MACHINE VISION (BCSE417L)

DIGITAL ASSIGNMENT- 2

DEEP LEARNING-BASED OCULAR DISEASE DETECTION USING FUNDUS IMAGES

SUBMITTED BY:

VIJITA MOHANRAAJ 22BAI1452

BRINDHA LN 22BAI1433

POOJA SASIKUMAR 22BAI1437

B. TECH- COMPUTER SCIENCE AND ENGINEERING (AI &ML)

SUBMITTED TO:

RAJARAJESWARI S

(SCOPE, VIT CHENNAI)

VIT CHENNAI

VANDALUR- KELAMBAKKAM ROAD, CHENNAI- 600127

ABSTRACT

Ocular diseases such as Diabetic Retinopathy (DR), Glaucoma (GL), and Cataract (CA) significantly impact vision and quality of life. This study proposes a deep learning-based multi-class classification system for fundus image analysis using a curated subset of the Eye Classification Dataset. Unlike traditional preprocessing, our approach combines Wavelet Transform for multi- resolution feature extraction, Contrast Limited Adaptive Histogram Equalization (CLAHE) for contrast enhancement, and blood vessel segmentation for vascular structure clarity. Two models—VGG-19 and ConvNeXt—are utilized to effectively capture both local and hierarchical image features. The proposed system improves classification performance, with ConvNeXt enhancing hierarchical feature extraction and VGG-19 leveraging deep spatial feature learning.

MOTIVATION

Early detection of ocular diseases such as Diabetic Retinopathy, Glaucoma, and Cataract is critical for preventing vision loss and ensuring timely treatment. Traditional screening methods, which often rely on manual examination, are not only time-consuming but also prone to inaccuracies.

To address these challenges, our project integrates advanced preprocessing techniques—including Wavelet Transform for multi-resolution feature extraction, CLAHE for improved contrast, and blood vessel segmentation to highlight key vascular structures. When combined with deep learning models like VGG-19 and ConvNeXt, these methods enhance feature clarity and enable robust classification, ultimately paving the way for a more accurate and efficient tool for early ocular disease detection.

OBJECTIVE

- Enhance fundus image quality using Wavelet Transform, CLAHE, and blood vessel segmentation for better feature extraction.
- The outputs of above three preprocessing techniques are stacked into a composite three-channel image that consolidates complementary diagnostic information.
- Develop a robust classification model using VGG-19 and ConvNeXt to improve disease recognition.
- Apply data augmentation techniques (rotation, flipping, contrast adjustments) to address class imbalances and improve generalization.
- Evaluate the system’s performance on a curated ODIR dataset, ensuring high accuracy, precision, and recall.
- Compare the performance of traditional CNNs (VGG-19) with ConvNeXt to determine the most effective architecture for ocular disease detection.

LITERATURE REVIEW

1. Ocular Disease Detection Using Advanced Neural Network Based Classification Algorithms.

This study explores the application of deep learning models for the early detection of ocular diseases using fundus images. Given the limitations of manual diagnosis, the authors propose an automated system trained on the ODIR dataset, consisting of 5000 fundus images classified into eight disease categories. The research employs four state-of-the-art convolutional neural network (CNN) architectures—VGG-16, ResNet-34, MobileNetV2, and EfficientNet—to evaluate their effectiveness in disease classification. The results indicate that VGG-16 achieved the highest accuracy of 97.23%, outperforming ResNet-34 (90.85%), MobileNetV2 (94.32%), and EfficientNet (93.82%). The study highlights the potential of deep learning in ophthalmology, emphasizing the importance of model selection for optimal performance. However, while the results are promising, the paper does not discuss computational efficiency or potential challenges in real-world deployment, such as class imbalances or dataset biases.

2. Automatic Recognition of Ocular Surface Diseases on Smartphone Images Using Densely Connected Convolutional Networks

Deep learning has significantly advanced medical image analysis, particularly in ophthalmology. Traditional diagnosis of ocular surface diseases requires clinical expertise, but convolutional neural networks (CNNs) have demonstrated high accuracy in automated classification. Chen et al. (2021) propose a smartphone-based system using DenseNet to classify ocular surface diseases with 90.6% accuracy. This approach reduces computational complexity while maintaining robust performance, enabling self-screening through mobile applications. The study aligns with prior research on mobile health applications, improving accessibility and early detection of eye diseases.

3. A Deep Learning Approach for Ocular Disease Detection

Deep learning has significantly improved ocular disease detection, with CNN-based models like VGG16, ResNet, and Inception achieving high accuracy. However, these models are computationally intensive, limiting their use on mobile devices. MobileNet, a lightweight architecture, has emerged as a suitable alternative for medical imaging due to its efficiency and accuracy. Studies show that MobileNet performs well in fundus image classification, making it ideal for smartphone-based detection. Recent research reports accuracy rates exceeding 90% using pre-trained models and fine-tuning techniques. The proposed study leverages MobileNet for automatic ocular disease detection, achieving 95.68% accuracy. This approach enhances accessibility for users without high-end computing resources, making early diagnosis more feasible.

4. Harnessing Deep Learning for Ocular Disease Diagnosis

This study investigates the effectiveness of various deep learning-based convolutional neural network (CNN) models—VGG-16, VGG-19, ResNet-50, and ResNet-152v2—in detecting ocular diseases. The research applies simple fine-tuning techniques to enhance model performance and compares their results. The findings indicate that all models perform well, even with minimal tuning. Among them, ResNet-152v2 achieves the highest training accuracy of 90.36%, demonstrating its strong learning capability. However, ResNet-50 is noted for its balanced performance, making it a more generalizable model. The study emphasizes the potential of deep learning in ophthalmology while acknowledging that fine-tuning and model selection significantly impact classification performance. Nonetheless, it does not explore computational efficiency or challenges related to dataset variability, which could influence real-world applicability.

5. Fundus-DeepNet: Multi-label Deep Learning Classification System for Enhanced Detection of Multiple Ocular Diseases through Data Fusion of Fundus Images

This study presents Fundus-DeepNet, a multi-label deep learning system for detecting multiple ocular diseases from fundus images. The model integrates deep feature representations from paired fundus images (left and right eyes) to improve diagnostic accuracy. The image preprocessing pipeline includes circular border cropping, contrast enhancement, noise removal, and data augmentation. Feature extraction is performed using High-Resolution Network (HRNet) and Attention Blocks, while SENet enhances feature quality. A Discriminative Restricted Boltzmann Machine (DRBM) with a Softmax layer is employed for classification. The model is evaluated on the OIA-ODIR dataset, achieving an F1-score of 88.56% (off-site) and 89.13% (on-site), demonstrating its strong capability in multi-disease classification. While the study highlights the benefits of deep feature fusion, it does not explore computational efficiency or the challenges of real-time deployment.

6. Application of Deep CNN Networks in Ocular Disease Detection

Deep convolutional neural networks (CNNs) have significantly advanced ocular disease detection, improving accuracy and efficiency. Previous studies have demonstrated various approaches to fundus image classification. In 2018, a deep CNN model achieved 87.51% accuracy in glaucoma detection, while in 2019, VGG-16 attained 88.71% accuracy for multiclass disease classification. EfficientNet B3, introduced in 2020, achieved 89.21% accuracy, highlighting its potential in automated diagnosis. Additionally, a two-VGG-16 model reported 85.57% accuracy, while a study in 2021 utilizing RFMiD with DenseNet201 and EfficientNet B4 achieved 80.2% accuracy. These results underscore the impact of CNN architectures in improving diagnostic precision. This study extends previous research by evaluating DenseNet, Inception-ResNet, EfficientNetB4, and EfficientNetB6, leveraging transfer learning for enhanced classification. The integration of deep learning techniques offers a robust solution to the limitations of traditional diagnostic methods, aiding ophthalmologists in early disease detection and effective treatment planning.

7. Deep Learning-Based Ocular Disease Classification in Fundus Images

This study proposes a deep learning-based multi-class classification model for detecting ocular diseases, including glaucoma, cataracts, and age-related macular degeneration (AMD), using fundus images. The model employs a pre-trained ResNet-50 architecture, trained on a dataset of 1,096 fundus images. The results demonstrate a 97% classification accuracy, with high precision, recall, and F1-score, indicating its robustness. The study highlights the model's ability to generalize well, avoiding overfitting, as shown by consistent training and validation performance. While effective, the study does not explore multi-label classification or advanced fusion techniques, which could enhance disease detection in complex cases.

8. Dual-Mode Imaging System for Early Detection and Monitoring of Ocular Surface Diseases

Deep learning has significantly advanced medical image analysis, particularly in ophthalmology. Traditional diagnosis of ocular surface diseases (OSDs) requires clinical expertise, but convolutional neural networks (CNNs) have demonstrated high accuracy in automated classification. Chen et al. (2021) propose a smartphone-based system using DenseNet to classify OSDs with 90.6% accuracy, improving accessibility for self-screening. Expanding on this, Li et al. (2024) introduce a dual-mode infrared (IR) and visible (RGB) imaging system for early detection and monitoring of OSDs. Their approach achieves 98.7% accuracy with an F1 score of 0.980 in disease classification and 96.2% accuracy with an F1 score of 0.956 in subconjunctival hemorrhage (SCH) detection. Additionally, their method analyzes meibomian gland dysfunction (MGD) with 88.1% accuracy. This system's automation and integration with portable IR cameras enhance home-based OSD monitoring. Together, these advancements significantly improve early detection, continuous assessment, and overall precision in eye healthcare.

9. Enhancing Ocular Healthcare: Deep Learning-Based Multi-Class Diabetic Eye Disease Segmentation and Classification

The study focuses on the detection and classification of Diabetic Eye Disease (DED) using deep learning models on retinal fundus images. It enhances image quality through green channel extraction, Contrast-Limited Adaptive Histogram Equalization (CLAHE), and illumination correction, followed by segmentation techniques like the Tyler Coye Algorithm, Otsu thresholding, and Circular Hough Transform to extract Regions of Interest (ROIs) such as the optic nerve, blood vessels, and macular region. The research evaluates four pre-trained models—ResNet-50, VGG-16, Exception, and EfficientNetB7—and finds that EfficientNetB7 achieves the best performance. Additionally, a customized Deep Convolutional Neural Network (DCNN) is proposed, which outperforms the pre-trained models, achieving classification accuracies of 96.43% for Cataracts, 98.33% for Diabetic Retinopathy, 97% for Glaucoma, and 96% for Normal cases. The study demonstrates that integrating image enhancement, segmentation, and deep learning significantly improves early diagnosis and treatment planning for diabetic patients, although it primarily focuses on DED classification rather than a broader multi-label disease detection approach.

10. Deep Learning-Based CNN for Multiclassification of Ocular Diseases Using Transfer Learning

Deepak and Bhat proposed a deep learning-based CNN model for multi-class classification of ocular diseases, specifically focusing on glaucoma and cataract detection. The study compared three pre-trained CNN models—SqueezeNet, Darknet-53, and EfficientNet-B0—and optimized them by varying batch size (6, 8, 10) and optimizer type (SGDM, RMSProp, Adam) to achieve maximum classification accuracy. Among the three models, Darknet-53 with a batch size of 6 and Adam optimizer achieved the highest accuracy of 99.4% on a test dataset of 1,000 images. The confusion matrix was used to evaluate performance metrics, including accuracy, sensitivity, specificity, F1-score, and ROC curve analysis. The study highlights the impact of hyperparameter tuning on the model’s performance and suggests that transfer learning with optimized CNN architectures can significantly improve ocular disease classification accuracy. By comparing the effectiveness of multiple models, the research underscores the importance of selecting an appropriate deep learning architecture for early detection and diagnosis of ocular diseases. The study provides valuable insights into AI-driven diagnostic tools that can assist ophthalmologists in making more accurate and efficient medical diagnoses.

11. Ocular Disease Detection with Deep Learning (Fine-Grained Image Categorization) Applied to Ocular B-Scan Ultrasound Images

Ye et al. developed a fine-grained deep learning classification system (DPLA-Net) for ocular disease detection using B-scan ultrasound images. The study focused on four major ocular conditions: intraocular tumors (IOT), retinal detachment (RD), vitreous hemorrhage (VH), and posterior scleral staphyloma (PSS). The dataset consisted of 6,054 ultrasound images collected from six hospitals in China, divided into training, validation, and testing sets (7:1:2 ratio). The proposed DPLA-Net model achieved a mean accuracy of 94.3% on the test set and AUC scores exceeding 0.98 for all disease categories. Additionally, the study found that using DPLA-Net significantly improved junior ophthalmologists' classification accuracy from 69.6% to 91.9%, while also reducing diagnosis time per image from 16.84 seconds to 10.09 seconds. These results highlight the potential of deep learning models in clinical applications, as they can enhance diagnostic efficiency, reduce human error, and assist ophthalmologists in making faster and more precise decisions. The study demonstrates that deep learning-based automated screening systems can effectively improve ocular disease detection in real-world medical settings.

12. Deep Learning for Ocular Disease Recognition: An Inner-Class Balance

Deep learning has significantly advanced ocular disease recognition, particularly through Convolutional Neural Networks (CNNs). The paper explored the use of deep learning to classify fundus images while addressing class imbalance by converting a multiclass problem into multiple binary classifications. The study employed the VGG-19 model and reported high accuracy. However, this article was later retracted due to concerns raised by the publisher. Class imbalance is a key challenge in medical image analysis, leading to biased models. Researchers have proposed solutions like instance-wise class-balanced sampling and hierarchical knowledge-guided learning. For example, Ju et al. introduced a hierarchical knowledge-based learning approach to enhance feature representation in long-tailed datasets. Similarly, Bhati et al. developed a Discriminative Kernel Convolution Network (DKCNet) with attention and squeeze-excitation blocks to improve feature extraction without increasing computational cost. These advancements highlight the importance of balancing medical datasets to improve deep learning performance in ocular disease recognition. Future research should focus on refining class imbalance solutions and integrating advanced feature extraction techniques to enhance diagnostic accuracy and robustness in real-world medical applications.

13. Recognition of Eye Diseases Based on Deep Neural Networks for Transfer Learning and Improved D-S Evidence Theory

Du et al. proposed a deep neural network (DNN)-based model for ocular disease detection, leveraging transfer learning and an improved Dempster-Shafer (D-S) evidence theory to enhance decision-making reliability. Unlike previous models that suffer from bias and inconsistencies, the authors introduced an Improved D-S Evidence Theory (ID-SET) to resolve paradoxes in traditional D-S theory, thereby improving credibility in classification results. The model was fine-tuned using transfer learning, significantly boosting learning efficiency and generalization ability. Experimental results demonstrated superior performance, with an accuracy of 92.37%, Kappa of 0.878, F1-score of 0.914, and an AUC of 0.987, outperforming comparable models. By incorporating decision fusion techniques, the model effectively mitigated decision bias and inconsistencies, making it robust for real-world applications. The study highlights the importance of fusing multiple deep learning models and leveraging improved probabilistic reasoning to achieve higher accuracy and reliability in eye disease recognition. These findings reinforce the potential of AI-driven ocular diagnostics, paving the way for more reliable and efficient automated screening tools in ophthalmology.

14. Ocular Eye Disease Prediction Using Machine Learning

Ocular Eye Disease Prediction Using Machine Learning Devanaboina et al. proposed a machine learning-based approach for ocular disease prediction, focusing on cataract detection using Convolutional Neural Networks (CNNs) and image pre-processing techniques. Cataract, a prevalent eye disease that causes blurred vision, is especially common among the elderly population. The study highlights the challenges of computer-aided diagnosis (CAD) in ophthalmology, emphasizing the need for automated systems to enhance early detection and treatment. The authors utilized CNN architectures to extract features from ocular images and classify them into diseased and non-diseased categories. The model's performance was evaluated using a confusion matrix, ensuring accurate prediction of cataract cases. The study demonstrates the effectiveness of deep learning in medical imaging and its potential in real-time eye disease screening. The proposed approach reduces manual diagnosis errors and improves early intervention strategies, making it a valuable tool for ophthalmologists. This research underscores the importance of machine learning in advancing ocular disease detection, paving the way for automated and efficient diagnostic systems in ophthalmology.

15. Automatic Recognition of Ocular Surface Diseases on Smartphone Images Using Densely Connected Convolutional Networks

Deep learning advancements have transformed ocular disease diagnosis by providing efficient alternatives to traditional, labor-intensive methods. Recent studies have implemented convolutional neural networks such as ResNet-152 and Inception V3 to analyze retinal images with promising accuracy. More recently, transformer-based architectures like CoAtNet have emerged, enhancing feature extraction and diagnostic precision. Utilizing comprehensive datasets like RFMiD, these approaches support real-time, computer-aided diagnosis. In our work, a custom-designed fundus camera with a 28-Diopter condensing lens captures retinal images for analysis. We compare the performance of ResNet-152, Inception V3, and CoAtNet, with CoAtNet demonstrating superior results. This integration of advanced imaging and deep learning holds significant potential for improving ocular disease screening and timely intervention.

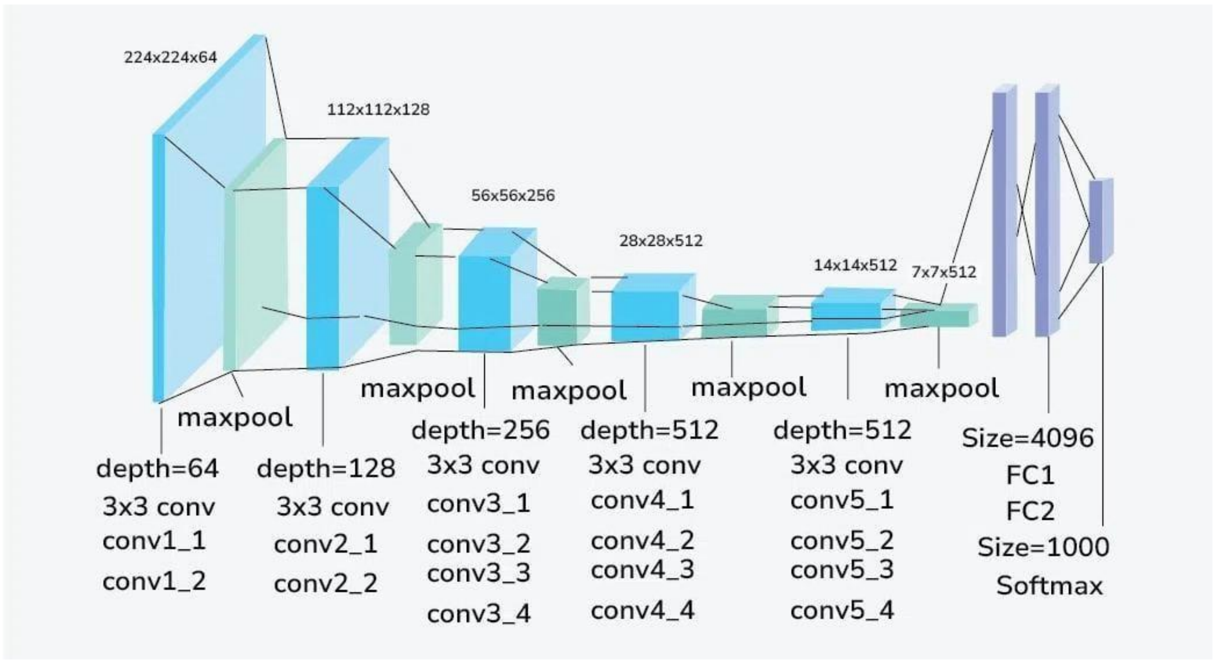
RESEARCH GAP

Current studies demonstrate the effectiveness of deep learning models for ocular disease detection, yet significant gaps remain. Many approaches focus on single-disease classification rather than comprehensive multi-label detection. Real-world applicability is often hindered by computational inefficiencies, dataset biases, and challenges in real-time clinical settings. Additionally, while transfer learning and decision fusion improve accuracy, studies rarely explore their effectiveness in diverse imaging conditions. Future research should focus on developing lightweight, real-time, multi-label classification models that address dataset biases and enhance clinical applicability across various imaging modalities.

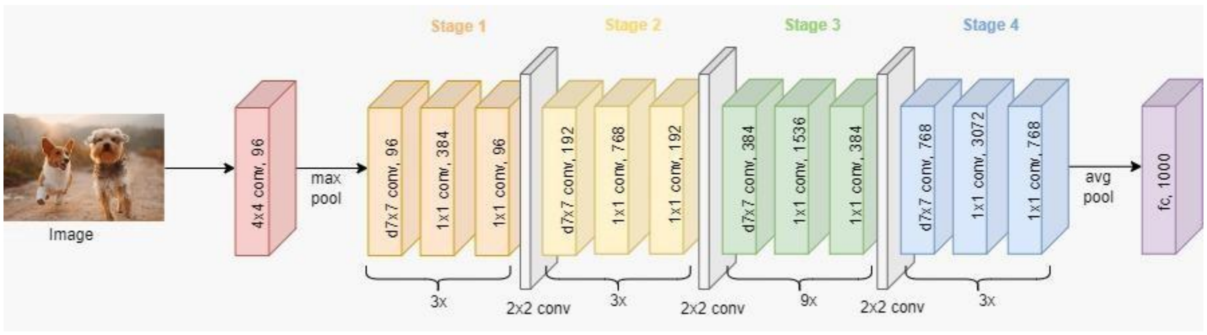
PROPOSED SYSTEM

ARCHITECTURE DIAGRAM:

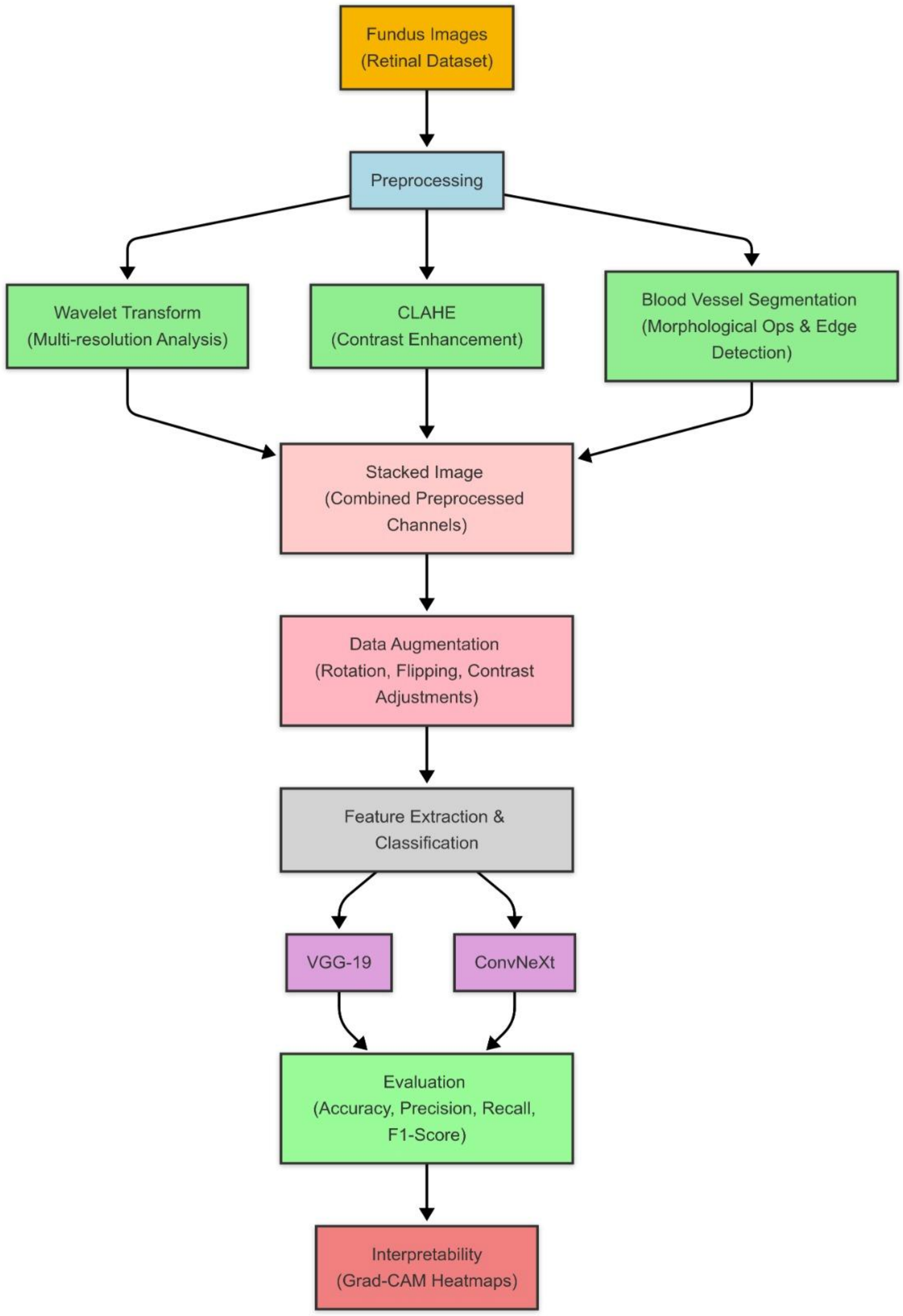
VGG-19:



ConvNeXt:



System architecture:



PROPOSED SYSTEM AND METHODOLOGY

The system is designed to detect ocular diseases—Diabetic Retinopathy (DR), Glaucoma (GL), and Cataract (CA)—from fundus images obtained from the ODIR dataset. The methodologies adopted in the system include:

Preprocessing pipeline:
The raw fundus images undergo a series of preprocessing steps to enhance the image quality:
Wavelet Transform is applied for multi-resolution feature extraction.
CLAHE is used to improve contrast.
Blood Vessel Segmentation isolates important vascular structures.
The outputs from these three preprocessing steps are then stacked to form a single multi-channel image, where each channel carries distinct enhanced information. This stacked image acts as a richer representation of the original fundus image, capturing complementary features useful for classification.

Data augmentation:
To further improve model robustness and mitigate class imbalance, the system applies data augmentation techniques such as rotation, flipping, and contrast adjustments to the pre-processed images.

Model training and comparison:
Two deep learning models—VGG-19 and ConvNeXt—are trained on the augmented dataset. Each model processes the enhanced images independently, and their performances are compared based on metrics such as accuracy, precision, recall, and F1-score.

Evaluation:
The performance of each model is evaluated on a separate test set from the ODIR dataset. The results are then compared to determine which architecture better captures the complex features of fundus images for the accurate detection of ocular diseases.

Model Interpretability with Grad-CAM
To enhance the interpretability of the deep learning predictions:

- Grad-CAM (Gradient-weighted Class Activation Mapping) is used to visualize the regions of the image that contribute most to the model's decision.
- These heatmaps help verify whether the model is focusing on medically relevant features in the fundus images.

ALGORITHM

The key algorithms implemented in this system are as follows:

Wavelet Transform:
This algorithm decomposes each fundus image into multiple frequency components, enabling a multi-resolution analysis that captures both fine and coarse details. Such an approach is crucial for identifying subtle lesions and structural changes.

CLAHE (Contrast Limited Adaptive Histogram Equalization):
CLAHE improves local contrast within the image by redistributing the lightness values. This enhancement makes subtle pathological features more prominent without amplifying noise excessively.

Blood Vessel Segmentation:
Through morphological operations and edge detection (or thresholding techniques), this algorithm isolates the blood vessel structures. Since vascular patterns are significant indicators in ocular diseases, this step helps to focus feature extraction on clinically relevant regions.

Stacked Image Construction:
To enrich the input representation, three enhanced versions of each fundus image—after Wavelet Transform, CLAHE, and Blood Vessel Segmentation—are stacked into a single 3-channel image. This composite input provides the model with multiple perspectives of the same retina, combining multi-resolution, contrast-enhanced, and vessel-focused features. This stacked image serves as a unified and information-rich input to the classification models.

Data Augmentation Techniques:
These techniques, including random rotations, flipping, and contrast adjustments, artificially enlarge the dataset, helping to overcome class imbalance and improve the robustness of the models.

VGG-19:
A widely recognized convolutional neural network known for its effective layer structure, VGG-19 excels at learning rich image features from fundus data. Its well-established design makes it a dependable choice for medical image analysis, enabling the detection of intricate patterns and variations in the input images.

ConvNeXt:
A convolutional neural network that incorporates modern design innovations and optimization techniques, ConvNeXt offers improved efficiency in feature extraction. Its advanced architecture leads to superior performance in medical image classification tasks, making it highly suitable for complex diagnostic applications.

Grad-CAM (Gradient-weighted Class Activation Mapping):
Grad-CAM works by backpropagating the gradients of a target class prediction into the final convolutional layer of the CNN. These gradients are then global-average pooled to obtain importance weights for each feature map channel. The weighted combination of these feature maps forms a coarse localization map—essentially a heatmap—that highlights the regions in the image that had the most influence on the prediction.

SOURCE CODE

#Downloading the Eye Disease Classification Dataset, Preprocessing the Images (Wavelet Transform, CLAHE, Blood Vessel Segmentation), and Loading Preprocessed Data for Model Training

```
from google.colab import drive
drive.mount('/content/drive')
from google.colab import files
files.upload()
import os
import shutil

# Create the .kaggle directory
os.makedirs('/root/.kaggle', exist_ok=True)

# Move the kaggle.json file to the .kaggle directory
shutil.move('kaggle.json', '/root/.kaggle/kaggle.json')

# Set the permissions of the kaggle.json file
os.chmod('/root/.kaggle/kaggle.json', 600)

# Define the path where you want to save the dataset in your Google Drive
dataset_path = '/content/drive/MyDrive/eye_diseases_classification'

# Create the directory if it doesn't exist
os.makedirs(dataset_path, exist_ok=True)

# Change the current working directory to the dataset path
os.chdir(dataset_path)

# Download the dataset using Kaggle API
!kaggle datasets download -d gunavenkatdoddi/eye-diseases-classificatio

# Unzip the downloaded dataset
!unzip eye-diseases-classification.zip -d .

# Optionally, remove the zip file to save space
os.remove('eye-diseases-classification.zip')
import os

# Path to the dataset directory
dataset_dir = '/content/drive/MyDrive/eye_diseases_classification/dataset'

# List files and directories inside the 'dataset' folder
print(os.listdir(dataset_dir))
import os
import cv2
import numpy as np
import pywt
from tqdm import tqdm
from tensorflow.keras.preprocessing.image import img_to_array
import matplotlib.pyplot as plt
```

```
# Google Drive path for saving preprocessed data
save_path = '/content/drive/MyDrive/eye_diseases_classification/preprocessed_stacked'
os.makedirs(save_path, exist_ok=True)

IMG_SIZE = (224, 224)
classes = ['cataract', 'diabetic_retinopathy', 'glaucoma', 'normal']
original_data_path = '/content/drive/MyDrive/eye_diseases_classification/dataset'

# ---------- Updated Preprocessing Functions ----------

# Replacing wavelet with Gaussian blur on Red channel
def wavelet_transform(image):
    _, _, r = cv2.split(image)
    return cv2.GaussianBlur(r, (5, 5), 0)

# CLAHE on Green channel
def apply_clahe(image):
    _, g, _ = cv2.split(image)
    clahe = cv2.createCLAHE(clipLimit=3.0, tileGridSize=(8, 8))
    return clahe.apply(g)

# Binary thresholding on Blue channel
def blood_vessel_segmentation(image):
    b, _, _ = cv2.split(image)
    _, segmented = cv2.threshold(b, 100, 255, cv2.THRESH_BINARY)
    return segmented

# ---------- Preprocessing and Saving ----------

def preprocess_and_save_all():
    for class_name in classes:
        class_input_path = os.path.join(original_data_path, class_name)
        class_output_path = os.path.join(save_path, class_name)
        os.makedirs(class_output_path, exist_ok=True)

        for filename in tqdm(os.listdir(class_input_path), desc=f"Processing {class_name}"):
            img_path = os.path.join(class_input_path, filename)
            try:
                image = cv2.imread(img_path)
                image = cv2.resize(image, IMG_SIZE)

                wavelet = cv2.resize(wavelet_transform(image), IMG_SIZE)
                clahe = cv2.resize(apply_clahe(image), IMG_SIZE)
                vessel = cv2.resize(blood_vessel_segmentation(image), IMG_SIZE)

                # Stack all into 3 channels
                stacked = np.stack([wavelet, clahe, vessel], axis=-1)

                # Save the stacked image as .npy (fast for loading later)
                np.save(os.path.join(class_output_path, filename.split('.')[0] + '.npy'), stacked)
            except Exception as e:
                print(f"Error processing {img_path}: {e}")
```

```
# Run the preprocessing
preprocess_and_save_all()
import os
import cv2
import numpy as np
import pywt
from tqdm import tqdm
from tensorflow.keras.preprocessing.image import img_to_array
import matplotlib.pyplot as plt

# Google Drive path for saving preprocessed data
save_path = '/content/drive/MyDrive/eye_diseases_classification/preprocessed_stacked'
os.makedirs(save_path, exist_ok=True)

IMG_SIZE = (224, 224)
classes = ['cataract', 'diabetic_retinopathy', 'glaucoma', 'normal']
original_data_path = '/content/drive/MyDrive/eye_diseases_classification/dataset'

# Wavelet Transform
def wavelet_transform(image):
    gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
    coeffs2 = pywt.dwt2(gray, 'bior1.3')
    LL, (LH, HL, HH) = coeffs2
    reconstructed = pywt.idwt2((LL, (LH, HL, HH)), 'bior1.3')
    return np.uint8(reconstructed)

# CLAHE Enhancement
def apply_clahe(image):
    gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
    clahe = cv2.createCLAHE(clipLimit=2.0, tileGridSize=(8, 8))
    return clahe.apply(gray)

# Blood Vessel Segmentation
def blood_vessel_segmentation(image):
    gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
    _, thresholded = cv2.threshold(gray, 100, 255, cv2.THRESH_BINARY)
    kernel = np.ones((3, 3), np.uint8)
    return cv2.morphologyEx(thresholded, cv2.MORPH_CLOSE, kernel)

# Preprocess and save all images
def preprocess_and_save_all():
    for class_name in classes:

        class_input_path = os.path.join(original_data_path, class_name)
        class_output_path = os.path.join(save_path, class_name)
        os.makedirs(class_output_path, exist_ok=True)

        for filename in tqdm(os.listdir(class_input_path), desc=f"Processing {class_name}"):
            img_path = os.path.join(class_input_path, filename)
            try:
                image = cv2.imread(img_path)
                image = cv2.resize(image, IMG_SIZE)
                wavelet = cv2.resize(wavelet_transform(image), IMG_SIZE)
                clahe = cv2.resize(apply_clahe(image), IMG_SIZE)
                vessel = cv2.resize(blood_vessel_segmentation(image), IMG_SIZE)

                # Stack all into 3 channels
                stacked = np.stack([wavelet, clahe, vessel], axis=-1)

                # Save the stacked image as .npy (fast for loading later)
                np.save(os.path.join(class_output_path, filename.split('.')[0] + '.npy'), stacked)
            except:
                print(f"Error processing {img_path}")

preprocess_and_save_all()
def show_preprocessing_steps(class_name='normal'):
    # Choose a sample file from the preprocessed folder
    class_folder = os.path.join(original_data_path, class_name)
    sample_file = os.listdir(class_folder)[0]
    sample_path = os.path.join(class_folder, sample_file)

    image = cv2.imread(sample_path)
    image = cv2.resize(image, IMG_SIZE)
    wavelet = wavelet_transform(image)
    clahe = apply_clahe(image)
    vessel = blood_vessel_segmentation(image)
    stacked = np.stack([wavelet, clahe, vessel], axis=-1)

    # Display
    plt.figure(figsize=(12, 8))
    plt.subplot(2, 3, 1); plt.imshow(cv2.cvtColor(image, cv2.COLOR_BGR2RGB));
plt.title("Original"); plt.axis('off')
    plt.subplot(2, 3, 2); plt.imshow(wavelet, cmap='gray'); plt.title("Wavelet"); plt.axis('off')
    plt.subplot(2, 3, 3); plt.imshow(clahe, cmap='gray'); plt.title("CLAHE"); plt.axis('off')
    plt.subplot(2, 3, 4); plt.imshow(vessel, cmap='gray'); plt.title("Blood Vessels"); plt.axis('off')
    plt.subplot(2, 3, 5); plt.imshow(stacked.astype('uint8')); plt.title("Stacked"); plt.axis('off')
    plt.tight_layout()
    plt.show()

show_preprocessing_steps('normal')
def load_stacked_images(preprocessed_path):
    X = []
    y = []
    class_map = {c: i for i, c in enumerate(classes)}
    for class_name in classes:


class_dir = os.path.join(preprocessed_path, class_name)
        for file in os.listdir(class_dir):

            if file.endswith('.npy'):
                arr = np.load(os.path.join(class_dir, file))
                X.append(arr)
                y.append(class_map[class_name])
    return np.array(X), np.array(y)
```

```
# Load from saved .npy files
X_data, y_data = load_stacked_images(save_path)
# Create a simple data generator for loading and batching preprocessed images
def load_data_from_npy(dataset_dir):
    images = []
    labels = []

    class_names = os.listdir(dataset_dir)
    for class_name in class_names:
        class_dir = os.path.join(dataset_dir, class_name)
        if os.path.isdir(class_dir):
            for filename in os.listdir(class_dir):
                if filename.endswith(".npy"):
                    img_path = os.path.join(class_dir, filename)
                    image = load_preprocessed_image(img_path)  # Load preprocessed image
                    images.append(image)
                    labels.append(class_names.index(class_name))  # Label is the class index

    images = np.array(images)
    labels = np.array(labels)

    return images, labels

# Load the full dataset
X_data, y_data = load_data_from_npy(dataset_dir)

# Check shape of the dataset
print("Training data shape:", X_data.shape)  # Should be (num_samples, 224, 224, channels)
print("Labels shape:", y_data.shape)

# Split data into train and validation sets
X_train, X_val, y_train, y_val = train_test_split(X_data, y_data, test_size=0.2,
random_state=42)

# Convert labels to one-hot encoding
y_train = to_categorical(y_train, num_classes=4)
y_val = to_categorical(y_val, num_classes=4)
```

#VGG19-based Transfer Learning for Eye Disease Classification with Grad-CAM Visualization

```
from tensorflow.keras.applications import VGG19
from tensorflow.keras.models import Model
from tensorflow.keras.layers import Dense, GlobalAveragePooling2D, Input
from tensorflow.keras.utils import to_categorical
from tensorflow.keras.optimizers import Adam
from sklearn.model_selection import train_test_split

# Assuming classes is already defined
num_classes = len(classes)

# Load your data from .npy files
X_data, y_data = load_stacked_images(dataset_dir)
```

```
# Normalize the input data
X_data = X_data.astype('float32') / 255.0

# One-hot encode the labels
y_data = to_categorical(y_data, num_classes=num_classes)

# Train-test split
X_train, X_val, y_train, y_val = train_test_split(X_data, y_data, test_size=0.2,
random_state=42, stratify=y_data)

# Define the input shape
input_shape = (224, 224, 3)

# Load the base VGG19 model without top layers
base_model = VGG19(weights='imagenet', include_top=False,
input_tensor=Input(shape=input_shape))

# Freeze base model layers
for layer in base_model.layers:
    layer.trainable = False

# Add custom top layers
x = base_model.output
x = GlobalAveragePooling2D()(x)
x = Dense(256, activation='relu')(x)
x = Dense(128, activation='relu')(x)
predictions = Dense(num_classes, activation='softmax')(x)

# Combine into final model
model = Model(inputs=base_model.input, outputs=predictions)

# Compile model
model.compile(optimizer=Adam(learning_rate=1e-4), loss='categorical_crossentropy',
metrics=['accuracy'])

# Print model summary
model.summary()

# Train the model
history = model.fit(X_train, y_train,
                    validation_data=(X_val, y_val),
                    epochs=10,
                    batch_size=32)

# Save the model
model.save('/content/drive/MyDrive/eye_diseases_classification/vgg19_eye_disease_model.h
5')
plt.plot(history.history['accuracy'], label='Train Accuracy')
plt.plot(history.history['val_accuracy'], label='Val Accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.title('VGG19 Training Accuracy')
plt.legend()
plt.grid(True)
plt.show()
import tensorflow as tf
from tensorflow.keras.callbacks import EarlyStopping, ModelCheckpoint, Callback
from tensorflow.keras.optimizers import Adam

# Safety batch size for Colab
BATCH_SIZE = 16
EPOCHS = 15

# Convert numpy arrays to TensorFlow datasets (better memory handling)
train_dataset = tf.data.Dataset.from_tensor_slices((X_train, y_train))
val_dataset = tf.data.Dataset.from_tensor_slices((X_val, y_val))

# Shuffle and batch
train_dataset =
train_dataset.shuffle(buffer_size=1024).batch(BATCH_SIZE).prefetch(tf.data.AUTOTUNE)
val_dataset = val_dataset.batch(BATCH_SIZE).prefetch(tf.data.AUTOTUNE)

# Freeze first layers, fine-tune later layers
fine_tune_at = 17
for layer in base_model.layers[:fine_tune_at]:
    layer.trainable = False
for layer in base_model.layers[fine_tune_at:]:
    layer.trainable = True

# Recompile with lower learning rate
model.compile(
    optimizer=Adam(learning_rate=1e-5),
    loss='categorical_crossentropy',
    metrics=['accuracy']
)

# Custom callback to show accuracy
class AccuracyPrinter(Callback):
    def on_epoch_end(self, epoch, logs=None):
        print(f"Epoch {epoch + 1}: Train Acc = {logs['accuracy']:.4f}, Val Acc =
{logs['val_accuracy']:.4f}")

# Callbacks
callbacks = [
    EarlyStopping(monitor='val_loss', patience=4, restore_best_weights=True),

ModelCheckpoint('/content/drive/MyDrive/eye_diseases_classification/vgg19_finetuned_best
_model.h5',
                    monitor='val_accuracy', save_best_only=True, verbose=1),
    AccuracyPrinter()
]

# Fine-tuning with stable training loop
fine_tune_history = model.fit(train_dataset,validation_data=val_dataset,
    epochs=EPOCHS,
```

```python
callbacks=callbacks,
    verbose=0  # We handle printing manually
)
# Combine history from initial training + fine-tuning
def plot_combined_history(initial, finetune):
    acc = initial.history['accuracy'] + finetune.history['accuracy']
    val_acc = initial.history['val_accuracy'] + finetune.history['val_accuracy']
    loss = initial.history['loss'] + finetune.history['loss']
    val_loss = initial.history['val_loss'] + finetune.history['val_loss']
    plt.figure(figsize=(12, 5))
    plt.subplot(1, 2, 1)
    plt.plot(acc, label='Train Acc')
    plt.plot(val_acc, label='Val Acc')
    plt.title('Training + Fine-Tuning Accuracy')
    plt.xlabel('Epoch')
    plt.ylabel('Accuracy')
    plt.legend()
    plt.subplot(1, 2, 2)
    plt.plot(loss, label='Train Loss')
    plt.plot(val_loss, label='Val Loss')
    plt.title('Training + Fine-Tuning Loss')
    plt.xlabel('Epoch')
    plt.ylabel('Loss')
    plt.legend()
    plt.tight_layout()
    plt.show()

plot_combined_history(history, fine_tune_history)
from sklearn.metrics import confusion_matrix, classification_report
import seaborn as sns
import numpy as np
import pandas as pd # Import the pandas library

# Predict on validation set
y_pred_probs = model.predict(X_val)
y_pred_classes = np.argmax(y_pred_probs, axis=1)
y_true_classes = np.argmax(y_val, axis=1)


# --- Confusion Matrix ---
cm = confusion_matrix(y_true_classes, y_pred_classes)

plt.figure(figsize=(8, 6))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues',
            xticklabels=class_labels,
            yticklabels=class_labels)
plt.xlabel('Predicted Label')
plt.ylabel('True Label')
plt.title('Confusion Matrix')
plt.show()

# --- Classification Report ---
print("Classification Report:\n")
print(classification_report(y_true_classes, y_pred_classes, target_names=class_labels))

# --- Correlation Matrix of Predictions ---
# Convert one-hot to raw probabilities, then compute correlation
pred_probs_df = pd.DataFrame(y_pred_probs, columns=class_labels) # Use pd alias to
access DataFrame
corr_matrix = pred_probs_df.corr()

plt.figure(figsize=(8, 6))
sns.heatmap(corr_matrix, annot=True, cmap='coolwarm', square=True)
plt.title("Prediction Correlation Matrix")
plt.show()
import numpy as np
import matplotlib.pyplot as plt

def display_actual_vs_predicted(model, X_data, y_data, class_labels, num_samples=9):
    # Predict the labels for the test set
    y_pred_probs = model.predict(X_data)
    y_pred_classes = np.argmax(y_pred_probs, axis=1)
    y_true_classes = np.argmax(y_data, axis=1)

    # Show predictions
    plt.figure(figsize=(12, 12))
    for i in range(num_samples):
        plt.subplot(3, 3, i + 1)

        # Normalize image for display if needed
        img = X_data[i]
        if img.max() > 1:
            img = img / 255.0

        plt.imshow(img)
        actual = class_labels[y_true_classes[i]]
        predicted = class_labels[y_pred_classes[i]]
        plt.title(f"Actual: {actual}\nPredicted: {predicted}", fontsize=10)
        plt.axis('off')


    plt.tight_layout()
    plt.show()

# Use on validation or test data
class_labels = ['Cataract', 'Diabetic Retinopathy', 'Glaucoma', 'Normal']
display_actual_vs_predicted(model, X_val, y_val, class_labels)
import numpy as np
import tensorflow as tf
import matplotlib.pyplot as plt
import cv2

def get_img_array(img, size):
    """Prepares image for model prediction"""
    img = cv2.resize(img, size)
    img = np.expand_dims(img, axis=0)
    return img
```

```
def make_gradcam_heatmap(img_array, model, last_conv_layer_name, pred_index=None):
    # Create a model that maps the input image to the activations and prediction
    grad_model = tf.keras.models.Model(
        [model.inputs],
        [model.get_layer(last_conv_layer_name).output, model.output]
    )

    # Compute gradients
    with tf.GradientTape() as tape:
        conv_outputs, predictions = grad_model(img_array)
        if pred_index is None:
            pred_index = tf.argmax(predictions[0])
        class_channel = predictions[:, pred_index]

    # Gradients of the predicted class wrt conv layer output
    grads = tape.gradient(class_channel, conv_outputs)

    # Mean intensity of gradients across the feature map
    pooled_grads = tf.reduce_mean(grads, axis=(0, 1, 2))

    # Multiply each channel in the feature map array by "how important this channel is" with
respect to the predicted class
    conv_outputs = conv_outputs[0]
    heatmap = conv_outputs @ pooled_grads[..., tf.newaxis]
    heatmap = tf.squeeze(heatmap)

    # Normalize the heatmap
    heatmap = tf.maximum(heatmap, 0) / tf.math.reduce_max(heatmap)
    return heatmap.numpy()

def save_and_display_gradcam(img, heatmap, alpha=0.4):
    # Resize heatmap to image size
    heatmap = cv2.resize(heatmap, (img.shape[1], img.shape[0]))

    # Convert heatmap to RGB
    heatmap = np.uint8(255 * heatmap)
    heatmap = cv2.applyColorMap(heatmap, cv2.COLORMAP_JET)

    # Superimpose the heatmap on original image
    superimposed_img = heatmap * alpha + img
    superimposed_img = np.uint8(superimposed_img)

    # Plot
    plt.figure(figsize=(6, 6))
    plt.imshow(superimposed_img)
    plt.axis('off')
    plt.title("Grad-CAM: Highlighted Region")
    plt.show()

# --- Use the functions on an image from X_val ---
# Pick any image from your validation set
idx = 0  # change index to try different examples
original_img = (X_val[idx] * 255).astype(np.uint8)  # convert back if normalized
img_array = get_img_array(original_img, size=(224, 224))
```

```
# Class labels
class_labels = ['Cataract', 'Diabetic Retinopathy', 'Glaucoma', 'Normal']

# Get heatmap
last_conv_layer_name = 'block5_conv4'  # last conv layer in VGG19
heatmap = make_gradcam_heatmap(img_array, model, last_conv_layer_name)

# Display heatmap over original image
save_and_display_gradcam(original_img, heatmap)

# Also show prediction
pred = np.argmax(model.predict(img_array), axis=1)[0]
print(f"Predicted Class: {class_labels[pred]}")
```

#ConvNeXt-based Transfer Learning for Eye Disease Classification with Fine-Tuning and Evaluation

```
from tensorflow.keras.applications import ConvNeXtBase
from tensorflow.keras.models import Model
from tensorflow.keras.layers import GlobalAveragePooling2D, Dense
from tensorflow.keras.optimizers import Adam

# Load the ConvNeXt model pre-trained on ImageNet, excluding the top layer (for fine-
tuning)
base_model = ConvNeXtBase(weights='imagenet', include_top=False, input_shape=(224,
224, 3))

# Freeze the convolutional base (if you want to fine-tune only the top layers)
for layer in base_model.layers:
    layer.trainable = False

# Add custom top layers for our classification task
x = base_model.output
x = GlobalAveragePooling2D()(x)  # Reduce dimensions
x = Dense(1024, activation='relu')(x)
x = Dense(4, activation='softmax')(x)  # 4 output classes

# Create the final model
model = Model(inputs=base_model.input, outputs=x)

# Compile the model
model.compile(optimizer=Adam(), loss='categorical_crossentropy', metrics=['accuracy'])
# Train the model on the preprocessed data
history = model.fit(X_train, y_train, epochs=10, validation_data=(X_val, y_val),
batch_size=32)
# Evaluate model on validation data
val_loss, val_acc = model.evaluate(X_val, y_val)
print(f'Validation Accuracy: {val_acc * 100:.2f}%')
# Display training history
plt.plot(history.history['accuracy'], label='Train Accuracy')
plt.plot(history.history['val_accuracy'], label='Val Accuracy')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()
plt.show()
```

```
from sklearn.metrics import confusion_matrix, classification_report
import seaborn as sns

# Predict on validation set
y_pred_probs = model.predict(X_val)
y_pred = np.argmax(y_pred_probs, axis=1)
y_true = np.argmax(y_val, axis=1)

# Confusion Matrix
cm = confusion_matrix(y_true, y_pred)
plt.figure(figsize=(8, 6))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', xticklabels=classes, yticklabels=classes)
plt.title('Confusion Matrix')
plt.xlabel('Predicted')
plt.ylabel('True')
plt.show()

# Classification Report
print("Classification Report:")
print(classification_report(y_true, y_pred, target_names=classes))

# Correlation Matrix of Confusion Matrix
plt.figure(figsize=(8, 6))
correlation_matrix = np.corrcoef(cm)
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', xticklabels=classes,
yticklabels=classes)
plt.title("Correlation Matrix of Confusion Matrix")
plt.show()
def display_actual_vs_predicted(model, X_test, y_test, class_labels):
    # Predict the labels for the test set
    y_pred = model.predict(X_test)
    y_pred_classes = np.argmax(y_pred, axis=1)

    # Plot some of the images
    plt.figure(figsize=(10, 10))
    for i in range(9):

        plt.subplot(3, 3, i+1)
        plt.imshow(X_test[i])

        plt.title(f"Actual: {class_labels[np.argmax(y_test[i])]} \nPred:
{class_labels[y_pred_classes[i]]}")
        plt.axis('off')
    plt.show()

# Example: Assuming you have your test data loaded in X_test and y_test
class_labels = ['Cataract', 'Diabetic Retinopathy', 'Glaucoma', 'Normal']  # Modify as per your
classes
display_actual_vs_predicted(model, X_val, y_val, class_labels)
from sklearn.metrics import precision_recall_curve

fig, axes = plt.subplots(1, 4, figsize=(24, 6), sharey=True)
```

```
for class_idx in range(4):
    precision, recall, thresholds = precision_recall_curve(y_val[:, class_idx], y_pred_probs[:,
class_idx])

    ax = axes[class_idx]
    ax.scatter(thresholds, precision[:-1], label='Precision', alpha=0.6)
    ax.scatter(thresholds, recall[:-1], label='Recall', alpha=0.6)
    ax.set_title(f'Class: {classes[class_idx]}')
    ax.set_xlabel('Threshold')
    if class_idx == 0:
        ax.set_ylabel('Score')
    ax.grid(True)
    ax.legend()

plt.suptitle('Precision-Recall vs Threshold for All Classes', fontsize=16)
plt.tight_layout(rect=[0, 0, 1, 0.95])
plt.show()
model.save('/content/drive/MyDrive/convnext_eye_disease_model.h5')
from tensorflow.keras.applications import ConvNeXtBase
from tensorflow.keras.models import Model
from tensorflow.keras.layers import GlobalAveragePooling2D, Dense
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.callbacks import EarlyStopping

# Step 1: Load ConvNeXtBase with frozen base (already trained for 10 epochs)
base_model = ConvNeXtBase(weights='imagenet', include_top=False, input_shape=(224,
224, 3))
for layer in base_model.layers:
    layer.trainable = False

# Custom head
x = base_model.output
x = GlobalAveragePooling2D()(x)
x = Dense(1024, activation='relu')(x)
predictions = Dense(4, activation='softmax')(x)
model = Model(inputs=base_model.input, outputs=predictions)



# Initial training (already done, just showing for completeness)
model.compile(optimizer=Adam(learning_rate=1e-3), loss='categorical_crossentropy',
metrics=['accuracy'])
model.fit(X_train, y_train, epochs=5, validation_data=(X_val, y_val), batch_size=32)


# Step 2: Unfreeze last few layers for fine-tuning
for layer in base_model.layers[-20:]:  # You can adjust this number based on the architecture
    layer.trainable = True

# Step 3: Re-compile with a lower learning rate
model.compile(optimizer=Adam(learning_rate=1e-5), loss='categorical_crossentropy',
metrics=['accuracy'])
```

```
# Step 4: Fine-tune the model
history_finetune = model.fit(
    X_train, y_train,
    epochs=5,
    validation_data=(X_val, y_val),
    batch_size=32,
    callbacks=[EarlyStopping(patience=2, restore_best_weights=True)]
)
# Final evaluation on validation data
val_loss, val_acc = model.evaluate(X_val, y_val)
print(f'Final Validation Accuracy after Fine-Tuning: {val_acc * 100:.2f}%')

# Final evaluation on training data
train_loss, train_acc = model.evaluate(X_train, y_train)
print(f'Final Training Accuracy after Fine-Tuning: {train_acc * 100:.2f}%')
# Display training history
plt.plot(history.history['accuracy'], label='Train Accuracy')
plt.plot(history.history['val_accuracy'], label='Val Accuracy')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()
plt.show()
from sklearn.metrics import confusion_matrix, classification_report
import seaborn as sns

# Predict on validation set
y_pred_probs = model.predict(X_val)
y_pred = np.argmax(y_pred_probs, axis=1)
y_true = np.argmax(y_val, axis=1)

# Confusion Matrix
cm = confusion_matrix(y_true, y_pred)
plt.figure(figsize=(8, 6))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', xticklabels=classes, yticklabels=classes)
plt.title('Confusion Matrix')
plt.xlabel('Predicted')
plt.ylabel('True')
plt.show()

# Classification Report
print("Classification Report:")
print(classification_report(y_true, y_pred, target_names=classes))

# Correlation Matrix of Confusion Matrix
plt.figure(figsize=(8, 6))
correlation_matrix = np.corrcoef(cm)
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', xticklabels=classes,
yticklabels=classes)
plt.title("Correlation Matrix of Confusion Matrix")
plt.show()
model.save('/content/drive/MyDrive/convnext_eye_disease_finetunned_model.h5')
```

RESULT AND DISSCUSSION

The performance of both the VGG-19 and ConvNeXt models was rigorously evaluated using metrics such as accuracy, precision, recall, F1-score, and confusion matrices. After training, the VGG-19 model achieved a training accuracy of 97.48% and a validation accuracy of 88.51%. In contrast, the ConvNeXt model, after fine-tuning, achieved a final training accuracy of 95.70% and a superior validation accuracy of 90.64%.

The classification reports Table I and Table II summarize the class-wise performance. While both models demonstrated high precision and recall in detecting Diabetic Retinopathy and Cataract, ConvNeXt consistently outperformed VGG-19 across all metrics, especially in the classification of Normal and Glaucoma cases.

TABLE I
PERFORMANCE METRICS OF VGG-19 MODEL

Class	**Precision**	**Recall**	**F1-Score**
Cataract	0.87	0.95	0.91
Diabetic Retinopathy	0.99	1.00	0.99
Glaucoma	0.83	0.78	0.80
Normal	0.86	0.82	0.84
Overall Accuracy		88.51%	

TABLE II
PERFORMANCE METRICS OF CONVNEXT MODEL

Class	**Precision**	**Recall**	**F1-Score**
Cataract	0.91	0.92	0.92
Diabetic Retinopathy	0.99	0.98	0.99
Glaucoma	0.83	0.81	0.82
Normal	0.87	0.89	0.88
Overall Accuracy		90.64%	

The confusion matrices (Fig. 3. and Fig. 4) further illustrate the classification performance. The ConvNeXt model demonstrates improved generalization by minimizing misclassifications, particularly in complex cases like Glaucoma and Normal categories. These findings highlight ConvNeXt as the more robust model for ocular disease classification in this study.

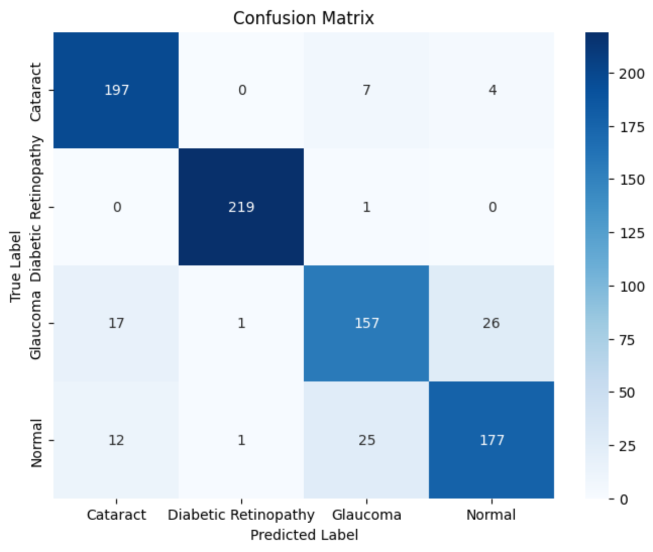


Fig. 3. Confusion Matrix for VGG-19 Model

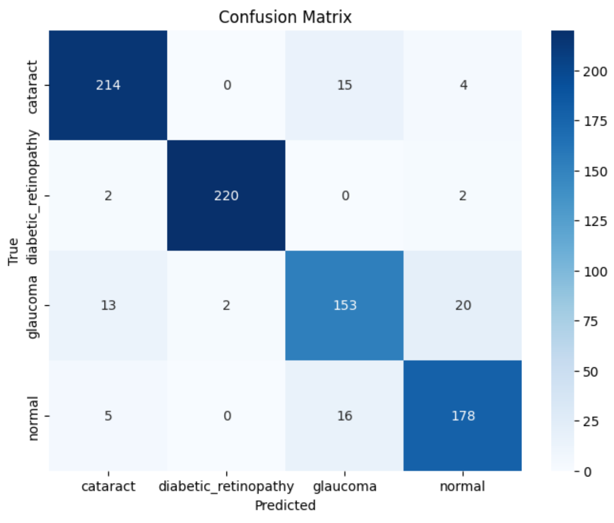
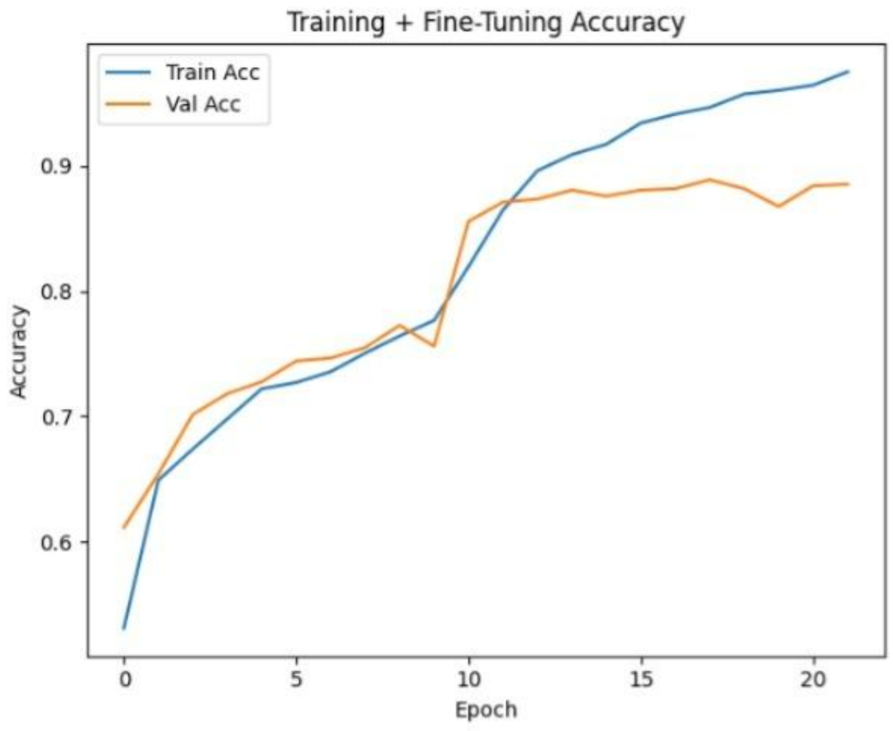
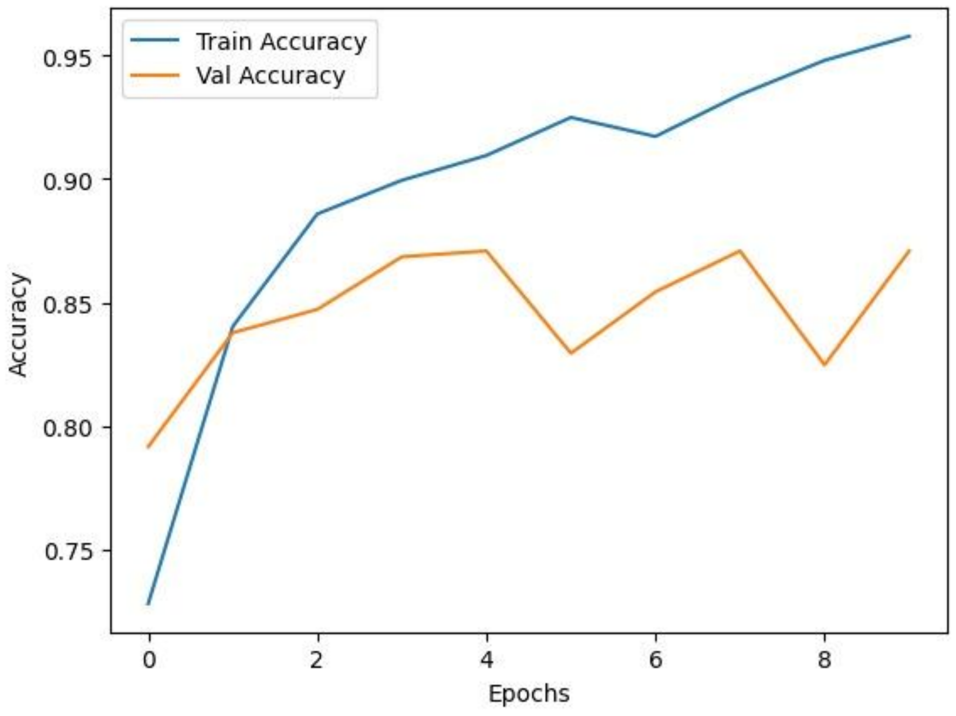


Fig. 4. Confusion Matrix for ConvNeXt Model

VGG19: Training and Validation Accuracy Graph Across Epochs



ConvNeXt: Training and Validation Accuracy Graph Across Epochs



To further interpret the model's decision-making process, Grad-CAM visualizations were generated for correctly and incorrectly classified fundus images. As shown in Figure 5, the highlighted regions represent the model's focus during classification. These visual explanations demonstrate that the ConvNeXt model generally concentrated on disease-relevant areas, reinforcing its reliability for medical diagnosis.

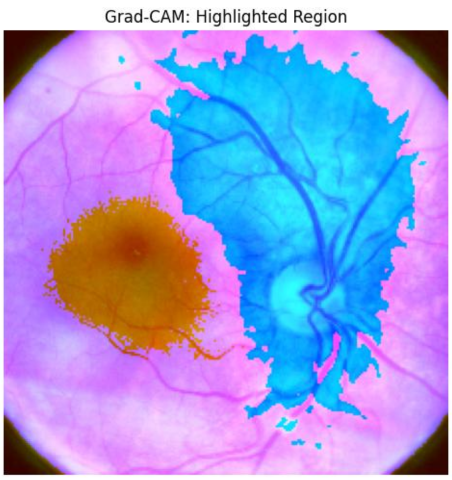


Fig. 5. Grad-CAM visualizations highlighting the image regions most influential to the model’s predictions.

The performance of both models is also demonstrated visually with sample actual vs. predicted images. The below image shows the corresponding actual and predicted labels for different disease classes. These images provide a clear comparison of the model’s predictions, reinforcing the evaluation metrics and accuracy results.