Automated Classification of Retinal Fundus Images for Diabetic Detection Using Machine Learning and Hyperparameter Optimization

Submitted by

Name	Roll Number
Aditya Jyosyula	3122225001006
K. Harish	3122225001036
Jagrath Mithun S	3122225001044
Janeshvar Sivakumar	3122225001047

Department of Computer Science and Engineering

SSN College of Engineering

April 8, 2025

Abstract

This project focuses on the classification of retinal fundus images for the detection of diabetes using machine learning models. Dimensionality reduction techniques and hyperparameter optimization were applied to enhance model performance. The project uses the Retina Blood Vessel dataset, specifically focusing on diabetic cases. Results show that optimized models significantly outperform their non-optimized counterparts.

1. Introduction

Diabetes is one of the leading causes of preventable blindness worldwide. Retinal fundus images provide critical information for the early detection of diabetic retinopathy. Manual inspection is time-consuming and requires expertise. Automating this task using machine learning can improve diagnostic efficiency and accessibility.

1.1. Problem Statement

To develop an optimized machine learning model capable of accurately classifying retinal fundus images as diabetic or non-diabetic using dimensionality reduction and hyperparameter tuning techniques.

2. Literature Survey

Several studies have explored diabetic retinopathy detection using deep learning and traditional machine learning methods. CNNs have shown strong performance in image-based tasks. Recent works emphasize the importance of preprocessing, feature extraction, and parameter tuning for improved model accuracy.

Miri et al. (2017) mention that inhomogeneous contrast and illumination, caused by imaging processes and biological characteristics, are significant challenges in retinal image analysis. They point out that color variations within and across images arise from factors like hemoglobin oxygen saturation, aging, cataract development, flash intensity and spectrum, camera distortions, flash artifacts, and focus.

Zhang et al. (2023) focuses on a particular preprocessing method called adaptive contrast enhancement (ACE) applied to the RITE dataset. They explain that ACE addresses the issues of uneven lighting, low overall illumination, and low contrast between blood vessels and the background, which are common in retinal fundus images and can negatively impact classification accuracy.

Ajaz et al. specifically focus on the relationship between retinal vessel geometrical features and the incidence and progression of DME. They found that Average Branching Angle (ABA) was the only parameter that exhibited a monotonic increase with disease severity.

Guo et al. investigate the association between retinal information and CVD in patients with type 2 diabetes, independent of traditional cardiovascular risk factors.

3. Proposed System

The proposed system includes a pipeline consisting of data preprocessing, feature engineering using PCA, model training, hyperparameter tuning via Grid Search and Random Search, and performance evaluation.

3.1. System Architecture Diagram

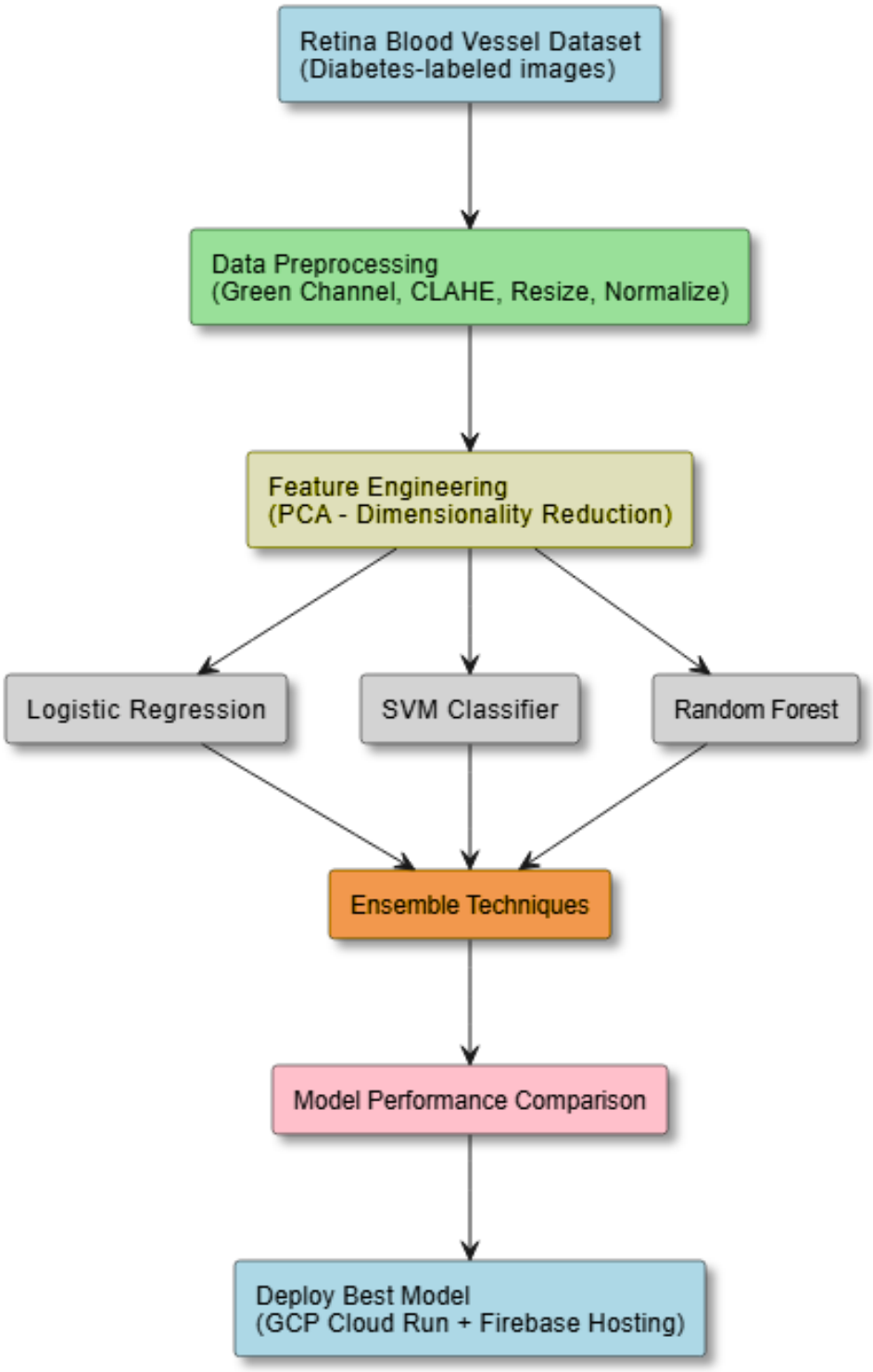


Figure 1: System Architecture Diagram

4. Implementation and Experiments

4.1. Development Environment

- Python 3.10
- Libraries: scikit-learn, pandas, matplotlib, seaborn, numpy, OpenCV
- Jupyter Notebook / VS Code
- Cloud Deployment: GCP (Cloud Run + Firebase Hosting)

4.2. Dataset Description

Source: Retina Blood Vessel Dataset on Kaggle
Focus: Only diabetes-labeled retinal images were used. The dataset contains high-resolution fundus images.

5. Implementation and Experiments

5.1. Development Environment

- Python 3.10
- Libraries: scikit-learn, pandas, matplotlib, seaborn, numpy, OpenCV
- Jupyter Notebook / VS Code
- Cloud Deployment: GCP (Cloud Run + Firebase Hosting)

5.2. Dataset Description

Source: Retina Blood Vessel Dataset on Kaggle
Focus: Only diabetes-labeled retinal images were used. The dataset contains high-resolution fundus images.

5.3. Implementation Steps

1. Data Collection:
Images were extracted from the Kaggle dataset. All diabetic-labeled images were filtered and selected for this binary classification task.

2. Preprocessing:
Each image underwent several enhancement techniques:

- Green channel extraction (retinal vessels are clearer in this channel)
- Contrast Limited Adaptive Histogram Equalization (CLAHE)
- Image resizing to 224×224
- Normalization to [0, 1] scale

3. Feature Engineering:
Flattened image pixels formed high-dimensional feature vectors (over 50,000 features per image). PCA was applied to reduce dimensionality while retaining essential variance. The number of components was chosen based on explained variance threshold (e.g., 95%).

4. Model Training:
We trained the following ML models:

- Logistic Regression: A linear baseline model
- SVM: Effective in high-dimensional space
- Random Forest: Captures non-linear relations with ensemble learning

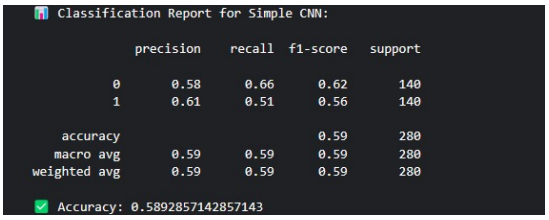
5. Hyperparameter Optimization:
We used:

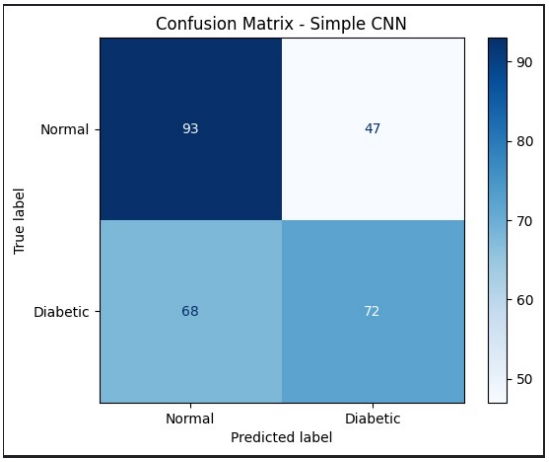
- Grid Search: Exhaustive parameter tuning
- Randomized Search: Faster alternative with good results

Both used cross-validation and prioritized F1-score.

6. Model Evaluation:
We evaluated the models using:

- Accuracy, Precision, Recall, F1-score
- Confusion Matrix
- Cross-validation statistics

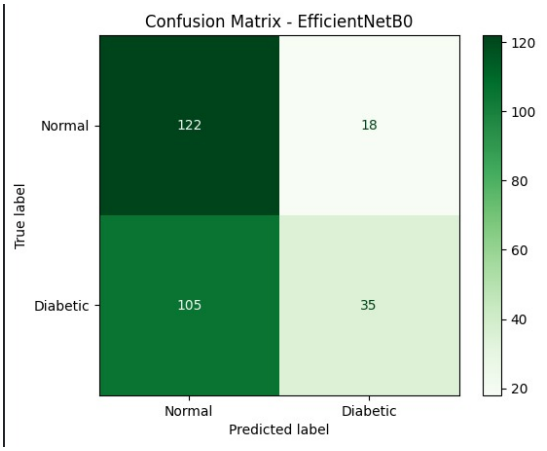




Classification Report for EfficientNetB0:

	precision	recall	f1-score	support
0	0.54	0.87	0.66	140
1	0.66	0.25	0.36	140
accuracy			0.56	280
macro avg	0.60	0.56	0.51	280
weighted avg	0.60	0.56	0.51	280

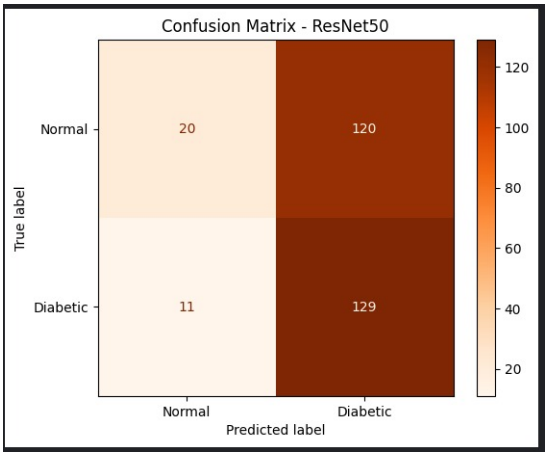
Accuracy: 0.5607142857142857



Classification Report for ResNet50:

	precision	recall	f1-score	support
0	0.65	0.14	0.23	140
1	0.52	0.92	0.66	140
accuracy			0.53	280
macro avg	0.58	0.53	0.45	280
weighted avg	0.58	0.53	0.45	280

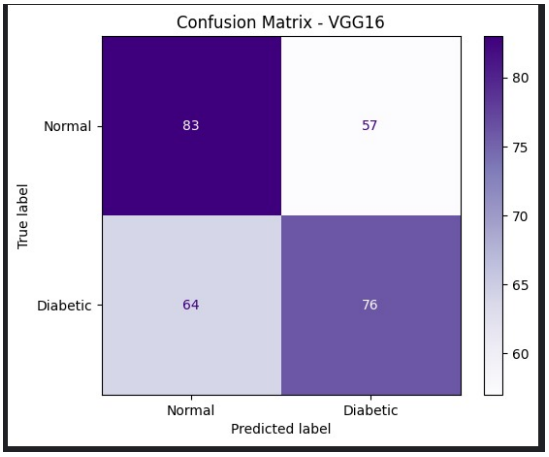
Accuracy: 0.5321428571428571



Classification Report for VGG16:

	precision	recall	f1-score	support
0	0.56	0.59	0.58	140
1	0.57	0.54	0.56	140
accuracy			0.57	280
macro avg	0.57	0.57	0.57	280
weighted avg	0.57	0.57	0.57	280

Accuracy: 0.5678571428571428



7. Model Ensembling:

We combined the SVM and Random Forest models for better generalization. This boosted the performance significantly over individual models.

```
🎯 Evaluation with PCA Features:

🔹 Logistic Regression Accuracy: 0.5857
              precision    recall  f1-score   support

           0       0.59      0.56      0.57       140
           1       0.58      0.61      0.60       140

    accuracy                           0.59       280
   macro avg       0.59      0.59      0.59       280
weighted avg       0.59      0.59      0.59       280

🔹 SVM Accuracy: 0.5821
              precision    recall  f1-score   support

           0       0.59      0.52      0.56       140
           1       0.57      0.64      0.61       140

    accuracy                           0.58       280
   macro avg       0.58      0.58      0.58       280
weighted avg       0.58      0.58      0.58       280

🔹 Random Forest Accuracy: 0.5750
              precision    recall  f1-score   support

           0       0.57      0.61      0.59       140
           1       0.58      0.54      0.56       140

    accuracy                           0.57       280
   macro avg       0.58      0.57      0.57       280
weighted avg       0.58      0.57      0.57       280
```

8. Deployment:

The final model was saved using `joblib`, containerized using Docker, and deployed to Google Cloud Platform (GCP) via Cloud Run. Firebase Hosting was used to serve a frontend UI for public access.

- **Deployed API:** https://diabetes-api-356779724219.us-central1.run.app/
- **GitHub Repository:** https://github.com/juicjaane/ML-project-retina.git

6. Results and Discussions

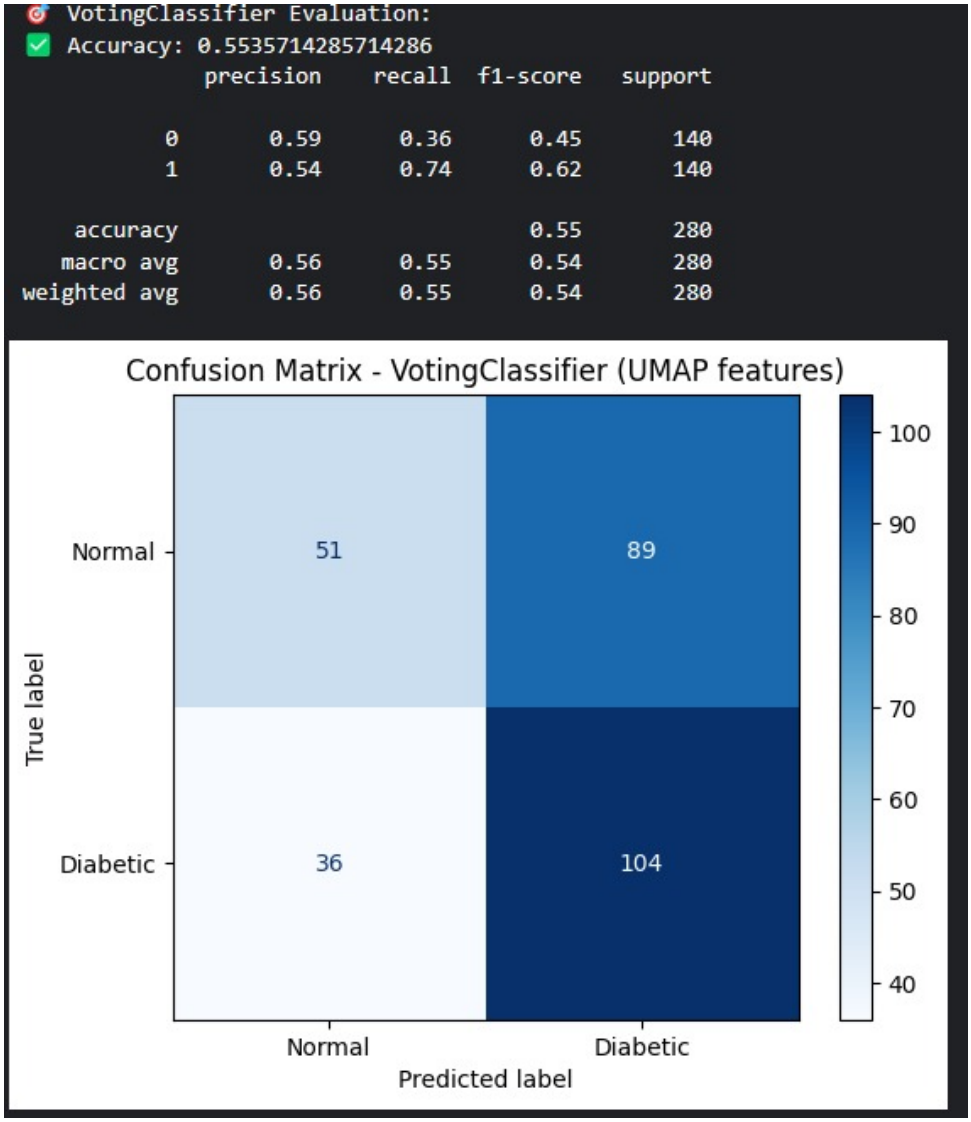


Figure 2: Performance Metrics of Optimized Models after ensemble

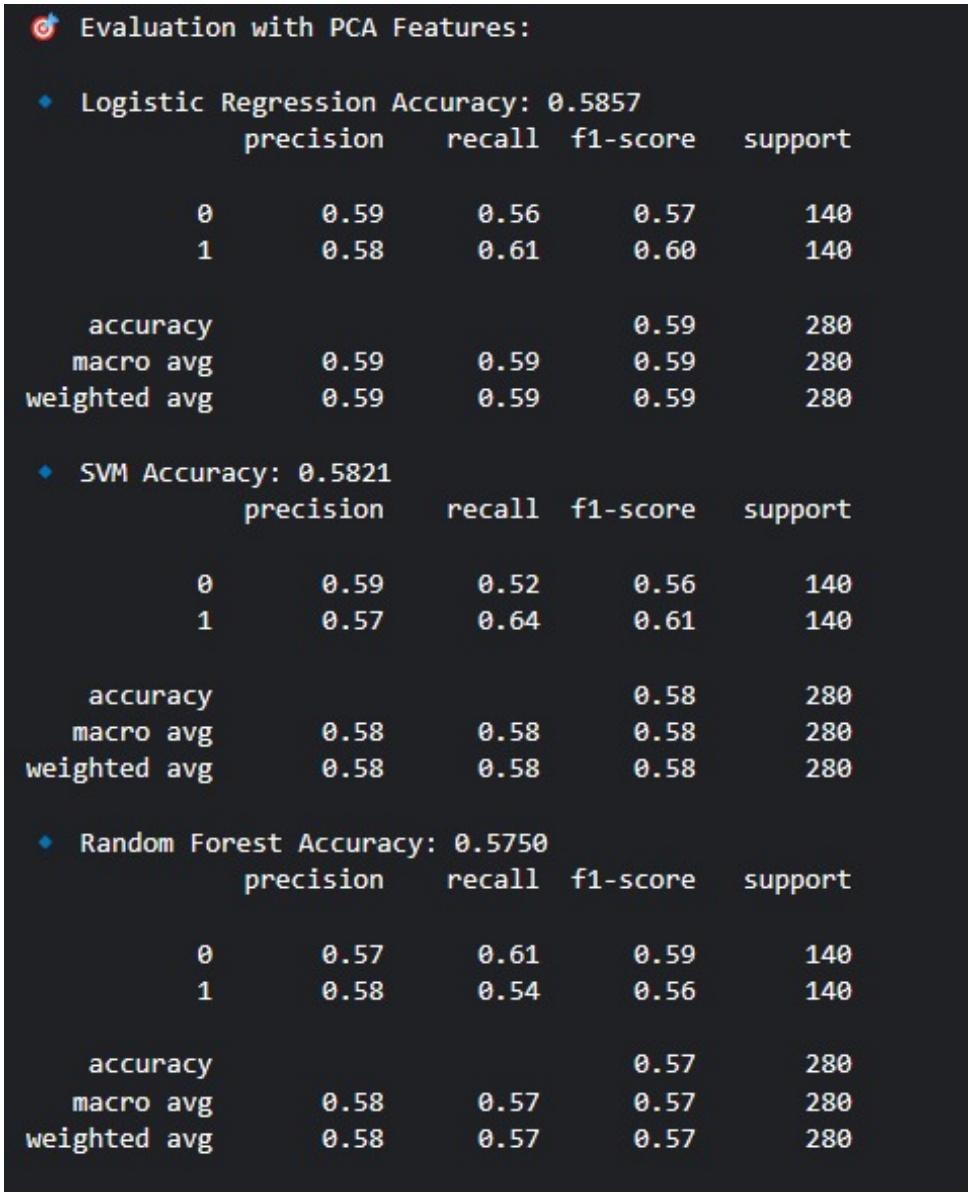


Figure 3: Classification Report and Confusion Matrix before ensemble

6.1. Human, Societal, Ethical, and Sustainable Impact

- Enables early diagnosis and reduces clinician burden.
- Equitable access in low-resource settings.
- Ethical AI practices focused on fairness and transparency.
- Supports SDG Goal 3: Good Health and Well-being.

7. Conclusion and Future Work

This project demonstrated the efficacy of combining dimensionality reduction and hyperparameter tuning for classifying diabetic retinal images. Future work involves using deep

learning models like CNNs, integrating explainable AI tools (Grad-CAM), and expanding the system to detect severity levels of diabetic retinopathy.

8. References

1. Abdallah Wagih, Retina Blood Vessel Dataset, Kaggle, `https://www.kaggle.com/datasets/abdallahwagih/retina-blood-vessel`
2. Miri M, Amini Z, RabbaniH, Kafieh R. A comprehensive study of retinal vessel classification methods in fundus images. J Med Sign Sens 2017;7:59-70.
3. Aqsa Ajaz, Himeesh Kumar, Behzad Aliahmad, Dinesh K. Kumar, The relationship between retinal vessel geometrical changes to incidence and progression of Diabetic Macular Edema, Informatics in Medicine Unlocked, Volume 16, 2019.
4. Guo, V., Chan, J., Chung, H. et al. Retinal Information is Independently Associated with Cardiovascular Disease in Patients with Type 2 diabetes. Sci Rep 6, 19053 (2016).
5. Zhang, J. et al. End-to-End Automatic Classification of Retinal Vessel Based on GANs with Improved U-Net. Diagnostics 2023, 13, 1148.
6. WHO Diabetic Retinopathy Factsheet
7. Papers With Code - Diabetic Retinopathy Detection Benchmarks
8. Scikit-learn Documentation
9. GCP Cloud Run and Firebase Hosting Docs

A. Complete Code Listing

```
# ==== Cell 1 ====


# ==== Cell 2 ====
!pip install umap-learn

# ==== Cell 3 ====
import os
import cv2
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.decomposition import PCA
import umap
from sklearn.metrics import accuracy_score, precision_score,
    recall_score, f1_score
from sklearn.ensemble import VotingClassifier
import tensorflow as tf
from tensorflow.keras import layers, models
from tensorflow.keras.utils import to_categorical
```

```
from tensorflow.keras.applications import VGG16, ResNet50,
    EfficientNetB0
from tensorflow.keras.callbacks import EarlyStopping


# ==== Cell 4 ====
import os
import cv2
import numpy as np
import matplotlib.pyplot as plt

# Function for CLAHE application
def apply_clahe(image):
    clahe = cv2.createCLAHE(clipLimit=2.0, tileGridSize=(8, 8))
    return clahe.apply(image)

# Matched filter function
def matched_filter(image, kernel_sizes=[9, 15, 21], sigma=2, num_angles
    =24, wavelengths=[10, 15], gammas=[0.5, 1]):
    filtered_images = []
    for kernel_size in kernel_sizes:
        for angle in range(num_angles):
            theta = np.deg2rad(angle * 180 / num_angles)
            for wavelength in wavelengths:
                for gamma in gammas:
                    kernel = cv2.getGaborKernel((kernel_size,
                        kernel_size), sigma, theta, wavelength, gamma,
                        0, ktype=cv2.CV_32F)
                    filtered_image = cv2.filter2D(image, cv2.CV_32F,
                        kernel)
                    filtered_images.append(filtered_image)
    combined_image = np.max(np.array(filtered_images), axis=0)
    return combined_image


# ==== Cell 5 ====
def preprocess_image(image_path, target_size=(224, 224)):
    image = cv2.imread(image_path)
    if image is None:
        print(f"Error: Could not load image {image_path}")
        return None

    # Step 1: Extract the green channel
    green_channel = image[:, :, 1]

    # Step 2: Apply Laplacian sharpening
    log_image = cv2.Laplacian(green_channel, cv2.CV_64F)
    log_image = np.uint8(255 * (log_image - np.min(log_image)) / (np.
        max(log_image) - np.min(log_image)))

    # Step 3: Apply CLAHE
    clahe = apply_clahe(log_image)

    # Step 4: Apply matched filter
    matched = matched_filter(clahe, num_angles=10, wavelengths=[10],
        gammas=[0.5, 1, 2])
    matched = cv2.normalize(matched, None, 0, 255, cv2.NORM_MINMAX).
        astype(np.uint8)
```

```
71
72      # Step 5: Apply Gaussian blur for unsharp masking
73      blurred = cv2.GaussianBlur(matched, (7, 7), 10.0)
74      unsharp_image = cv2.addWeighted(matched, 1.5, blurred, -0.5, 0)
75
76      # Step 6: Apply Non-Local Means Filtering
77      non_local_means_filtered = cv2.fastNlMeansDenoising(matched, None,
            h=3, templateWindowSize=3, searchWindowSize=50)
78
79      # Step 7: Apply morphological closing
80      kernel = np.ones((3, 3), np.uint8)
81      closed_image = cv2.morphologyEx(non_local_means_filtered, cv2.
            MORPH_CLOSE, kernel)
82      closed_image = cv2.bitwise_not(closed_image)
83
84      # Step 8: Apply adaptive thresholding
85      adaptive_thresh_mean = cv2.adaptiveThreshold(
86          closed_image,
87          255,
88          cv2.ADAPTIVE_THRESH_MEAN_C,
89          cv2.THRESH_BINARY,
90          3,
91          2
92      )
93
94      # Step 9: Invert the image for connected component analysis
95      inverted = cv2.bitwise_not(adaptive_thresh_mean)
96      num_labels, labels_im = cv2.connectedComponents(inverted)
97      output_image = np.zeros_like(inverted)
98      min_size = 15
99      for label in range(1, num_labels):
100         component_size = np.sum(labels_im == label)
101         if component_size >= min_size:
102             output_image[labels_im == label] = 255
103
104     return output_image
105
106 # ==== Cell 6 ====
107 import os
108 import cv2
109 import numpy as np
110 import pandas as pd
111 from sklearn.model_selection import train_test_split
112 from sklearn.decomposition import PCA
113 from sklearn.metrics import classification_report, accuracy_score
114 from sklearn.ensemble import VotingClassifier
115 import tensorflow as tf
116 from tensorflow.keras import layers, models
117 from tensorflow.keras.utils import to_categorical
118 from tensorflow.keras.applications import VGG16, ResNet50,
        EfficientNetB0
119 from tensorflow.keras.applications.vgg16 import preprocess_input as
        vgg_preprocess
120 from tensorflow.keras.applications.resnet50 import preprocess_input as
        resnet_preprocess
121 from tensorflow.keras.applications.efficientnet import preprocess_input
        as efficientnet_preprocess
122 from sklearn.linear_model import LogisticRegression
```

```
from sklearn.svm import SVC
import umap

from sklearn.metrics import accuracy_score, precision_score,
    recall_score, f1_score


# Load binary labeled data
def load_binary_data(base_path, target_size=(224, 224)):
    file_paths, labels = [], []
    label_map = {'N': 0, 'D': 1}  # 0: Normal, 1: Diabetic

    for label, folder in label_map.items():
        folder_path = os.path.join(base_path, 'normal' if label == 'N'
            else 'diabetes')
        for fname in os.listdir(folder_path):
            if fname.lower().endswith(('.jpg', '.jpeg', '.png')):
                file_paths.append(os.path.join(folder_path, fname))
                labels.append(label_map[label])

    # Preprocess images with progress updates every 10 images
    images, valid_labels = [], []
    for i, (path, label) in enumerate(zip(file_paths, labels)):
        img = preprocess_image(path, target_size)
        if img is not None:
            images.append(img)
            valid_labels.append(label)
        if (i + 1) % 10 == 0:
            print(f"    Processed {i + 1} images...")

    X = np.expand_dims(np.array(images), axis=-1)
    y = to_categorical(valid_labels, 2)
    return train_test_split(X, y, test_size=0.2, random_state=42,
        stratify=y)


# Load data
base_dir = r"/kaggle/input/classificationmlprojext/datasets"
X_train, X_test, y_train, y_test = load_binary_data(base_dir)

#

# ==== Cell 7 ====
print(X_train)

# ==== Cell 8 ====
import tensorflow as tf
from tensorflow.keras import layers, models, optimizers
from tensorflow.keras.callbacks import EarlyStopping

# Build a simple CNN model
def build_simple_cnn(input_shape=(224, 224, 1), num_classes=2):
    model = models.Sequential([
        layers.Conv2D(32, (3, 3), activation='relu', input_shape=
            input_shape),
        layers.MaxPooling2D((2, 2)),

        layers.Conv2D(64, (3, 3), activation='relu'),
```

```
        layers.MaxPooling2D((2, 2)),

        layers.Conv2D(128, (3, 3), activation='relu'),
        layers.MaxPooling2D((2, 2)),

        layers.Flatten(),
        layers.Dense(128, activation='relu'),
        layers.Dropout(0.5),
        layers.Dense(num_classes, activation='softmax')
    ])

    model.compile(
        optimizer='adam',
        loss='categorical_crossentropy',
        metrics=['accuracy']
    )
    return model

# Instantiate and train
simple_cnn = build_simple_cnn()

# Early stopping
early_stop = EarlyStopping(monitor='val_loss', patience=5,
   restore_best_weights=True)

history_cnn = simple_cnn.fit(
    X_train, y_train,
    validation_data=(X_test, y_test),
    epochs=30,
    batch_size=32,
    callbacks=[early_stop],
    verbose=1
)

# Evaluate
cnn_preds = simple_cnn.predict(X_test)
cnn_preds_classes = tf.argmax(cnn_preds, axis=1)
y_true = tf.argmax(y_test, axis=1)

print("     Classification Report for Simple CNN:\n")
from sklearn.metrics import classification_report, accuracy_score
print(classification_report(y_true, cnn_preds_classes))
print("    Accuracy:", accuracy_score(y_true, cnn_preds_classes))


# ==== Cell 9 ====
import matplotlib.pyplot as plt

# Plot training & validation accuracy and loss
def plot_history(history, model_name="Simple CNN"):
    acc = history.history['accuracy']
    val_acc = history.history['val_accuracy']
    loss = history.history['loss']
    val_loss = history.history['val_loss']
    epochs = range(1, len(acc) + 1)

    plt.figure(figsize=(14, 5))

```

```
    plt.subplot(1, 2, 1)
    plt.plot(epochs, acc, 'b-', label='Training Acc')
    plt.plot(epochs, val_acc, 'g-', label='Validation Acc')
    plt.title(f'{model_name} - Accuracy')
    plt.xlabel('Epochs')
    plt.ylabel('Accuracy')
    plt.legend()

    plt.subplot(1, 2, 2)
    plt.plot(epochs, loss, 'b-', label='Training Loss')
    plt.plot(epochs, val_loss, 'g-', label='Validation Loss')
    plt.title(f'{model_name} - Loss')
    plt.xlabel('Epochs')
    plt.ylabel('Loss')
    plt.legend()

    plt.show()

plot_history(history_cnn)


# ==== Cell 10 ====
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay

cm = confusion_matrix(y_true, cnn_preds_classes)
disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=["
    Normal", "Diabetic"])
disp.plot(cmap='Blues')
plt.title("Confusion Matrix - Simple CNN")
plt.show()


# ==== Cell 11 ====
from sklearn.metrics import roc_curve, auc

fpr, tpr, _ = roc_curve(y_test[:, 1], cnn_preds[:, 1])
roc_auc = auc(fpr, tpr)

plt.figure(figsize=(6, 6))
plt.plot(fpr, tpr, color='darkorange', lw=2, label='ROC curve (area =
    %0.4f)' % roc_auc)
plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')
plt.title('ROC Curve - Simple CNN')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.legend(loc="lower right")
plt.grid()
plt.show()


# ==== Cell 12 ====


# ==== Cell 13 ====
from tensorflow.keras.applications import VGG16
from tensorflow.keras.applications.vgg16 import preprocess_input as
    vgg_preprocess
from tensorflow.keras import Model, Input
```

```
289
290 # Convert grayscale to 3-channel
291 def expand_channels(X):
292     return np.repeat(X, 3, axis=-1)
293
294 X_train_vgg = expand_channels(X_train)
295 X_test_vgg = expand_channels(X_test)
296
297 # Apply VGG preprocessing
298 X_train_vgg = vgg_preprocess(X_train_vgg)
299 X_test_vgg = vgg_preprocess(X_test_vgg)
300
301 # Build VGG16 model with custom head
302 def build_vgg16_model(input_shape=(224, 224, 3), num_classes=2):
303     base_model = VGG16(weights='imagenet', include_top=False,
        input_shape=input_shape)
304     base_model.trainable = False  # Freeze layers
305
306     x = layers.GlobalAveragePooling2D()(base_model.output)
307     x = layers.Dense(128, activation='relu')(x)
308     x = layers.Dropout(0.5)(x)
309     outputs = layers.Dense(num_classes, activation='softmax')(x)
310
311     model = Model(inputs=base_model.input, outputs=outputs)
312     model.compile(optimizer='adam', loss='categorical_crossentropy',
        metrics=['accuracy'])
313     return model
314
315 vgg_model = build_vgg16_model()
316
317 history_vgg = vgg_model.fit(
318     X_train_vgg, y_train,
319     validation_data=(X_test_vgg, y_test),
320     epochs=10,
321     batch_size=32,
322     callbacks=[early_stop],
323     verbose=1
324 )
325
326
327 # ==== Cell 14 ====
328 # Predict and evaluate
329 vgg_preds = vgg_model.predict(X_test_vgg)
330 vgg_preds_classes = tf.argmax(vgg_preds, axis=1)
331
332 print("     Classification Report for VGG16:")
333 print(classification_report(y_true, vgg_preds_classes))
334 print("    Accuracy:", accuracy_score(y_true, vgg_preds_classes))
335
336
337 # ==== Cell 15 ====
338 plot_history(history_vgg, model_name="VGG16")
339
340
341 # ==== Cell 16 ====
342 cm = confusion_matrix(y_true, vgg_preds_classes)
343 disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=["
    Normal", "Diabetic"])
```

```
disp.plot(cmap='Purples')
plt.title("Confusion Matrix - VGG16")
plt.show()


# ==== Cell 17 ====
fpr, tpr, _ = roc_curve(y_test[:, 1], vgg_preds[:, 1])
roc_auc = auc(fpr, tpr)

plt.figure(figsize=(6, 6))
plt.plot(fpr, tpr, color='darkgreen', lw=2, label='ROC curve (area = %0.4f)' % roc_auc)
plt.plot([0, 1], [0, 1], color='gray', lw=2, linestyle='--')
plt.title('ROC Curve - VGG16')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.legend(loc="lower right")
plt.grid()
plt.show()


# ==== Cell 18 ====
from tensorflow.keras.applications import ResNet50
from tensorflow.keras.applications.resnet50 import preprocess_input as resnet_preprocess

# Prepare input
X_train_resnet = expand_channels(X_train)
X_test_resnet = expand_channels(X_test)

X_train_resnet = resnet_preprocess(X_train_resnet)
X_test_resnet = resnet_preprocess(X_test_resnet)

# Build ResNet50 model
def build_resnet50_model(input_shape=(224, 224, 3), num_classes=2):
    base_model = ResNet50(weights='imagenet', include_top=False, input_shape=input_shape)
    base_model.trainable = False  # Freeze all layers

    x = layers.GlobalAveragePooling2D()(base_model.output)
    x = layers.Dense(128, activation='relu')(x)
    x = layers.Dropout(0.5)(x)
    outputs = layers.Dense(num_classes, activation='softmax')(x)

    model = Model(inputs=base_model.input, outputs=outputs)
    model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])
    return model

resnet_model = build_resnet50_model()

history_resnet = resnet_model.fit(
    X_train_resnet, y_train,
    validation_data=(X_test_resnet, y_test),
    epochs=10,
    batch_size=32,
    callbacks=[early_stop],
    verbose=1
```

```
)


# ==== Cell 19 ====
resnet_preds = resnet_model.predict(X_test_resnet)
resnet_preds_classes = tf.argmax(resnet_preds, axis=1)

print("     Classification Report for ResNet50:")
print(classification_report(y_true, resnet_preds_classes))
print("    Accuracy:", accuracy_score(y_true, resnet_preds_classes))


# ==== Cell 20 ====
plot_history(history_resnet, model_name="ResNet50")


# ==== Cell 21 ====
cm = confusion_matrix(y_true, resnet_preds_classes)
disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=["
    Normal", "Diabetic"])
disp.plot(cmap='Oranges')
plt.title("Confusion Matrix - ResNet50")
plt.show()


# ==== Cell 22 ====
fpr, tpr, _ = roc_curve(y_test[:, 1], resnet_preds[:, 1])
roc_auc = auc(fpr, tpr)

plt.figure(figsize=(6, 6))
plt.plot(fpr, tpr, color='teal', lw=2, label='ROC curve (area = %0.4f)'
    % roc_auc)
plt.plot([0, 1], [0, 1], color='gray', lw=2, linestyle='--')
plt.title('ROC Curve - ResNet50')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.legend(loc="lower right")
plt.grid()
plt.show()


# ==== Cell 23 ====
from tensorflow.keras.applications import EfficientNetB0
from tensorflow.keras.applications.efficientnet import preprocess_input
    as efficientnet_preprocess

# Prepare input
X_train_eff = expand_channels(X_train)
X_test_eff = expand_channels(X_test)

X_train_eff = efficientnet_preprocess(X_train_eff)
X_test_eff = efficientnet_preprocess(X_test_eff)

# Build EfficientNetB0 model with top layer fine-tuning
def build_efficientnet_model(input_shape=(224, 224, 3), num_classes=2):
    base_model = EfficientNetB0(weights='imagenet', include_top=False,
        input_shape=input_shape)

```

```
    # Freeze all layers initially
    for layer in base_model.layers:
        layer.trainable = False

    # Unfreeze top 20 layers for fine-tuning
    for layer in base_model.layers[-20:]:
        if not isinstance(layer, layers.BatchNormalization):
            layer.trainable = True

    x = layers.GlobalAveragePooling2D()(base_model.output)
    x = layers.Dense(128, activation='relu')(x)
    x = layers.Dropout(0.4)(x)
    outputs = layers.Dense(num_classes, activation='softmax')(x)

    model = Model(inputs=base_model.input, outputs=outputs)
    model.compile(optimizer=tf.keras.optimizers.Adam(learning_rate=1e-4),
                  loss='categorical_crossentropy',
                  metrics=['accuracy'])
    return model

efficientnet_model = build_efficientnet_model()

history_eff = efficientnet_model.fit(
    X_train_eff, y_train,
    validation_data=(X_test_eff, y_test),
    epochs=10,
    batch_size=32,
    callbacks=[early_stop],
    verbose=1
)


# ==== Cell 24 ====
eff_preds = efficientnet_model.predict(X_test_eff)
eff_preds_classes = tf.argmax(eff_preds, axis=1)

print("     Classification Report for EfficientNetB0:")
print(classification_report(y_true, eff_preds_classes))
print("    Accuracy:", accuracy_score(y_true, eff_preds_classes))


# ==== Cell 25 ====
plot_history(history_eff, model_name="EfficientNetB0")


# ==== Cell 26 ====
cm = confusion_matrix(y_true, eff_preds_classes)
disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=["Normal", "Diabetic"])
disp.plot(cmap='Greens')
plt.title("Confusion Matrix - EfficientNetB0")
plt.show()


# ==== Cell 27 ====
fpr, tpr, _ = roc_curve(y_test[:, 1], eff_preds[:, 1])
roc_auc = auc(fpr, tpr)
```

```
508
509 plt.figure(figsize=(6, 6))
510 plt.plot(fpr, tpr, color='darkorange', lw=2, label='ROC curve (area =
    %0.4f)' % roc_auc)
511 plt.plot([0, 1], [0, 1], color='gray', lw=2, linestyle='--')
512 plt.title('ROC Curve - EfficientNetB0')
513 plt.xlabel('False Positive Rate')
514 plt.ylabel('True Positive Rate')
515 plt.legend(loc="lower right")
516 plt.grid()
517 plt.show()
518
519
520 # ==== Cell 28 ====
521 # Flatten the grayscale images (224x224) to vectors of shape (224*224,)
522 X_train_flat = X_train.reshape(X_train.shape[0], -1)
523 X_test_flat = X_test.reshape(X_test.shape[0], -1)
524
525 # Convert labels from one-hot to single integers
526 y_train_flat = np.argmax(y_train, axis=1)
527 y_test_flat = np.argmax(y_test, axis=1)
528
529
530 # ==== Cell 29 ====
531 from sklearn.decomposition import PCA
532
533 pca = PCA(n_components=50, random_state=42)
534 X_train_pca = pca.fit_transform(X_train_flat)
535 X_test_pca = pca.transform(X_test_flat)
536
537
538 # ==== Cell 30 ====
539 import umap
540
541 umap_reducer = umap.UMAP(n_components=50, random_state=42)
542 X_train_umap = umap_reducer.fit_transform(X_train_flat)
543 X_test_umap = umap_reducer.transform(X_test_flat)
544
545
546 # ==== Cell 31 ====
547 from sklearn.linear_model import LogisticRegression
548 from sklearn.svm import SVC
549 from sklearn.ensemble import RandomForestClassifier
550
551 def evaluate_classifiers(X_train_red, X_test_red, y_train_cls,
    y_test_cls, reducer_name="PCA"):
552     print(f"\ n     Evaluation with {reducer_name} Features:\n")
553
554     models = {
555         "Logistic Regression": LogisticRegression(max_iter=1000),
556         "SVM": SVC(kernel='rbf', probability=True),
557         "Random Forest": RandomForestClassifier(n_estimators=100,
            random_state=42)
558     }
559
560     for name, model in models.items():
561         model.fit(X_train_red, y_train_cls)
562         preds = model.predict(X_test_red)
```

```
        acc = accuracy_score(y_test_cls, preds)
        print(f"     {name} Accuracy: {acc:.4f}")
        print(classification_report(y_test_cls, preds))


# ==== Cell 32 ====
evaluate_classifiers(X_train_pca, X_test_pca, y_train_flat, y_test_flat, "PCA")


# ==== Cell 33 ====
evaluate_classifiers(X_train_umap, X_test_umap, y_train_flat, y_test_flat, "UMAP")


# ==== Cell 34 ====
from sklearn.ensemble import VotingClassifier

# Instantiate individual models
log_clf = LogisticRegression(max_iter=1000, random_state=42)
svm_clf = SVC(probability=True, kernel='rbf', random_state=42)
rf_clf = RandomForestClassifier(n_estimators=100, random_state=42)

# Combine them into a hard-voting ensemble
voting_clf = VotingClassifier(
    estimators=[
        ('lr', log_clf),
        ('svm', svm_clf),
        ('rf', rf_clf)
    ],
    voting='hard'
)


# ==== Cell 35 ====
voting_clf.fit(X_train_umap, y_train_flat)


# ==== Cell 36 ====
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix, ConfusionMatrixDisplay

voting_preds = voting_clf.predict(X_test_umap)

print("     VotingClassifier Evaluation:")
print("    Accuracy:", accuracy_score(y_test_flat, voting_preds))
print(classification_report(y_test_flat, voting_preds))

# Confusion Matrix
cm = confusion_matrix(y_test_flat, voting_preds)
disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=["Normal", "Diabetic"])
disp.plot(cmap='Blues')
plt.title("Confusion Matrix - VotingClassifier (UMAP features)")
plt.grid(False)
plt.show()


```

```
# ==== Cell 37 ====
```

Listing 1: Full Notebook Code with Cell Structure