Deep Learning Fundus Image Analysis For Early Detection of Diabetic Retinopathy.

A Project Report

Submitted in MOOC Course Completion by SmartBridge

Bachelor of Technology — Artificial Intelligence & Machine Learning

Submitted By:

Maithilee Vijay Kale

PRN: 2022011041022

D.Y. Patil Agricultural and Technical University, Talsande.

Academic Year: 2025 – 2026

Tech Stack:

• Flask • TensorFlow 2.3 • InceptionV3 • IBM Granite LLM • HuggingFace API • OpenCV • Python

1. ABSTRACT

This project presents an end-to-end deep learning web application for automated detection and classification of Diabetic Retinopathy (DR) from retinal fundus images. Using InceptionV3 Convolutional Neural Network with Transfer Learning, the system classifies images into five severity classes: No DR, Mild DR, Moderate DR, Severe DR, and Proliferative DR. A two-phase training strategy — frozen feature extraction followed by fine-tuning — was applied on a 5-class labeled dataset. The model achieves 80.52% test accuracy (734 samples, weighted F1 = 0.78). IBM Granite 3.1 LLM via HuggingFace API generates clinical explanations for each prediction. The full system is deployed as a Flask web application with bcrypt authentication, image upload, real-time inference, and JSON-based prediction logging.

2. INTRODUCTION & PROBLEM STATEMENT

Diabetic Retinopathy is a leading cause of preventable blindness, affecting millions of diabetic patients who remain undiagnosed until irreversible damage occurs. Manual screening is resource-intensive and geographically constrained, creating a critical gap in early detection — particularly in developing regions. This project addresses the need for an automated, scalable, web-deployable screening tool by implementing a deep learning-based classification system that identifies DR severity from retinal fundus images, provides AI-generated clinical explanations, and logs all predictions for audit and follow-up purposes.

3. OBJECTIVES

- Train an InceptionV3-based CNN using two-phase transfer learning for 5-class DR grading.
- Apply CLAHE contrast enhancement and Keras augmentation (rotation, zoom, flip) for better generalization.
- Evaluate the model using accuracy, precision, recall, F1-score, and confusion matrix metrics.
- Develop a Flask web app with bcrypt authentication, image upload, real-time inference, and prediction display.
- Integrate IBM Granite 3.1 LLM via HuggingFace API for AI-generated clinical explanations.
- Log all prediction records (user, image, class, confidence, explanation, timestamp) to a JSON database.
- Package the application for cloud deployment via Procfile and requirements.txt.

4. SYSTEM ARCHITECTURE & TECH STACK

The system is organized into four layers: (1) Data Layer — class-labeled fundus image directories, users.json and database.json flat files; (2) Processing Layer — CLAHE preprocessing, Keras augmentation, InceptionV3 model training and evaluation; (3) Application Layer — Flask web server with session management, image upload, model inference, and LLM API calls; (4) Presentation Layer — Jinja2 HTML templates for register, login, dashboard, and prediction views.

Component	Technology / Detail
Deep Learning Framework	TensorFlow 2.3.2 / Keras
Model Architecture	InceptionV3 (Transfer Learning, ImageNet weights)
Image Preprocessing	OpenCV — CLAHE (LAB space), resize 299×299
Web Framework	Flask (Python) — Jinja2 templating
Authentication	Werkzeug scrypt/bcrypt password hashing
LLM Integration	IBM Granite 3.1 via HuggingFace Inference API
Data Storage	JSON flat files (users.json, database.json)
Evaluation	Scikit-learn — accuracy, F1, confusion matrix
Deployment	Gunicorn + Procfile (Render / Railway / Heroku)

5. DATASET & PREPROCESSING

5.1 Dataset

The dataset comprises retinal fundus images organized in ImageNet-compatible directory format under data/training/ and data/testing/. Training: 2,931 images across 5 classes. Testing: 734 images (361 No DR, 74 Mild, 200 Moderate, 39 Severe, 60 Proliferative). The dataset exhibits significant class imbalance, with No DR comprising ~49% of test samples and Severe DR only ~5%.

5.2 Preprocessing Pipeline

- BGR→RGB conversion using OpenCV cv2.cvtColor().
- Spatial resize to 299×299 pixels (InceptionV3 requirement).
- CLAHE contrast enhancement applied in LAB color space (clipLimit=2.0, tileGridSize=8×8) to enhance retinal microstructure visibility.
- Runtime: Keras image.load_img(), pixel normalization ÷255, np.expand_dims() for batch dimension.

5.3 Data Augmentation (during training)

- Rotation ±25°, Zoom ±20%, Width/Height shift ±20%, Horizontal flip, Validation split 20%.

6. MODEL BUILDING & TRAINING

6.1 Architecture

The model is built on InceptionV3 (include_top=False, weights='imagenet', input 299×299×3). The custom classification head adds: GlobalAveragePooling2D → BatchNormalization → Dense(512, ReLU) → Dropout(0.5) → Dense(5, Softmax). Loss: Categorical Crossentropy. Optimizer: Adam.

6.2 Two-Phase Training Strategy

Phase 1 (Feature Extraction): InceptionV3 frozen (trainable=False), Adam lr=1e-3, 10 epochs — trains only the custom head. Phase 2 (Fine-Tuning): Full model unfrozen (trainable=True), Adam lr=1e-5, up to 25

epochs — adapts InceptionV3 filters to retinal image features. Callbacks: EarlyStopping (patience=5, restore_best_weights) + ReduceLROnPlateau (factor=0.3, patience=3).

Training Screenshots — Phase 1 (Epochs 1–10) running in the dr_detection conda environment on CPU (TensorFlow 2.3, AVX2 optimized):

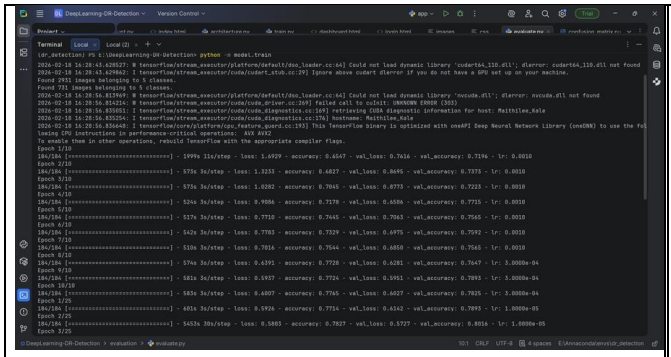

Figure 1: *Phase 1 training — Epochs 1–10 (lr=1e-3)*

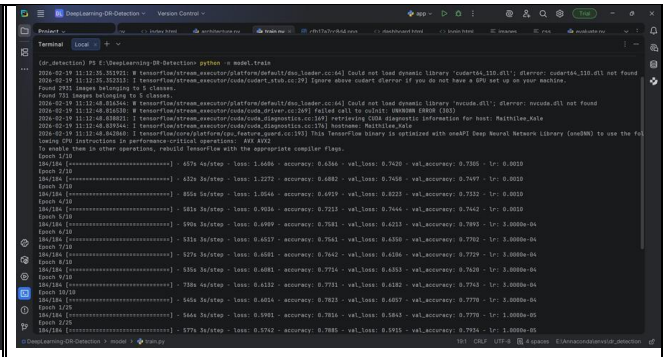

Figure 2: *Phase 2 start — Fine-tuning begins (lr=1e-5)*

Training completion — Model saved to model/saved_model.h5 with training plots exported to evaluation/ directory:

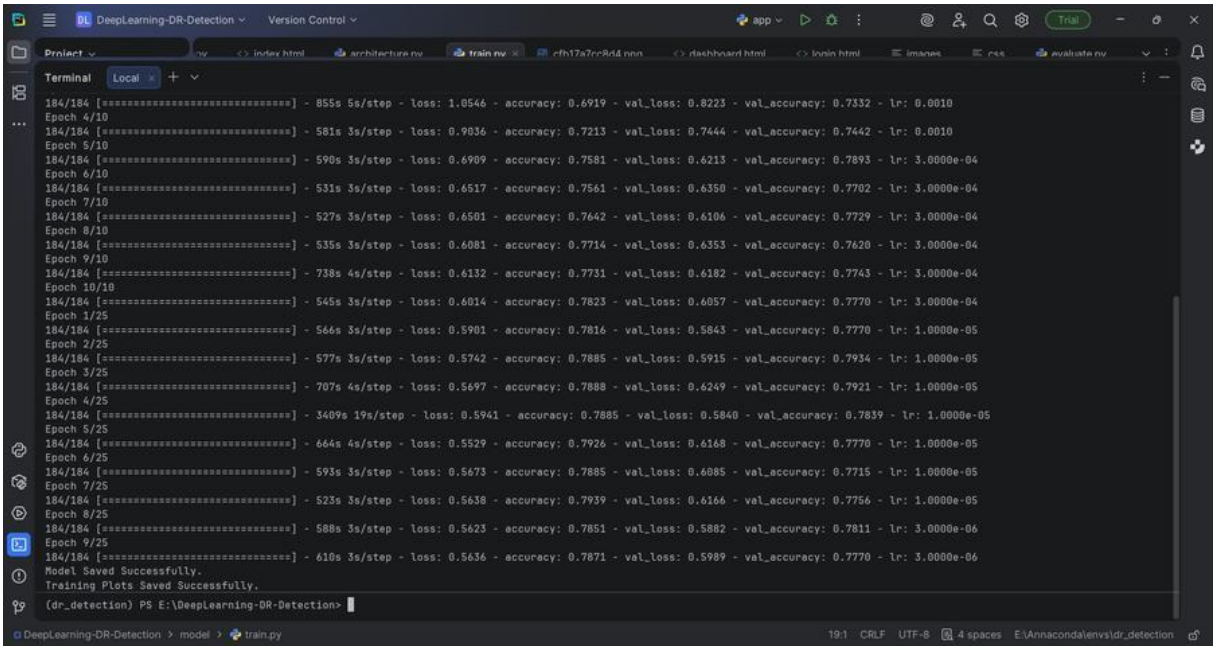

Figure 3: *Training complete — 'Model Saved Successfully' and 'Training Plots Saved Successfully'*

7. EVALUATION & RESULTS

7.1 Evaluation Methodology

Evaluation is performed via evaluation/evaluate.py. The trained model is loaded and applied to the test dataset (734 images, shuffle=False). Metrics computed using Scikit-learn: accuracy_score, classification_report, confusion_matrix. Outputs saved to evaluation/accuracy.txt, classification_report.txt, confusion_matrix.png, accuracy_plot.png, and loss_plot.png.

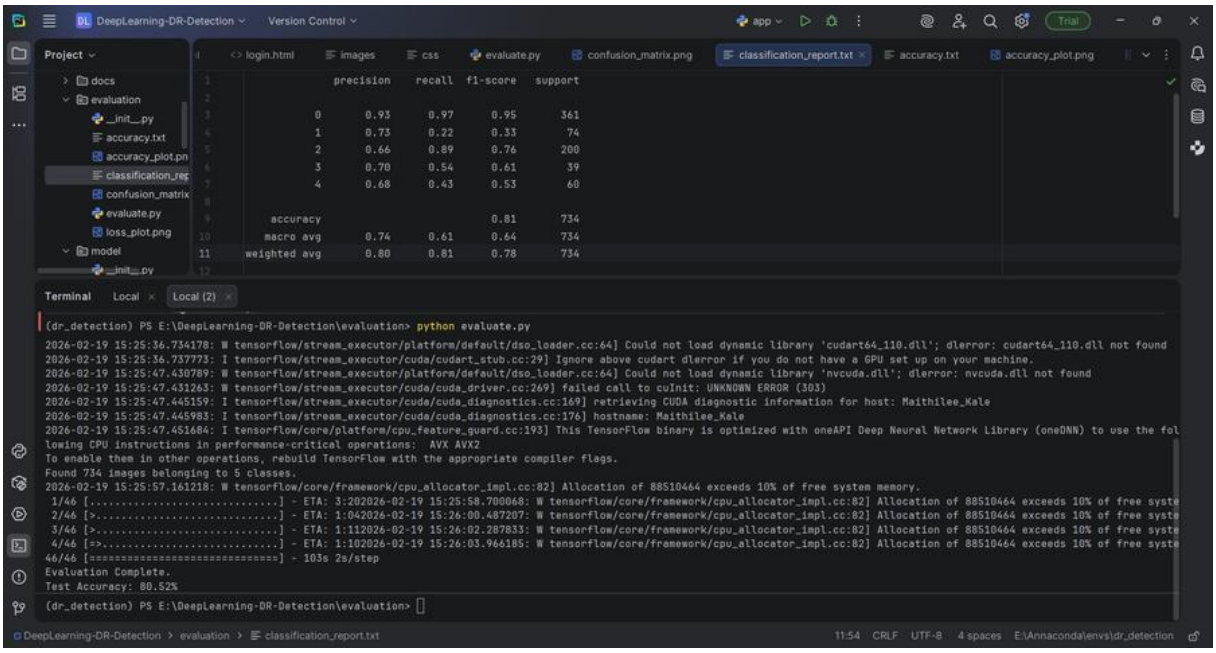

Figure 4: *Evaluation run — Test Accuracy: 80.52% with classification report showing per-class metrics*

7.2 Performance Metrics

Class	Label	Prec.	Recall	F1	Support
0	No DR	0.93	0.97	0.95	361
1	Mild DR	0.73	0.22	0.33	74
2	Moderate DR	0.66	0.89	0.76	200
3	Severe DR	0.70	0.54	0.61	39
4	Proliferative	0.68	0.43	0.53	60
—	**Weighted Avg**	**0.80**	**0.81**	**0.78**	**734**

Metric	Value
Test Accuracy	80.52%
Weighted F1-Score	0.78
Best Class — No DR (Class 0)	F1 = 0.95, Recall = 0.97
Weakest Class — Mild DR (Class 1)	F1 = 0.33 (class imbalance)
Total Test Samples	734 images across 5 classes

Training Epochs (actual)	~19 (EarlyStopping)

7.3 Key Observations

- No DR (Class 0): Precision 0.93, Recall 0.97 — strongest performance. Critical for minimizing false positives in screening.
- Mild DR (Class 1): Recall 0.22 — most mis-classified as Moderate DR due to subtle visual similarity and low sample count (74).
- Moderate DR (Class 2): Recall 0.89 — best among pathological classes, reflecting strong training representation (200 samples).
- Fine-tuning (Phase 2) improved val_accuracy from ~78% to ~80%, confirming the benefit of differential learning rates.
- Training/validation curves show no significant overfitting — both converge near 0.78–0.80 accuracy with consistent loss reduction.

8. WEB APPLICATION & WORKFLOW

8.1 Flask Routes

- GET /login — Login form with session guard, check_password_hash() validation
- GET/POST /register — Registration with bcrypt hashing, duplicate-username detection
- GET /dashboard — Authenticated upload form (session guard redirects unauthenticated users)
- POST /predict — Image upload → preprocess → InceptionV3 inference → LLM explanation → DB log → render result
- GET /logout — session.pop('user'), redirect to /login

8.2 Application Screenshots

Figure 5 shows the Login page at 127.0.0.1:5000/login with session guard message. Figure 6 shows the Dashboard for authenticated user 'maithilee'.

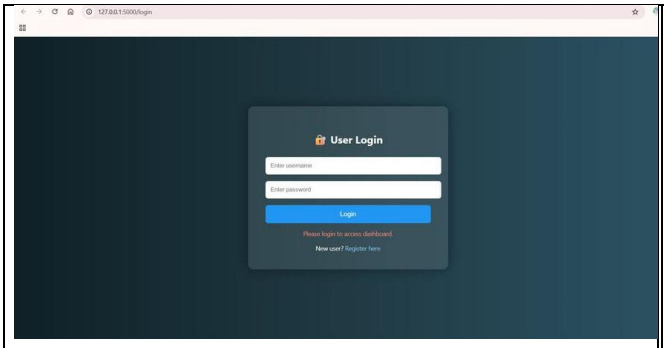

Figure 5: *Login page — 127.0.0.1:5000/login*

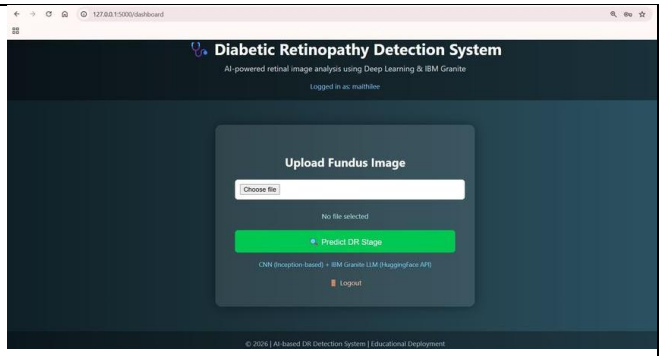

Figure 6: *Dashboard — Upload Fundus Image (logged in as: maithilee)*

Figures 7 and 8 show the Prediction Result page at 127.0.0.1:5000/predict — displaying the uploaded retinal fundus image, the predicted DR stage badge (No DR — green), 52.92% confidence score, and the full AI Clinical Explanation generated via IBM Granite LLM:

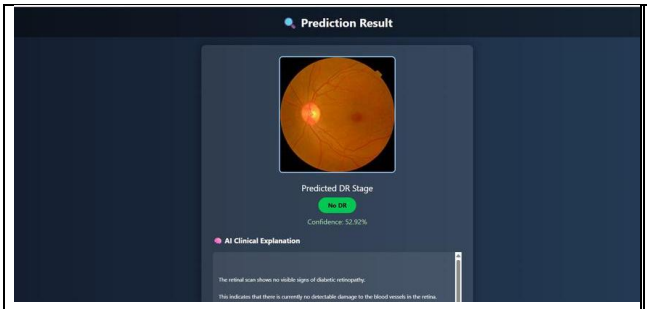

Figure 7: *Prediction result — No DR badge, 52.92% confidence*

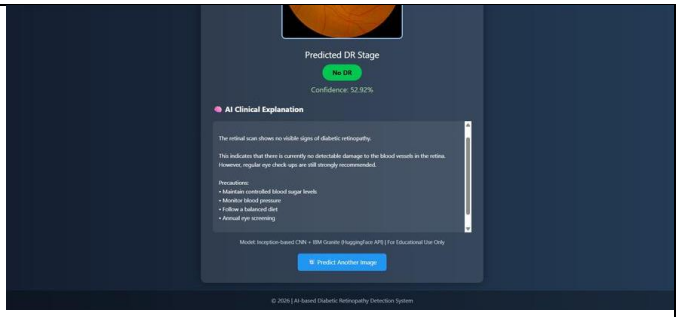

Figure 8: *AI Clinical Explanation with precautions (No DR)*

8.3 Prediction Pipeline

- Image uploaded via multipart/form-data POST to /predict.
- File type validated by allowed_file() — accepts .jpg, .jpeg, .png only.
- UUID-prefixed filename generated (uuid.uuid4()) for collision-free storage in static/uploads/.
- preprocess_image(): load_img(299×299) → img_to_array() → ÷255.0 → expand_dims().
- model.predict() produces 5-element softmax vector. np.argmax() → class, np.max() → confidence.
- generate_explanation() queries IBM Granite 3.1 API; falls back to pre-defined clinical text on failure.
- save_to_database() appends full record to database.json (14 production records confirmed).

9. DATABASE, ADVANTAGES & FUTURE SCOPE

9.1 Database Design

Two JSON flat-file stores are used. users.json stores username + scrypt-hashed password per registered user (currently 2 users: Maithilee Kale, Riya). database.json stores each prediction record: { user, image (UUID filename), prediction, confidence, explanation, timestamp } — currently 14 live records from multiple sessions.

9.2 Advantages

- 80.52% accuracy via Transfer Learning without training from scratch on limited medical data.
- CLAHE preprocessing enhances retinal vessel contrast for improved lesion detection.
- IBM Granite LLM integration provides plain-language clinical context beyond numeric class labels.
- Graceful API fallback — pre-defined clinical explanations served when HuggingFace API is unavailable.
- Secure scrypt/bcrypt password hashing protects credentials even if users.json is compromised.
- Complete audit trail — every prediction timestamped with user, image, class, confidence, and explanation.
- Cloud-ready deployment with Procfile + Gunicorn for Render, Railway, and Heroku.

9.3 Limitations

- Mild DR recall of 0.22 — minority class under-detection due to class imbalance and visual similarity with Moderate DR.
- JSON flat-file storage not suitable for high-concurrency environments (race conditions on concurrent writes).

- No Grad-CAM visualization — no heatmap showing which retinal regions influenced the prediction.
- HuggingFace API token exposed in source code — must be moved to environment variable for production.

9.4 Future Scope

- Add Grad-CAM / LIME heatmap visualization to highlight lesion regions driving the prediction.
- Implement class-weighted training to improve Mild DR (Class 1) and Severe DR (Class 3) recall.
- Migrate to PostgreSQL or MongoDB for concurrent multi-user database access.
- Build a prediction history dashboard using existing database.json records.
- Explore EfficientNetB4 / ResNet50 ensemble to further improve overall accuracy.

10. CONCLUSION & REFERENCES

10.1 Conclusion

This project successfully demonstrates a complete end-to-end deep learning pipeline for Diabetic Retinopathy detection — from raw fundus image preprocessing through model training, evaluation, and deployment as a production-ready Flask web application. The system achieves 80.52% test accuracy across five DR severity classes using InceptionV3 Transfer Learning with CLAHE-enhanced preprocessing. Integration of IBM Granite 3.1 LLM provides clinical context beyond numeric predictions, while bcrypt authentication and JSON-based audit logging ensure secure, accountable operation. The live application has processed 14 real predictions across two registered users, confirming the full pipeline functions correctly end-to-end.

10.2 References

[1] Gulshan et al. (2016). Development and Validation of a Deep Learning Algorithm for Detection of DR in Retinal Fundus Photographs. JAMA, 316(22).

[2] Szegedy et al. (2016). Rethinking the Inception Architecture for Computer Vision. CVPR, pp. 2818–2826.

[3] Pan & Yang (2010). A Survey on Transfer Learning. IEEE TKDE, 22(10), 1345–1359.

[4] Shorten & Khoshgoftaar (2019). A Survey on Image Data Augmentation for Deep Learning. J. Big Data, 6(60).

[5] Zuiderveld (1994). Contrast Limited Adaptive Histogram Equalization. Graphics Gems IV, pp. 474–485.

[6] IBM Research (2024). Granite Code Models: A Family of Open Foundation Models. arXiv preprint.

[7] Grinberg, M. (2018). Flask Web Development (2nd ed.). O'Reilly Media.

[8] Pedregosa et al. (2011). Scikit-learn: Machine Learning in Python. JMLR, 12, 2825–2830.

— END OF REPORT —

Maithilee Kale | B.Tech AI & ML | 2025–2026