Project: Diabetic Retinopathy Detection using AI. aka, “DiaRet-Scan AI”

Topic: Computer Vision, Deep Learning, AI/ML, Diabetic Retinopathy.
Github: Link

Problem Statement: Diabetic Retinopathy (DR) is a leading cause of blindness. Early detection is crucial but manual screening by ophthalmologists is time-consuming and prone to human error.

Objective: To build an AI system that can detect and identify Diabetic Retinopathy and its severity, by analysing retinal fundus images.

And we classify them into 5 stages of severity:

1. No DR (Healthy)
2. Mild NPDR
3. Moderate NPDR
4. Severe NPDR
5. Proliferative DR

Dataset: APTOS 2019 Blindness Detection Dataset.
It is a gold standard benchmark for Diabetic Retinopathy, allowing for direct comparison with good results. It consists of 3,662 high-resolution retinal fundus images captured under various imaging conditions.

Class Composition & Imbalance: The dataset is highly imbalanced, heavily skewed towards healthy eyes. This poses a challenge for training, which I addressed by using weighted metrics to evaluate performance accurately.

- **Class 0 (No DR):** ~1,805 images (Majority)
- **Class 1 (Mild):** ~370 images
- **Class 2 (Moderate):** ~999 images
- **Class 3 (Severe):** ~193 images (Minority)
- **Class 4 (Proliferative):** ~295 images

Architecture: ResNet50

Tools and Technologies: Python, PyTorch, OpenCV, Google Colab, ngrok, Streamlit, etc.

Methodology: (i) Computer Vision.

1. **Data Preprocessing:** To ensure the lighting and reduce the noise, and standardize inputs so that the AI only focuses on the retina, we used a technique called "*Ben Graham's Method*" (Color Space correction & Gaussian Blur). *Automatic Cropping & Standardizing Lighting (Ben Graham's Method)*: `New Image = Original - Blurred + 128`.
   Why Classical Processing? I chose this classical computer vision technique over a learnable Neural Network layer for three reasons:

   - **Domain Knowledge:** We know that lighting variation is "noise." Mathematically subtracting the average luminosity (the blur) is a guaranteed way to normalize this without requiring the model to "learn" lighting invariance from scratch.
   - **Consistency:** This operation is deterministic. It works exactly the same way on every single image, ensuring the input to the ResNet is always standardized.
   - **Efficiency:** It is computationally inexpensive (milliseconds) compared to adding extra convolutional layers, keeping the inference time low for the web app.

2. **Model Architecture:** The project employed **Transfer Learning** to efficiently train a model despite limited data and time**,** with **ResNet50** (Residual Network with 50 layers) a Convolutional Neural Network (CNN) pre-trained on **ImageNet**, served as the base model.
   Architecture Choice: ResNet-50 was selected because it offers an optimal balance between accuracy and computational speed. While larger models (like ResNet-101 or EfficientNet-B7) might offer marginal accuracy gains, they are significantly slower to run in a real-time web application.

   Transfer Learning Strategy:

   - **Frozen Layers:** The distinct convolutional blocks (Layers 1–49) were frozen. Having been pre-trained on ImageNet, these layers serve as powerful feature extractors for basic visual elements such as edges, curves, and textures.

- **Fine-Tuned Layers:** The original 1000-class classification head was removed and replaced with a custom **Fully Connected (FC) Linear Layer** with 5 outputs. Only this final layer was trained using backpropagation to map the extracted features to the specific diabetic retinopathy severity stages.

3. **The Training Strategy:** To teach the model to distinguish between the 5 classes by minimizing error.

   Loss Function (CrossEntropyLoss): Punishes if the model predicts wrong with high confidence.

   Optimizer (Adam): Dynamically adjusts the learning rate for efficient training.

   Hardware: Used **NVIDIA GPUs** (via Google Colab/CUDA) for faster computation.

   Training Configuration & Metrics:

   - **Hyperparameters:** The model was trained for **15 epochs** with a learning rate of **1e-4** and a batch size of **32**. The low learning rate was selected to ensure stable convergence without destroying the pre-trained weights.

   Performance Metrics: To ensure the model is not overfitting, I monitored both Training and Validation loss. The validation accuracy stabilized around **91.8%**, closely tracking the training accuracy, which indicates good generalization.

   - **Training Accuracy:** 98.2%
   - **Validation Accuracy:** 91.8%
   - **Test Accuracy:** 86.0%
   - **F1-Score:** 0.86 (A crucial metric given the class imbalance)
   - **AUC Score:** 0.94 (Demonstrating strong separability between classes)

4. **Inference System (The Real-Time Logic):** To make predictions on new, unseen images uploaded by users.

   Tensor Transformation: User images (Numpy arrays) are converted to PyTorch **Tensors**. Pixel values (0-255) are normalized to a 0-1 range (floating point).

   Forward Pass: The image passes through the 50 layers of the network.

   Output: "Logits" (raw numerical scores, e.g., [2.1, -0.5, 4.8, 1.2, -1.0]).

   Probabilistic Conversion (Softmax): We apply the **Softmax function** to convert logits into percentages that sum to 100%. *Result:* [5%, 1%, 85%, 8%, 1%]. Classification is done by **Argmax,** the index with the highest percentage is selected as the predicted class.

   Deployment Architecture: The system operates using **Online Inference**. When a user uploads an image to the Streamlit frontend, it is sent securely via ngrok to the backend server (Google Colab GPU). The neural network processes the image in real-time and returns the prediction JSON to the UI. This "Cloud/Server-side" architecture allows us to run a heavy ResNet-50 model on any device, including mobile phones, without requiring local client-side compute power.ui

   AI Analysis View: The "AI Analysis View" displayed in the app is the visualization of the **Ben Graham Preprocessing** step. It is generated using the OpenCV pipeline described in the methodology, not a separate neural network. This provides the doctor with transparency, showing exactly what the AI "sees"—the high-contrast retinal structure with lighting noise removed.

   Clinical Assessment Text: The text-based assessment is generated via a **deterministic mapping logic**. Once the ResNet-50 predicts a class (e.g., "Moderate NPDR"), the system queries a medical knowledge base dictionary to retrieve the corresponding clinical description and recommendation, ensuring the medical advice is standard and accurate.

(ii) Deployment & Application Development: To create a user-friendly product interface for doctors and patients, using **Streamlit** (Frontend) + **ngrok** (Tunneling).

1. **User Interface (UI):** Designed a custom modern dark theme webpage using CSS injection.

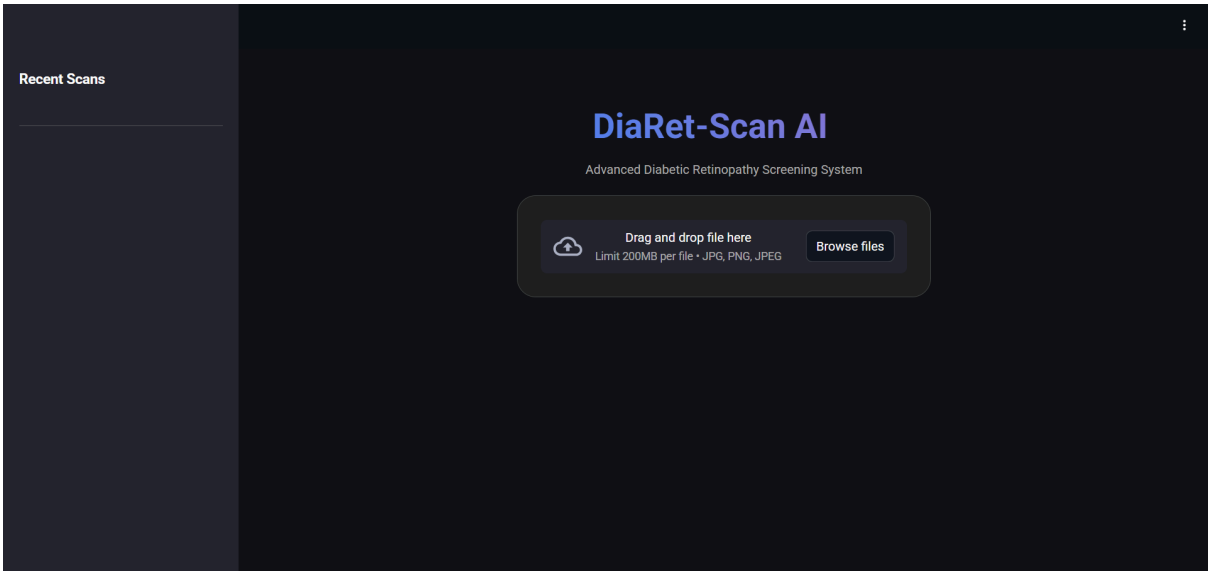
   Layout: Centered upload area, side-by-side comparison of "Original" vs "AI Enhanced" views.

   Session State Management: Implemented a history feature that stores the last 5 scans in RAM (`st.session_state`). Allows users to toggle between previous results instantly without re-processing.

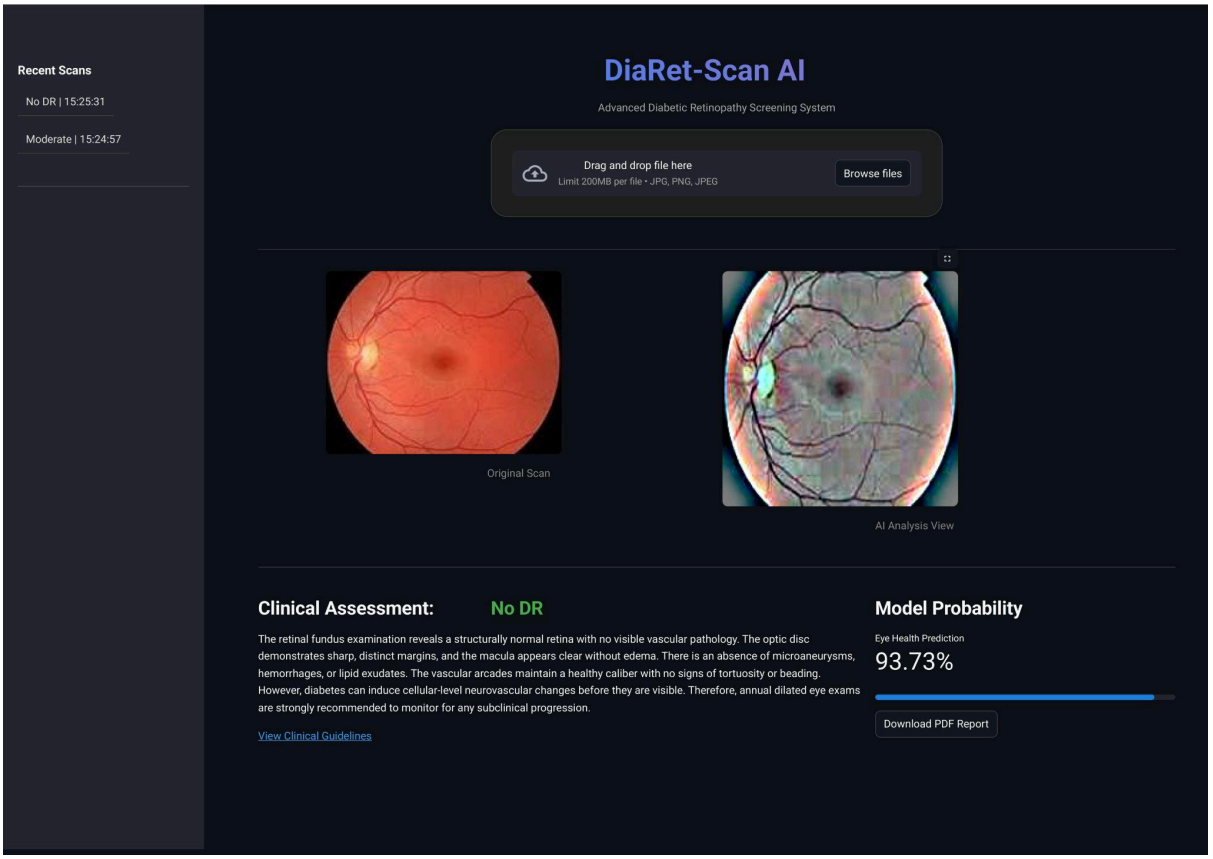
2. **Report Generation:** Integrated **ReportLab** to generate dynamic PDF reports. The system draws the patient's ID, timestamp, diagnosis, probability score, and both images onto a standardized medical template.

Examples:

Before Input:



Output Result:



Conclusion:

This project successfully bridges the gap between raw Deep Learning code and a usable medical tool. By combining advanced Computer Vision techniques (Ben Graham's method) with the power of Transfer Learning (ResNet50), we created a system that is robust to lighting changes and highly accurate in grading Diabetic Retinopathy.