	Use in the project
Python	Backend and model-related implementation language.
TensorFlow / Keras	Model construction, training, loading, and inference.
MobileNetV2	Pretrained image feature extractor used for transfer learning.
Flask	Backend web framework and API routes.
Flask-CORS	Enables cross-origin frontend API requests.
React.js	Frontend component-based user interface.
Vite	Frontend development server and build tool.
HTML / CSS / JavaScript	Page structure, styling, and browser interactions.

Table 2. Technologies used in the DermaAI project.

The frontend README specifies that the Flask backend runs on port 5000 and the Vite development server runs on port 5173. Vite forwards /api requests to the local Flask service. The project requirements file is included in the uploaded source package; exact dependency versions should be taken from the environment used for the final demonstration.

2.4 Dataset and class labels

The training script reads images from Dataset/Train and Dataset/Test using TensorFlow's image_dataset_from_directory utility. The training directory is divided into training and validation subsets using a 10% validation split and seed 123. The test directory is loaded separately without shuffling. The script uses integer labels and obtains class names from the training dataset directory structure. The uploaded project includes a class_names.json file containing 22 labels. The dataset's original name, source URL, total image count, licensing details, and collection procedure are not included in the supplied project files and are therefore not asserted in this journal.

No.	Class label in project	No.	Class label in project
1	Acne	12	Psoriasis
2	Actinic_Keratosis	13	Rosacea
3	Benign_tumors	14	Seborrh_Keratoses
4	Bullous	15	SkinCancer

No.	Class label in project	No.	Class label in project
5	Candidiasis	16	Sun_Sunlight_Damage
6	DrugEruption	17	Tinea
7	Eczema	18	Unknown_Normal
8	Infestations_Bites	19	Vascular_Tumors
9	Lichen	20	Vasculitis
10	Lupus	21	Vitiligo
11	Moles	22	Warts

Table 3. Class labels configured in the project.

2.5 Dataset preprocessing and augmentation

The training script sets the input image size to 224 × 224 pixels and batch size to 32. TensorFlow loads the images from the class folders and produces integer labels. The model pipeline applies random horizontal and vertical flips, rotation, zoom, and contrast augmentation during training. A rescaling layer maps pixel values to the range expected by MobileNetV2, approximately −1 to 1. The training script also computes class weights from the number of files in each training class to help address differences in class representation. The available source does not specify a complete dataset provenance or a separate manual annotation process.

2.6 Model architecture and training

The project uses MobileNetV2 with include_top=False and ImageNet pretrained weights. The base model is initially frozen, and a task-specific head is added. The head consists of data augmentation, rescaling, the MobileNetV2 feature extractor, global average pooling, dropout with rate 0.3, and a dense softmax layer with one output per class. The model is compiled using the Adam optimizer with an initial learning rate of 0.001 and sparse categorical cross-entropy loss. Accuracy and sparse top-three categorical accuracy are included as training metrics.
The training script defines two training phases: 10 epochs with the pretrained base frozen, followed by 10 epochs of fine-tuning. It sets the final 40 layers as the fine-tuning region. The model is saved as skin_disease_mobilenet.keras, and the class labels are saved to class_names.json. The project also contains a separate CNN model file; the backend prediction service is configured to use the MobileNet model file.

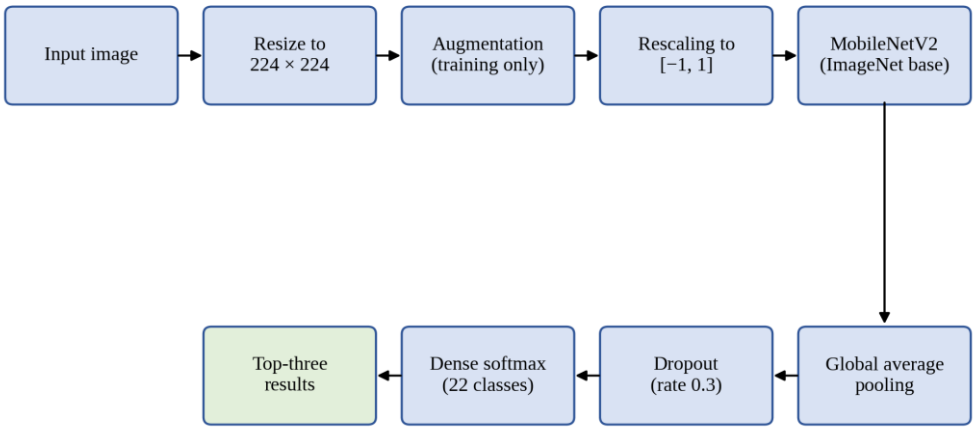


Fig. 2. MobileNetV2 transfer-learning pipeline used in DermaAI.

2.7 Prediction workflow

1. The user opens the prediction page in the React frontend.
2. The user selects a supported image file and submits it.
3. The frontend sends the image to POST /api/predict using the image form field.
4. Flask checks whether a file was selected and whether its extension is PNG, JPG, JPEG, or WEBP.
5. The backend saves the image temporarily using a secure filename and unique identifier.
6. The prediction service preprocesses the image and runs the loaded model.
7. The backend formats the predicted label and the top-three class labels.

8. The temporary uploaded file is deleted, including after prediction errors.
9. The frontend displays the returned predictions and scores.

2.8 Chatbot workflow

The application provides a DermaBot chat interface for general skin-health questions. The frontend sends a JSON request to POST /api/chat containing a message and, when available, the latest predicted disease. The Flask route limits the message to 500 characters and calls the chatbot module. The module returns a response that the frontend displays. The README describes chatbot topics including symptoms, causes, treatment, prevention, and when to see a doctor. The chatbot is a general information feature and should not be interpreted as a clinician or emergency service.

2.9 Backend API design

Endpoint	Method	Purpose
/api/health	GET	Returns backend status and model information.
/api/classes	GET	Returns the configured class names and display labels.
/api/predict	POST	Accepts an image upload and returns prediction results.
/api/chat	POST	Accepts a chatbot message and optional disease context and returns a reply.
/	GET / POST	Legacy HTML page for the original template-based interface.

Table 4. Backend API endpoints of the DermaAI application.

2.10 Input validation and safety

The backend accepts PNG, JPG, JPEG, and WEBP image extensions and sets a maximum request size of 8 MB. The uploaded filename is sanitized, and the file is stored under a generated identifier rather than the original name. The prediction handler removes the temporary file in a finally block. Missing files, unsupported extensions, unreadable images, and oversized requests are handled with error responses. These controls reduce common upload-handling risks but do not by themselves establish that the application is production-secure. The prediction output is informational and must not be used as a medical diagnosis.

2.11 Testing and evaluation method

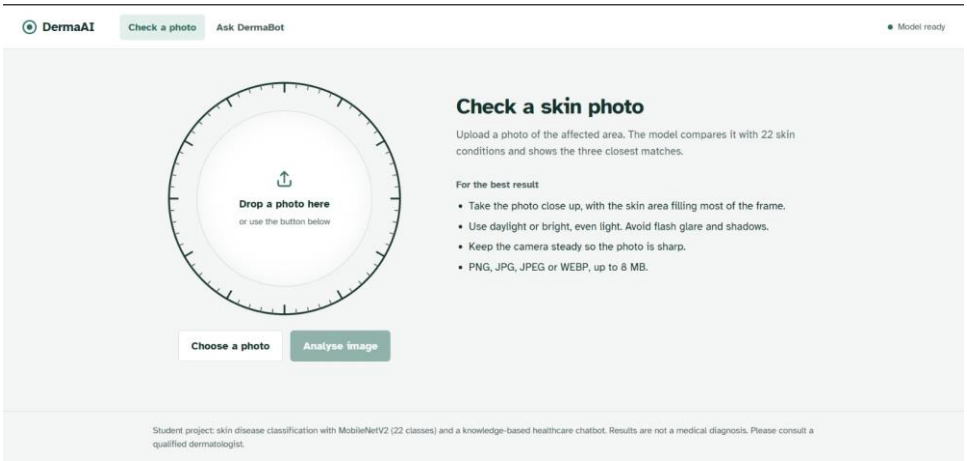
The project includes a model evaluation script, a classification report, a confusion matrix, and chatbot test code. Model evaluation uses the separate test directory and reports precision, recall, F1-score, support, and overall accuracy. Functional checks can cover backend health, class-list retrieval, valid and invalid image submissions, top-three prediction response, chatbot request/response, and frontend-to-backend communication. The journal reports only the measured evaluation result available in the uploaded report; no additional test pass rates or response-time measurements are claimed.

3. RESULTS AND DISCUSSION

3.1 Implemented application features

The supplied project contains a React frontend with prediction and chatbot pages, a Flask backend, a MobileNetV2 model, a prediction service, and a chatbot module. The prediction route returns the top-three classes and scores. The backend also provides class-list and health-check endpoints. The application validates image extensions and request size and removes temporary uploads after processing. These are implementation features verified from the project source; user usability and clinical effectiveness require separate evaluation.

Fig. 3. Prediction page of the DermaAI web interface.



3.2 Model evaluation

The uploaded evaluation_report.txt records an overall test accuracy of 25.68% on 1,546 test images. The report contains 22 class-wise precision, recall, F1-score, and support values. The weighted-average F1-score is 0.19, while macro-average F1-score is 0.11. The results vary substantially across classes: Unknown_Normal has precision 0.60, recall 0.79, and F1-score 0.68; Vitiligo has F1-score 0.42; Benign_tumors and Eczema have recall values of 0.70 and 0.66 respectively. Several classes have zero reported precision and recall. These values indicate that the current model does not yet provide reliable classification across all classes.

Evaluation metric	Reported value
Test images	1,546
Number of classes	22
Overall accuracy	25.68%
Macro average precision	0.11
Macro average recall	0.15
Macro average F1-score	0.11
Weighted average precision	0.17
Weighted average recall	0.26
Weighted average F1-score	0.19

Table 5. Summary of the model evaluation report included in the project.

3.3 Class-wise observations

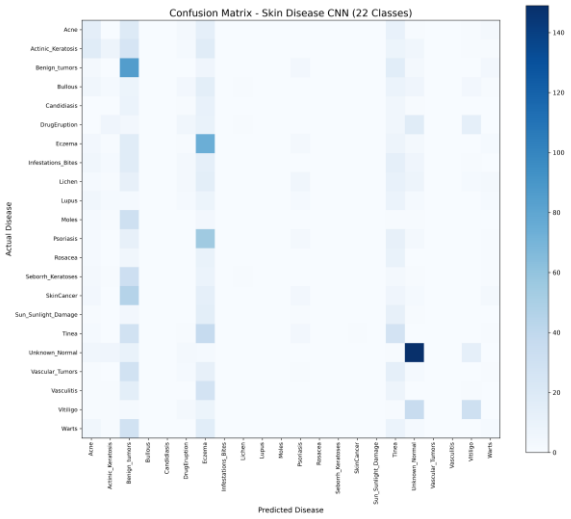
Class	Precision	Recall	F1-score	Support
Unknown_Normal	0.60	0.79	0.68	189
Vitiligo	0.46	0.38	0.42	82
Benign_tumors	0.19	0.70	0.30	121
Eczema	0.19	0.66	0.29	112
Acne	0.17	0.22	0.19	65
Tinea	0.14	0.26	0.18	102
SkinCancer	0.00	0.00	0.00	77
Bullous	0.00	0.00	0.00	55
Candidiasis	0.00	0.00	0.00	27
Rosacea	0.00	0.00	0.00	28

Table 6. Selected class-wise metrics from the supplied classification report. Values are reproduced from the project report; the remaining classes are present in the full evaluation file.

The results suggest that class imbalance, visual overlap between conditions, image variation, and training-data quality may affect performance. Class weights and augmentation are included in the training script, but the available report does not demonstrate that these methods adequately resolve the observed errors. A more complete analysis should inspect the confusion matrix, verify class folder quality, and evaluate a revised model using the same held-out test set.

3.4 Confusion matrix

Fig. 4. Confusion matrix generated by the project evaluation script. The matrix visualizes correct and incorrect class predictions on the test set.



3.5 Functional test scenarios

Test ID	Module	Test scenario	Expected / observed handling
F1	Health API	Request /api/health	Returns status and model information.
F2	Class API	Request /api/classes	Returns configured class labels.
F3	Image upload	Submit supported image	Backend processes image and returns top-three predictions when inference succeeds.
F4	Validation	Submit without image	Returns a clear missing-image error.
F5	Validation	Submit unsupported extension	Rejects file type; accepted types are PNG, JPG, JPEG, WEBP.
F6	Upload limit	Submit request exceeding configured limit	Flask request-size limit is 8 MB.
F7	Chatbot	Send message to /api/chat	Returns chatbot reply from chatbot module.
F8	Cleanup	Complete or fail prediction	Temporary uploaded file is removed in cleanup block.

Table 7. Source-based functional scenarios. These are derived from implemented route and validation logic; they are not presented as a record of a completed formal test campaign.

3.6 Discussion and limitations

DermaAI demonstrates an integrated application that connects image classification with a general healthcare chatbot. The frontend and backend are separated, making it possible to update the interface, prediction service, or chatbot independently. The model evaluation, however, shows limited classification performance. An overall accuracy of 25.68% and a weighted F1-score of 0.19 are insufficient to support dependable disease identification. In particular, a model prediction could incorrectly reassure a user or suggest an inappropriate condition. The application must therefore clearly identify outputs as model estimates and encourage professional assessment.

The available project files do not include the original dataset name, dataset license, exact dataset size before splitting, detailed hardware, complete training logs, or verified deployment results. These details should be added only after confirmation from the project team. The chatbot's knowledge and safety behaviour also require systematic review by qualified healthcare personnel before any real-world medical use.

4. CONCLUSION

DermaAI is a web-based skin disease classification and healthcare information chatbot developed using React, Vite, Flask, TensorFlow/Keras, and MobileNetV2. The application allows users to submit a skin image, view the top-three predicted classes, and ask DermaBot general questions related to skin health. The backend provides API routes for health status, class labels, image prediction, and chatbot communication. Input validation and temporary-file cleanup are included in the image upload workflow.

The available model evaluation reports 25.68% test accuracy across 1,546 images and 22 classes. This result demonstrates that the current classifier requires substantial improvement and should not be used for clinical diagnosis. Future work includes reviewing dataset provenance and quality, improving class balance, expanding and validating training examples, tuning the model, evaluating per-class performance, and conducting repeated tests on a held-out dataset. The chatbot should be reviewed for medical safety, and the application should be assessed for privacy, usability, and reliability. With further development and validation, DermaAI may serve as an educational demonstration of computer vision and conversational interfaces in healthcare.

5. ACKNOWLEDGMENTS

The authors express their sincere gratitude to Sri Shakthi Institute of Engineering and Technology for providing the academic environment and facilities required for this project. The authors thank the project guide, NISHMITHA R, for guidance, encouragement, and support throughout the development of DermaAI. The authors also acknowledge the faculty members and contributors who supported the learning and implementation process.

6. REFERENCES

1. M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L.-C. Chen, "MobileNetV2: Inverted Residuals and Linear Bottlenecks," in Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2018.
2. F. Chollet, "Xception: Deep Learning with Depthwise Separable Convolutions," in Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2017.
3. R. Pangti et al., "A machine learning-based, decision support, mobile phone application for diagnosis of common dermatological diseases," Journal of the European Academy of Dermatology and Venereology, vol. 35, no. 2, pp. 536–545, 2021. doi: 10.1111/jdv.16967.
4. TensorFlow, "Image classification," TensorFlow Documentation. Available: https://www.tensorflow.org/tutorials/images/classification
5. TensorFlow, "MobileNetV2," TensorFlow API Documentation. Available: https://www.tensorflow.org/api_docs/python/tf/keras/applications/MobileNetV2
6. Flask, "Flask Documentation." Available: https://flask.palletsprojects.com/

7. React, "React Documentation." Available: https://react.dev/
8. Vite, "Vite Documentation." Available: https://vite.dev/
9. Project source code and evaluation artifacts, "DermaAI: Skin Disease Detection and Healthcare Chatbot," supplied project archive, EE sem 3.zip. Includes training script, prediction service, Flask routes, React frontend, class_names.json, evaluation_report.txt, and confusion_matrix.png.