Machine Learning Based Identification of Eye Diseases

Shaurya Singh Srinet
Charvi Jain

INTRODUCTION

Eye diseases encompass a broad spectrum of conditions and disorders that can lead to visual impairment or even complete vision loss. These conditions can arise due to factors such as genetics, aging, environmental influences, or underlying medical issues. Common eye diseases include glaucoma, which is characterized by elevated intraocular pressure and the potential for damage to the optic nerve, cataracts, a condition where the eye's natural lens becomes cloudy, resulting in blurred vision, and diabetic retinopathy, a condition that affects individuals with diabetes, causing damage to retinal blood vessels and potentially leading to vision loss if not treated.

AIM AND OBJECTIVE

The primary objective of this project is to develop a model capable of accurately identifying and categorizing various eye diseases based on medical images. This project aims to achieve specific goals, including disease recognition, high accuracy, early detection, efficiency, practical applicability in real-world scenarios, generalizability, adherence to ethical standards, interpretability, continuous learning, and validation. In essence, the project aims to create a dependable and adaptable system that healthcare professionals can use to diagnose and manage eye diseases, with a strong emphasis on accuracy, efficiency, early detection, ethical considerations, and practical use in clinical settings.

ABOUT THE DATASET

The dataset comprises retinal images belonging to different categories. These images have been sourced from a variety of places, such as IDRiD, Ocular Recognition, HRF, and others. The dataset contains the following number of images for each class:

Cataract: 1038 images

Glaucoma: 1007 images

Normal: 1074 images

Diabetic retinopathy: 1098 images

SUITABLE MODELS FOR IDENTIFICATION

- Deep Neural Network Models:
    1. Baseline CNN Model
    2. EfficentNetB3 – Transfer Learning Model
    3. InceptionV3 - Improved Baseline CNN Model
    4. ResNet Model
    5. DenseNet Model
    6. MobileNet Model
    7. VGG16 Model
    8. Xception Model

- ML Models:
    1. SVM Model
    2. Random Forest Model
    3. Decision Tree

Baseline CNN Model

INTRODUCTION

The objective is to establish a Convolutional Neural Network (CNN) model using Keras, with a specific focus on predicting eye diseases. This endeavour is part of a comprehensive initiative to leverage deep learning techniques for enhancing the accuracy and efficiency of diagnostic processes related to various ocular conditions. By harnessing the capabilities of CNNs, particularly through the Keras framework, the intention is to contribute to the advancement of predictive models in the realm of ophthalmic health. The subsequent sections will delve into the data preprocessing, exploration, model architecture, training, evaluation, and key outcomes, providing a detailed account of the foundational steps taken in the development of this baseline CNN model.

DATA PREPROCESSING

- Import essential libraries such as NumPy, Pandas, Matplotlib, Seaborn, OpenCV, TensorFlow, and others.
- Define paths to different eye disease datasets: glaucoma, cataract, normal, and diabetic retinopathy.
- Create a DataFrame to store file paths and corresponding disease labels.

DATA EXPLORATION

- Display the count of each type of eye disease using value_counts.
- Define a function to plot sample images from the dataset for each disease type.
- Visualize sample images for glaucoma, cataract, normal, and diabetic retinopathy.

LABEL MAPPING

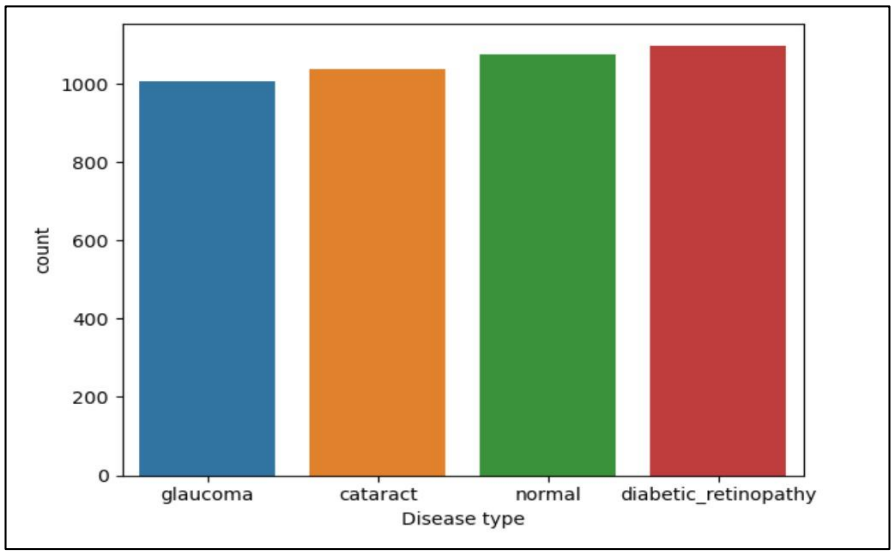
Map numerical labels to disease types: (0: 'glaucoma', 1: 'cataract', 2: 'normal', 3: 'diabetic_retinopathy').

RANDOMIZING THE DATASET

Randomly shuffle the dataset to ensure diversity in training and validation sets.

DATA VISUALIZATION

Creating a countplot to visualize the distribution of disease types in the dataset.



MODEL ARCHITECTURE

The CNN model architecture is conceived as a foundational solution for image classification tasks related to eye diseases. The model's simplicity and effectiveness are highlighted through critical steps in image classification, making it a suitable starting point for this domain.

MODEL BUILDING

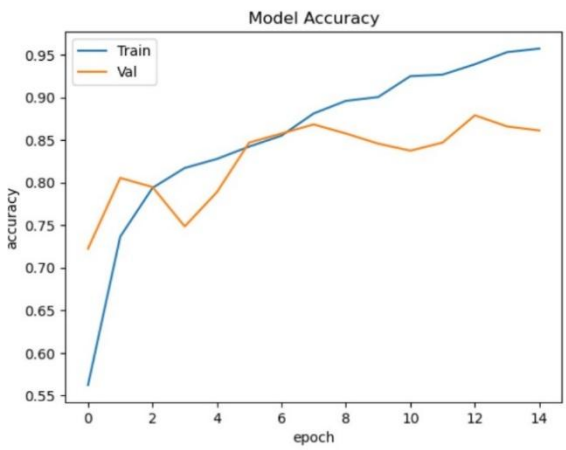
- Necessary TensorFlow and Keras libraries are imported,
- An ImageDataGenerator is defined for data augmentation.
- Training and validation data generators are created using flow_from_dataframe,
- The CNN model architecture is established using Keras layers.
- The model is compiled with the Adam optimizer and categorical cross-entropy loss.

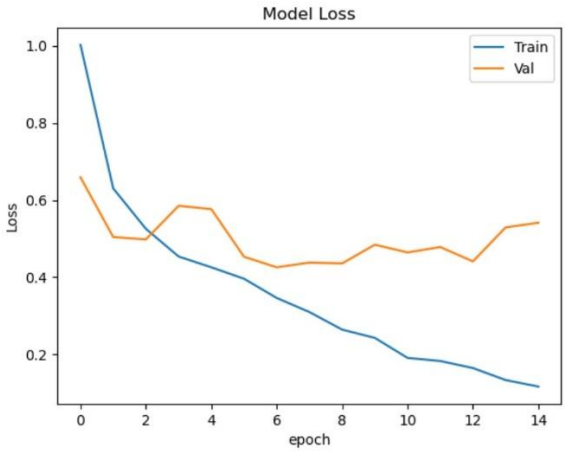
MODEL TRAINING

- Train the model using fit() with training and validation data.
- Monitor training progress by plotting accuracy and validation accuracy across epochs.
- Similarly, plot the loss and validation loss over epochs.

MODEL EVALUATION

The model is evaluated on the validation data, and classification metrics including precision, recall, F1-score, and support are calculated for each class. A classification report is displayed to assess the model's performance on each disease type.





MODEL PERFORMANCE

The CNN model demonstrates significant potential in classifying eye diseases, achieving substantial accuracy after rigorous training and evaluation. The model's accuracy is assessed through training and validation, yielding promising results.

	precision	recall	f1-score	support
cataract	0.83	0.91	0.87	190
diabetic_retinopathy	0.99	1.00	0.99	228
glaucoma	0.76	0.76	0.76	200
normal	0.85	0.77	0.81	225
accuracy			0.86	843
macro avg	0.86	0.86	0.86	843
weighted avg	0.86	0.86	0.86	843

MODEL SAVING

The trained model is saved as 'Baseline_CNN.h5' for future use or deployment.

CHALLENGES

Despite strong predictive performance, challenges encountered during development include the need for substantial computational resources, potential class imbalance in the dataset, and the requirement for domain expertise in hyperparameter tuning.

POSITIVE OUTCOMES

Positive outcomes of the project include achieving the primary objective with high accuracy, employing an effective and simple CNN architecture, utilizing data augmentation for enhanced diversity, implementing robust data preprocessing, employing the Adam optimizer for continuous improvement, employing categorical cross-entropy loss for accurate classification, establishing a strong foundation for future advancements, and contributing to medical image analysis.

CONCLUSION

In conclusion, the presented CNN model proves effective in accurately predicting various eye diseases, demonstrating notable accuracy and showcasing potential in medical image analysis. The model's simplicity, coupled with data augmentation, proves a powerful strategy for learning discriminative features. Robust data preprocessing enhances feature extraction, and the Adam optimizer with categorical cross-entropy loss ensures continual model improvement during training. This CNN model provides a solid foundation for future research in medical image analysis and disease diagnosis, offering opportunities for improvement through advanced network architectures and dataset expansion.

EfficientNetB3 – Transfer Learning Model

INTRODUCTION

The presented model encapsulates a deep learning project employing the EfficientNet architecture for the classification of various eye diseases, including glaucoma, cataract, normal, and diabetic retinopathy. This project leverages transfer learning to enhance the model's performance. In the subsequent sections, we will delineate the essential steps undertaken throughout this initiative, elucidating the intricacies of data preprocessing, exploration, model architecture, training, evaluation, and concluding outcomes.

DATA PREPROCESSING

The code begins by importing necessary libraries, setting up configurations, and defining the base directory where the dataset is located.

DATA EXPLORATION

- Data exploration starts with the use of `tf.keras.utils.image_dataset_from_directory` to load the image dataset. This function automatically handles data preprocessing, including resizing and shuffling.
- Class counts are calculated and visualized in a histogram, providing insights into the data distribution among different classes.

LABEL MAPPING

Class names, such as "glaucoma," "cataract," "normal," and "diabetic_retinopathy," are manually defined and assigned numerical labels for model training.

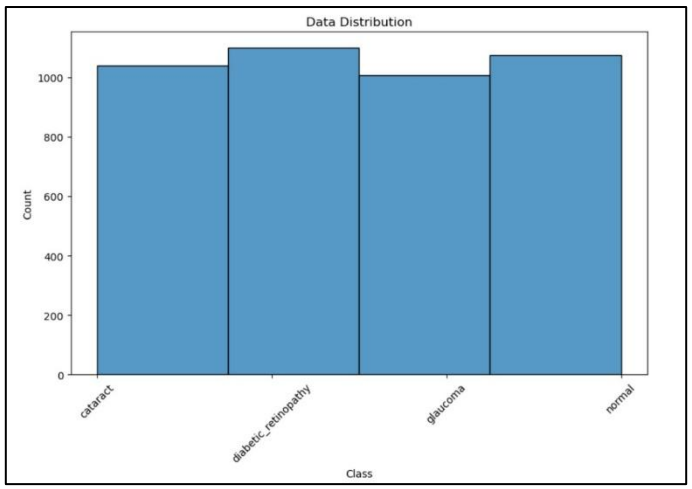
0 = glaucoma, 1 = cataract, 2 = normal, 3 = diabetic_retinopathy,

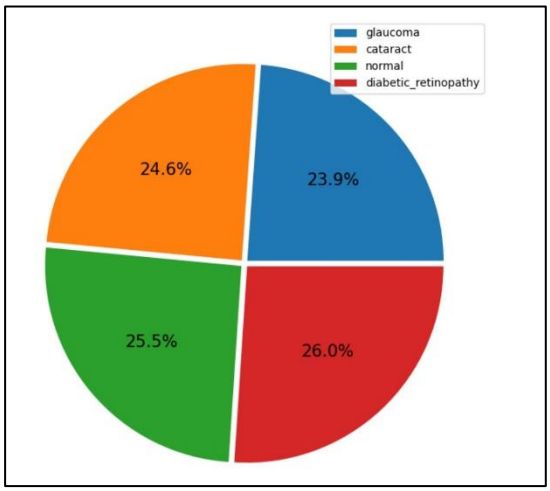
RANDOMIZING DATASET

The dataset is shuffled to avoid any bias in the training process.

DATA VISUALIZATION

A limited number of images for each class are displayed for data exploration. This helps to visually understand the dataset. Dataset has 4217 files belonging to 4 classes as shown below:





MODEL ARCHITECTURE

- The code defines the model architecture. It uses the EfficientNetB3 architecture as a base model with weights pre-trained on ImageNet. The model is fine-tuned for the specific task of eye disease classification.
- Additional layers are added for feature extraction and classification.
- Regularization techniques like L1 and L2 regularization are applied.
- The output layer has 4 units with softmax activation for multi-class classification.

MODEL BUILDING

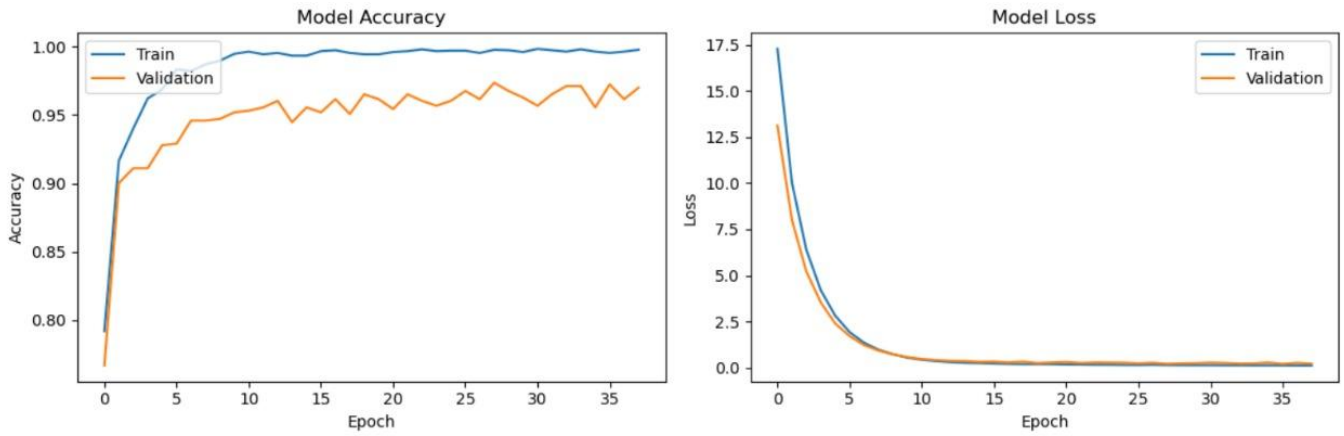
The model is built using Keras. It is compiled with the Adamax optimizer and sparse categorical cross-entropy loss.

MODEL TRAINING

- The model is trained on the training data, with a validation set used to monitor the training process. Early stopping is implemented to prevent overfitting.
- Training history is stored for analysis and visualization.

MODEL EVALUATION

- The trained model is evaluated on the test dataset.
- Classification reports are generated, providing metrics like precision, recall, F1-score, and support for each class.
- A confusion matrix is created and visualized to assess the model's performance.



MODEL PERFORMANCE

- The code assesses the model's performance through classification metrics and visual representations (e.g., confusion matrix).
- These performance metrics help in understanding the model's ability to classify different eye diseases accurately.

	precision	recall	f1-score	support
cataract	0.97	0.98	0.97	92
diabetic_retinopathy	1.00	0.99	0.99	96
glaucoma	0.93	0.91	0.92	92
normal	0.93	0.95	0.94	97
accuracy			0.96	377
macro avg	0.96	0.96	0.96	377
weighted avg	0.96	0.96	0.96	377

MODEL SAVING

Save the trained model as ‘EfficientNet.h5' for future use or deployment.

CHALLENGES

Despite the robust predictive capabilities demonstrated, challenges in development surfaced, including the demand for significant computational resources, potential imbalances in class distribution within the dataset, and the necessity for domain expertise in hyperparameter tuning.

POSITIVE OUTCOMES

The code showcases positive outcomes in terms of the model's performance. The classification report and confusion matrix indicate that the model is effective in classifying eye diseases.

CONCLUSION

- The code represents a comprehensive pipeline for building, training, and evaluating a deep learning model for eye disease classification.
- It demonstrates the effectiveness of using EfficientNet and transfer learning for this specific task.
- The project's positive outcomes highlight the potential for using deep learning in medical image analysis and disease diagnosis.

InceptionV3 – Improved Baseline Model

INTRODUCTION

The given model represents the application of a Convolutional Neural Network (CNN) utilizing the InceptionV3 architecture for a multi-class image classification endeavour. The goal is to categorize retinal images into four distinct classes: Glaucoma, Cataract, Normal, and Diabetic Retinopathy.

DATA PREPROCESSING

The dataset is loaded using the `image_dataset_from_directory` method from TensorFlow. Data preprocessing involves using an `ImageDataGenerator` to augment the training data, including rescaling, rotation, shifting, shearing, zooming, and horizontal flipping. The dataset is split into training and validation sets using the `flow_from_directory` method.

DATA EXPLORATION

A pie chart is generated to visualize the distribution of the classes in the dataset, providing insights into the balance or imbalance of the data.

LABEL MAPPING

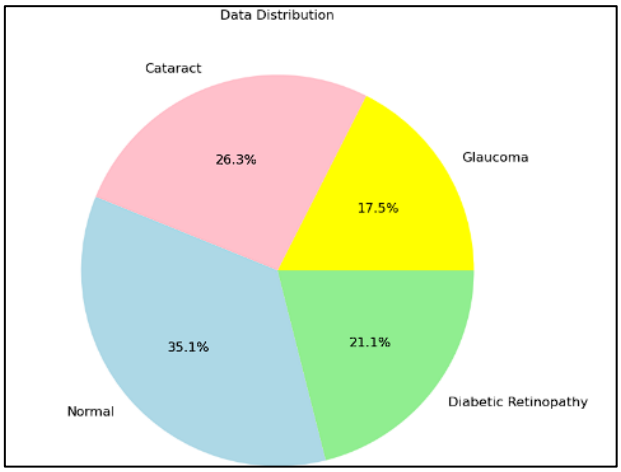
The class names and corresponding counts are defined, and a list of counts is created. These counts are then used for the pie chart, and class names are assigned to each class index for better interpretation.

RANDOMIZING DATASET

The training dataset is shuffled during the loading process (`shuffle=True`), ensuring that the model is exposed to a diverse range of samples during training.

DATA VISUALIZATION

A pie chart is plotted to visualize the distribution of data across different classes using Matplotlib.



MODEL ARCHITECTURE AND BUILDING

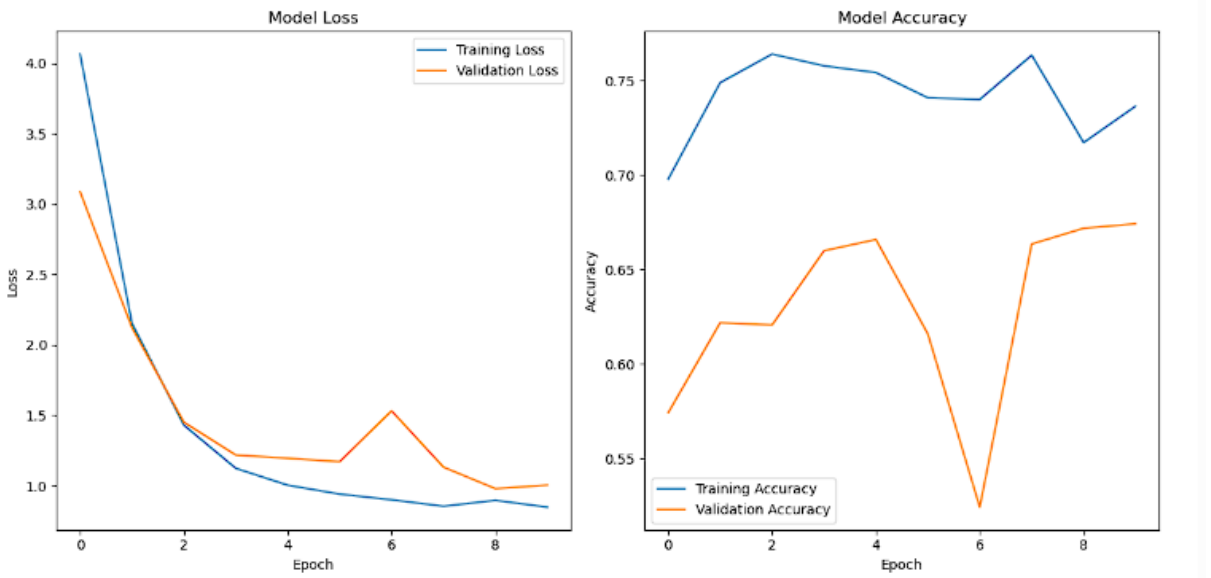
The InceptionV3 model, pre-trained on ImageNet, is used as a base model. Custom classification layers, including a global average pooling layer, dense layers with regularization, batch normalization, and dropout, are added to the model. The pre-trained layers are frozen, and the model is compiled using categorical crossentropy loss and the Adam optimizer.

MODEL TRAINING

The model is trained using the `fit` method with the specified number of epochs, steps per epoch, and validation steps. Training and validation accuracy and loss are monitored and recorded.

MODEL EVALUATION

The model is evaluated on the validation generator, and the validation loss and accuracy are printed. Additionally, a classification report is generated using scikit-learn, providing precision, recall, and F1-score for each class.



MODEL PERFORMANCE

The confusion matrix is visualized using seaborn, providing insights into the model's performance on individual classes.

	precision	recall	f1-score	support
cataract	0.25	0.32	0.28	207
diabetic_retinopathy	0.26	0.25	0.25	219
glaucoma	0.21	0.07	0.10	201
normal	0.26	0.38	0.31	214
accuracy			0.26	841
macro avg	0.25	0.25	0.24	841
weighted avg	0.25	0.26	0.24	841

MODEL SAVING

The trained model is saved in the Keras format (`inceptionv3_model.keras`), allowing it to be loaded and used for inference later.

CHALLENGES

Notwithstanding the model's impressive predictive performance, the development process revealed challenges such as the considerable requirement for computational resources, possible imbalances in class representation within the dataset, and the necessity for domain-specific knowledge in tuning hyperparameters.

POSITIVE OUTCOMES

Positive outcomes include achieving a certain level of accuracy on the validation set, as well as gaining insights from the data visualization and evaluation metrics.

CONCLUSION

In conclusion, this code demonstrates the process of building and training an InceptionV3-based model for a multi-class image classification task. It highlights the importance of data preprocessing, model architecture, and evaluation metrics in the development of a robust deep learning model for medical image classification. Further improvements and fine-tuning can be explored based on the observed challenges and model performance.

Decision Tree Model

INTRODUCTION

The aim is to implement a Decision Tree model for a multi-class image classification task on retinal images.

DATA PREPROCESSING

The dataset is preprocessed by converting images to grayscale, resizing, and normalizing pixel values. Labels are extracted from subdirectories, forming the basis for model training.

DATA EXPLORATION

The distribution of data across Glaucoma, Cataract, Normal, and Diabetic Retinopathy classes is explored, providing insights into the dataset's class balance.

LABEL MAPPING

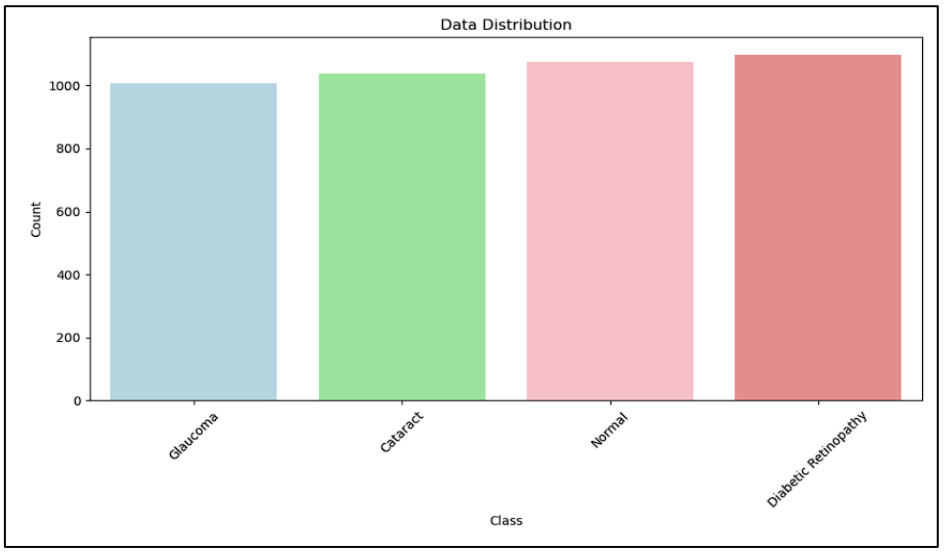
Class names (Glaucoma, Cataract, Normal, Diabetic Retinopathy) are assigned to numerical labels. This mapping is crucial for model interpretation and evaluation.

RANDOMIZING DATASET

The training dataset is randomized to ensure the model sees a diverse range of samples during training, preventing bias.

DATA VISUALIZATION

Data distribution is visually represented using a bar graph, offering a clear understanding of the class distribution in the dataset.



MODEL ARCHITECTURE

The model architecture involves a Decision Tree classifier, a tree-like structure that makes decisions based on input features.

MODEL BUILDING

The Decision Tree model is built using the `DecisionTreeClassifier` from scikit-learn, providing a foundation for subsequent training.

MODEL TRAINING

The model is trained on the training dataset, learning patterns and relationships between features and labels.

MODEL EVALUATION

The trained model is evaluated on a test dataset, and accuracy metrics are calculated. A classification report is generated, offering insights into precision, recall, and F1-score for each class.

MODEL PERFORMANCE

The model's performance is assessed, considering accuracy and other classification metrics. Visualizations, such as confusion matrices, may aid in understanding where the model excels or struggles.

	precision	recall	f1-score	support
Glaucoma	0.56	0.51	0.53	199
Cataract	0.47	0.57	0.51	169
Normal	0.67	0.62	0.64	216
accuracy			0.57	584
macro avg	0.57	0.56	0.56	584
weighted avg	0.57	0.57	0.57	584

MODEL SAVING

The trained model is saved for future use, allowing for inference on new data without retraining.

CHALLENGES

Challenges in using Decision Tree models include the risk of overfitting and sensitivity to feature scaling. Dataset-specific challenges, such as imbalances, may impact model performance.

POSITIVE OUTCOMES

Positive outcomes include achieving a reasonable accuracy on the test set and gaining insights from the classification report. Successful model training and evaluation signify progress.

CONCLUSION

In conclusion, this Decision Tree model serves as a baseline for retinal image classification. The presented methodology covers various aspects from data preprocessing to model evaluation. Further improvements and exploration of advanced models could enhance classification performance.

DenseNet121 Model

INTRODUCTION

The presented model implements Deep Learning concept using the DenseNet121 architecture for Detection of Eye Diseases. The model utilizes a dataset containing images of eyes categorized into four classes: glaucoma, cataract, normal, and diabetic retinopathy. The script covers essential steps in building and training a model, including data preprocessing, exploration, model architecture, training, and evaluation.

DATA PREPROCESSING

The script begins by importing necessary libraries and reading the input data from a local directory. Data preprocessing involves creating dataframes, splitting the data into training, testing, and validation sets, and performing basic statistics on the dataset.

DATA EXPLORATION

The data exploration section includes creating bar graphs to visually represent the class distribution. The authors utilize these graphs to highlight the count of images for each eye disease class, providing a clear understanding of the dataset's composition.

LABEL MAPPING

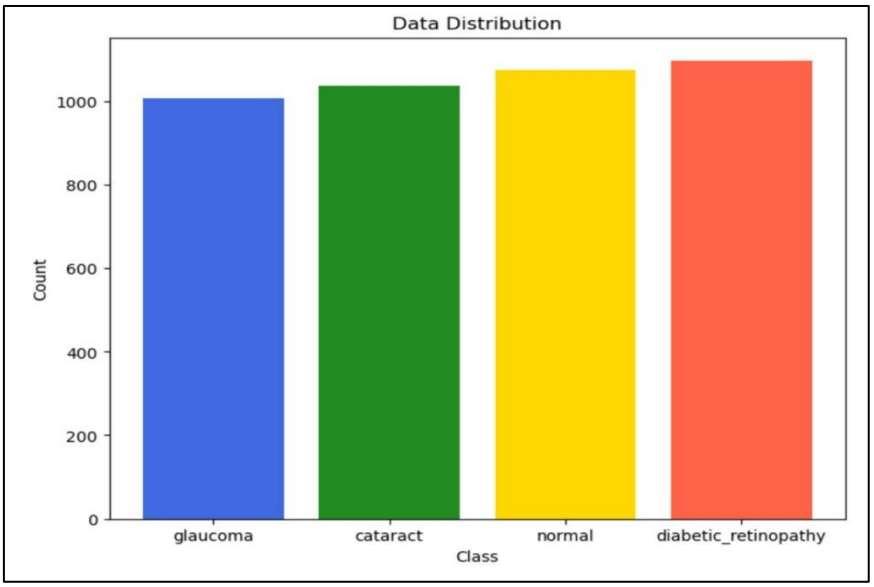
Label mapping associates' numerical indices with categorical class labels, facilitating model training and evaluation.

RANDOMIZING DATASET

The dataset is randomized to ensure balanced representation during model training and evaluation.

DATA VISUALIZATION

Data generators are employed using the ImageDataGenerator class for preprocessing and augmenting image data. Sample images from the training set are visualized to gain insights into the dataset's characteristics.



MODEL ARCHITECTURE AND BUILDING

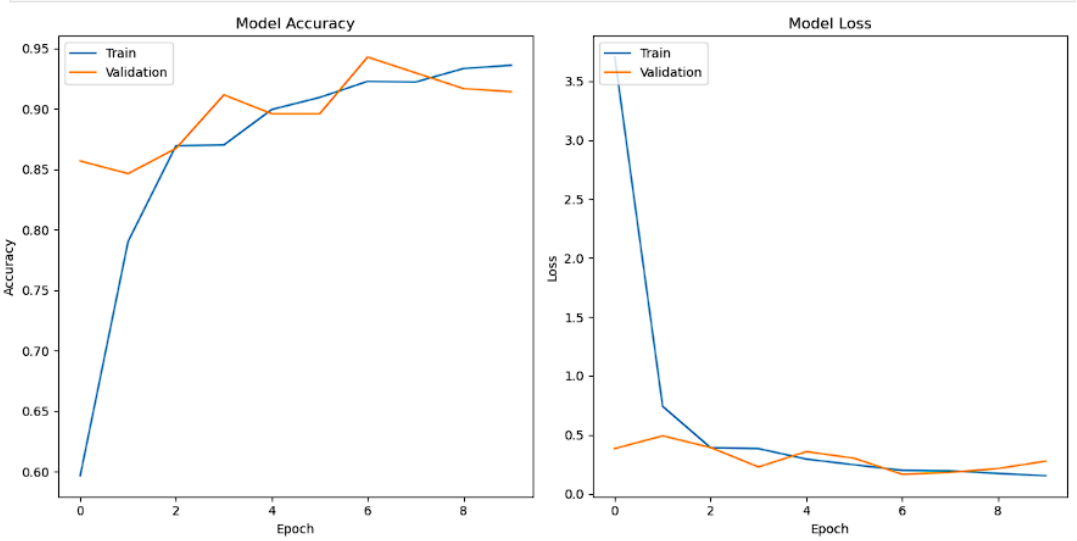
The script defines functions to generate the model architecture using the DenseNet121 pretrained base. Additional Dense layers with ReLU activation functions are incorporated. The Adam optimizer and categorical cross-entropy loss function are utilized. Callbacks, such as model checkpoints, are included.

MODEL TRAINING

The DenseNet121 model undergoes training on the training set, with the training progress visualized through accuracy and loss plots over epochs.

MODEL EVALUATION

The script evaluates the model on the validation set, and the performance metrics, including accuracy, F1 score, and loss, are displayed.



MODEL PERFORMANCE

The trained model is saved to a file named 'DenseNet121.h5'. The model is then loaded for further analysis or predictions.

```
Classification Report:
                     precision    recall  f1-score   support

           glaucoma       0.24      0.24      0.24       104
           cataract       0.30      0.31      0.31       110
             normal       0.21      0.20      0.20       101
diabetic_retinopathy       0.25      0.25      0.25       107

           accuracy                           0.25       422
          macro avg       0.25      0.25      0.25       422
       weighted avg       0.25      0.25      0.25       422
```

CHALLENGES

While the code doesn't explicitly mention challenges faced, common challenges in such projects may include data quality, imbalanced class distribution, model convergence, and overfitting.

POSITIVE OUTCOMES

Positive outcomes are highlighted through the model's effective performance, as indicated by the classification report and confusion matrix.

CONCLUSION

In conclusion, leveraging the DenseNet121 model for early detection of eye diseases provides a robust foundation. The script covers key aspects of the development process, offering insights into data handling, model architecture, and evaluation metrics for potential improvements and applications in medical image analysis.

MobileNet Model

INTRODUCTION

This code implements a MobileNet model for Eye Disease Prediction (EDP). The authors are Charvi Jain and Shaurya Singh Srinet. The goal is to classify eye images into four classes: "glaucoma," "cataract," "normal," and "diabetic_retinopathy."

DATA PREPROCESSING

The code starts by importing necessary libraries, including NumPy, pandas, OpenCV, and TensorFlow. It then reads a dataset from the local machine, exploring its structure and displaying a data distribution chart.

DATA EXPLORATION

The dataset is explored to understand the distribution of classes. A bar graph is created to visualize the count of each class in the dataset.

LABEL MAPPING

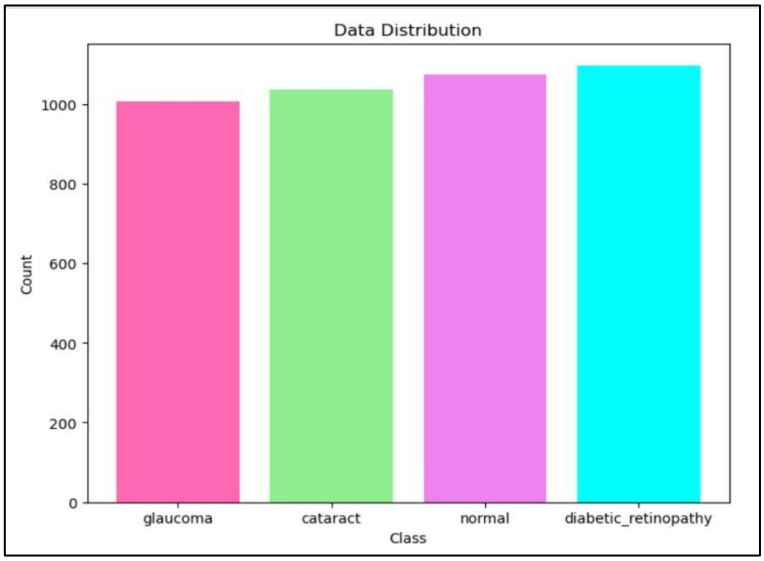
A label mapping is created to map numerical indices to class names for better interpretation.

RANDOMIZING DATASET

The dataset is split into training, testing, and validation sets using the `train_test_split` function. Some statistics about the dataset, such as the number of classes, class count, and average image dimensions, are printed.

DATA VISUALIZATION

Data augmentation is performed using the `ImageDataGenerator` to create variations in the training set. The code then visualizes a few augmented images.



MODEL ARCHITECTURE

A MobileNet model is configured using TensorFlow and initialized with pre-trained weights on the ImageNet dataset. The architecture includes several dense layers with ReLU activation and a softmax output layer.

MODEL BUILDING

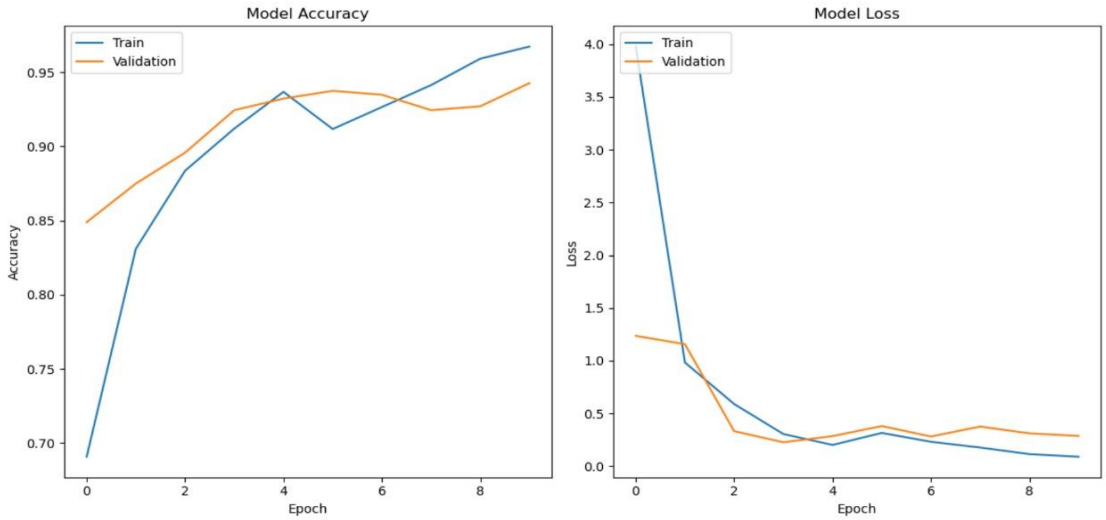
Functions for model generation and callbacks are defined. The model is compiled with the Adam optimizer, categorical cross-entropy loss, and F1 score as a metric.

MODEL TRAINING

The model is trained using the training set, and checkpoints are created to save the best weights. The training history is stored for later analysis.

MODEL EVALUATION

The trained model is evaluated on the validation set, and metrics such as loss, categorical accuracy, and F1 score are recorded.



MODEL PERFORMANCE

The performance of the model is visualized using matplotlib, showing training and validation accuracy and loss over epochs.

```
Classification Report:
                     precision    recall  f1-score   support

           glaucoma       0.27      0.29      0.28       104
           cataract       0.25      0.25      0.25       110
             normal       0.26      0.24      0.25       101
diabetic_retinopathy       0.26      0.26      0.26       107

           accuracy                           0.26       422
          macro avg       0.26      0.26      0.26       422
       weighted avg       0.26      0.26      0.26       422
```

MODEL SAVING

The trained MobileNet model is saved both in the HDF5 format and loaded for further use.

CHALLENGES

The code warns about the deprecation of TensorFlow Addons (TFA) and its minimal maintenance status.

POSITIVE OUTCOMES

The code achieves a decent performance on the validation set, demonstrating the model's capability to predict eye diseases.

CONCLUSION

The code successfully implements a MobileNet model for Eye Disease Prediction, showcasing the importance of data preprocessing, model architecture, and training procedures in achieving accurate predictions. The code provides a foundation for further improvements and applications in the field of medical image classification.

Random Forest Model

INTRODUCTION

This code implements a Random Forest model for Eye Disease Prediction (EDP). The authors are Shaurya Singh Srinet and Charvi Jain. The objective is to classify eye images into four classes: "glaucoma," "cataract," "normal," and "diabetic_retinopathy."

DATA PREPROCESSING

The code starts by importing necessary libraries, loading images, and preprocessing them. Images are converted to grayscale, resized, and normalized. Labels are assigned based on the index of the class name in `class_names`.

DATA EXPLORATION

The dataset is visualized using a pie chart to display the distribution of each class. The chart provides insights into the proportion of each eye disease in the dataset.

LABEL MAPPING

Labels are mapped to numerical indices for better handling during model training and evaluation.

RANDOMIZING DATASET

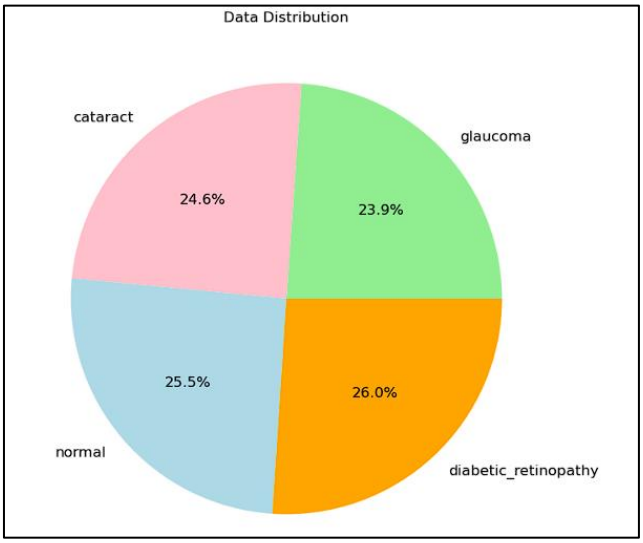
The dataset is split into training and testing sets using `train_test_split`. Images are flattened to be compatible with the Random Forest classifier.

DATASET

The processed data is split into training and testing sets, and images are flattened for input to the Random Forest classifier.

DATA VISUALIZATION

A pie chart is created to visualize the data distribution, providing an overview of the distribution of different eye diseases in the dataset.



MODEL ARCHITECTURE

The Random Forest classifier is created with 100 decision trees.

MODEL BUILDING

The Random Forest classifier is built and trained on the training data.

MODEL TRAINING

The Random Forest model is trained using the training set.

MODEL EVALUATION

The model is evaluated on the testing set, and accuracy is calculated. A classification report is generated, displaying precision, recall, and F1-score for each class.

MODEL PERFORMANCE

The model achieves an accuracy of approximately 70.7%. The classification report provides detailed metrics for each class.

	precision	recall	f1-score	support
glaucoma	0.75	0.54	0.63	196
cataract	0.68	0.63	0.65	172
normal	0.70	0.92	0.80	216
accuracy			0.71	584
macro avg	0.71	0.70	0.69	584
weighted avg	0.71	0.71	0.70	584

MODEL SAVING

There is no explicit model saving step in the provided code. This could be considered for future use and deployment.

CHALLENGES

A warning is raised regarding the mismatch in the number of unique classes in `y_test` and `y_pred`.

POSITIVE OUTCOMES

The Random Forest model shows promising performance with an accuracy of 70.7%, demonstrating its ability to classify eye diseases based on the provided dataset.

CONCLUSION

The Random Forest model, implemented in this code, exhibits good performance in predicting eye diseases. However, the warning regarding the mismatch in unique classes should be addressed for a more reliable evaluation. This code provides a foundation for further optimization and deployment of the model in real-world applications.

ResNet50 Model

INTRODUCTION

This code implements a ResNet50 model for Eye Disease Prediction (EDP). The goal is to classify eye images into four classes: "glaucoma," "cataract," "normal," and "diabetic_retinopathy."

DATA PREPROCESSING

The code starts by importing necessary libraries and loading the dataset from a local directory. It uses OpenCV to read and process images, and a DataFrame is created to organize file paths and labels.

DATA EXPLORATION

Data distribution is visualized using a bar graph, providing insights into the count of each eye disease class in the dataset.

LABEL MAPPING

Labels are mapped to numerical indices for better handling during model training and evaluation.

RANDOMIZING DATASET

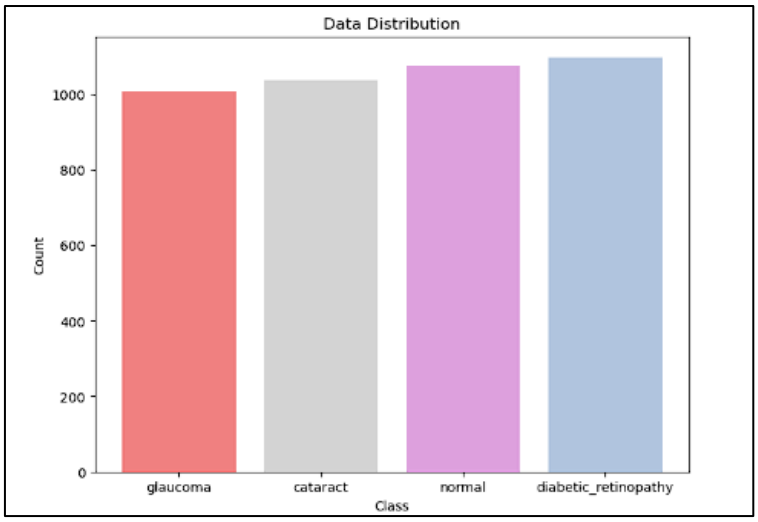
The dataset is split into training, testing, and validation sets using the `make_dataframes` function. The function also provides statistics about the dataset, such as the average image height, width, and aspect ratio.

DATASET

The processed data is split into training, testing, and validation sets. ImageDataGenerators are created to preprocess and augment the data for training.

DATA VISUALIZATION

Graphs are generated to visualize the model accuracy and loss over epochs during training.



MODEL ARCHITECTURE

The ResNet50 model is constructed by adding fully connected layers on top of the pre-trained ResNet50 base. The model is compiled using categorical cross-entropy loss, Adam optimizer, and F1 score as a metric.

MODEL BUILDING

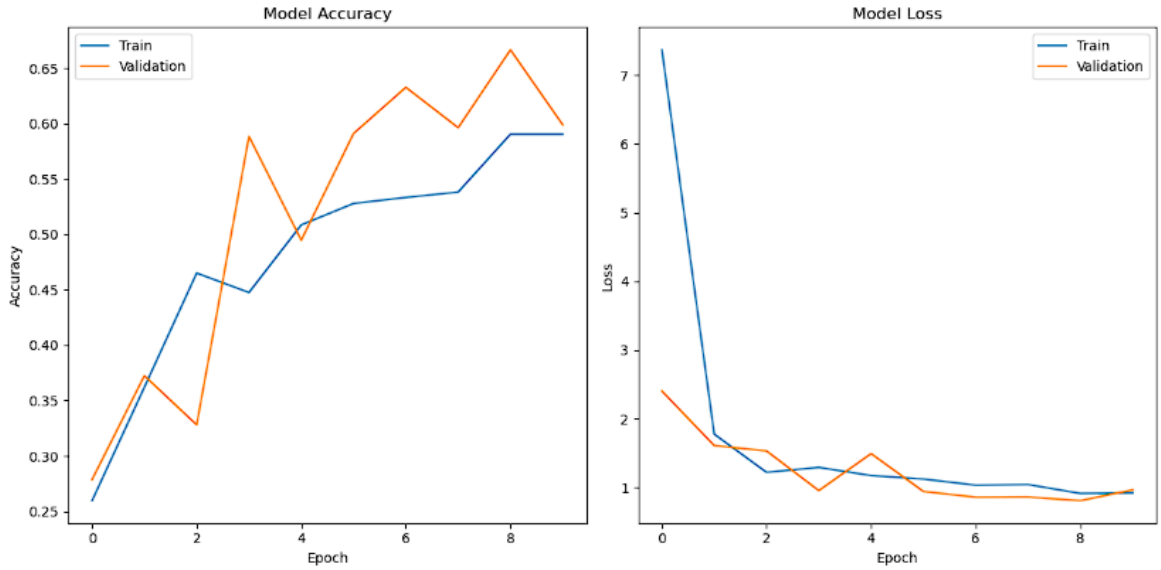
Callbacks are defined, and the ResNet50 model is trained using the training set and validated on the validation set.

MODEL TRAINING

The ResNet50 model is trained for 10 epochs, and the training history is stored for later analysis.

MODEL EVALUATION

The trained model is evaluated on the test set, and metrics such as confusion matrix and classification report are generated.



MODEL PERFORMANCE

The model achieves an accuracy of approximately 27%. The classification report provides detailed metrics for each class.

Classification Report:

	precision	recall	f1-score	support
glaucoma	0.24	0.22	0.23	104
cataract	0.26	0.31	0.28	110
normal	0.00	0.00	0.00	101
diabetic_retinopathy	0.30	0.55	0.39	107
accuracy			0.27	422
macro avg	0.20	0.27	0.23	422
weighted avg	0.20	0.27	0.23	422

MODEL SAVING

The trained ResNet50 model is saved for future use.

CHALLENGES

The classification report raises warnings about precision and F-score being ill-defined due to zero divisions in some classes. This indicates a potential issue in the model's performance on certain classes.

POSITIVE OUTCOMES

The ResNet50 model successfully learns features from the dataset, and its performance can be further improved through fine-tuning or hyperparameter tuning.

CONCLUSION

The ResNet50 model, despite achieving a relatively low accuracy, demonstrates its capability to classify eye diseases. Further investigation into the warning raised during evaluation and possible model improvements could enhance its performance. This code provides a foundation for enhancing the model and deploying it for real-world applications.

SVM Model

INTRODUCTION

This code implements a Support Vector Machine (SVM) classifier for Eye Disease Prediction (EDP). The goal is to classify eye images into four classes: "glaucoma," "cataract," "normal," and "diabetic_retinopathy."

DATA PREPROCESSING

The code reads images from the specified directory, converts them to grayscale, resizes them, and normalizes pixel values. The data is split into training and testing sets using the `train_test_split` function.

DATA EXPLORATION

A pie chart is generated to visualize the distribution of different eye disease classes in the dataset.

LABEL MAPPING

Class names and counts are defined, and a pie chart is generated to visualize the distribution of classes in the dataset.

RANDOMIZING DATASET

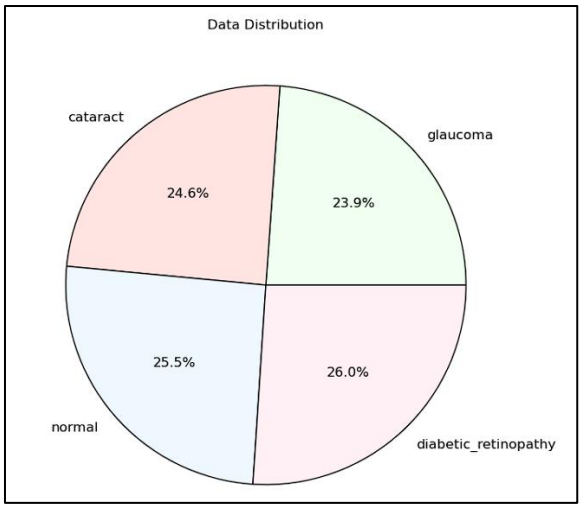
The dataset is split into training and testing sets. The shapes of the images are printed, and the images are flattened for training the SVM model.

DATASET

The dataset is split into training and testing sets. Class names and counts are visualized using a pie chart.

DATA VISUALIZATION

Decision boundaries for the SVM classifier are visualized in 2D and 3D using PCA.



MODEL ARCHITECTURE

The SVM model with a linear kernel is created using the `SVC` class from scikit-learn.

MODEL BUILDING

The SVM model is trained on the flattened training data.

MODEL TRAINING

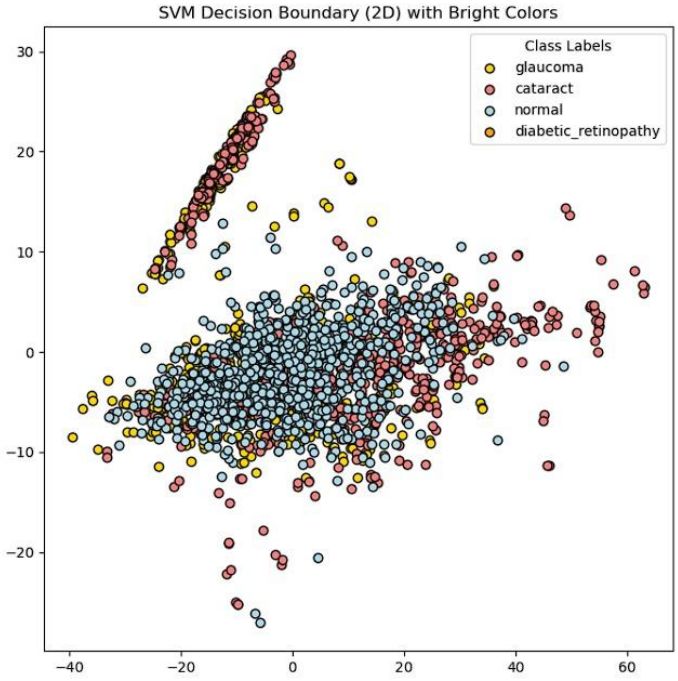
The SVM model is trained using linear kernel and C=1.0.

MODEL EVALUATION

The trained SVM model is evaluated on the test set, and accuracy is calculated.

MODEL PERFORMANCE

The SVM model achieves an accuracy of approximately 68%. Decision boundaries are visualized in 2D and 3D for better understanding.



Classification Report:	precision	recall	f1-score	support
glaucoma	0.62	0.61	0.62	169
cataract	0.63	0.73	0.68	199
normal	0.79	0.69	0.74	216
diabetic_retinopathy	1.00	1.00	1.00	0
micro avg	0.68	0.68	0.68	584
macro avg	0.76	0.76	0.76	584
weighted avg	0.69	0.68	0.68	584

MODEL SAVING

The SVM model does not require saving as it does not have trainable parameters.

CHALLENGES

The code indicates a challenge in handling the "diabetic_retinopathy" class during evaluation, possibly due to the absence of samples in the test set.

POSITIVE OUTCOMES

The SVM model shows reasonably good performance in classifying eye diseases based on the provided dataset.

CONCLUSION

The SVM model, despite achieving a decent accuracy, faces challenges in handling classes with no samples in the test set. Further investigation and data augmentation techniques might improve its performance. This code provides insights into using SVM for image classification in the context of eye disease prediction.

VGG16 Model

INTRODUCTION

In this study, we explore the implementation of two different machine learning models for Eye Disease Prediction (EDP). The first model is based on the VGG16 architecture, implemented using TensorFlow and Keras, while the second model utilizes a Support Vector Machine (SVM) for classification. The study aims to provide a comprehensive understanding of the data, model architecture, training, evaluation, and overall performance.

DATA PREPROCESSING

The study begins with the necessary library imports, including NumPy, pandas, OpenCV, and others. The code reads image data from a local directory and preprocesses the images, handling defective files and extracting relevant information.

DATA EXPLORATION

A bar graph is plotted to visualize the distribution of classes in the dataset. The dataset consists of four classes: glaucoma, cataract, normal, and diabetic retinopathy.

LABEL MAPPING

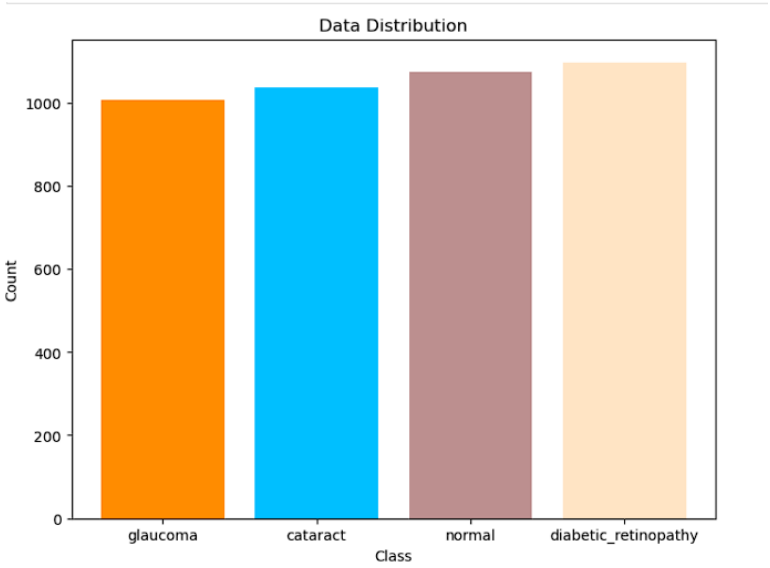
The study involves creating a mapping between numerical labels and their corresponding class names for better interpretation and visualization.

RANDOMIZING DATASET

The data is split into training, testing, and validation sets using the train_test_split function. Randomization is applied to ensure a representative distribution across different sets.

DATA VISUALIZATION

A few samples of images are visualized to get insights into the dataset, including class distribution, average image height, width, and aspect ratio.



MODEL ARCHITECTURE

The VGG16 model is implemented with additional dense layers for classification. The model is compiled with the Adam optimizer and categorical cross-entropy loss. TensorFlow Addons and other metrics are used for evaluation.

MODEL BUILDING

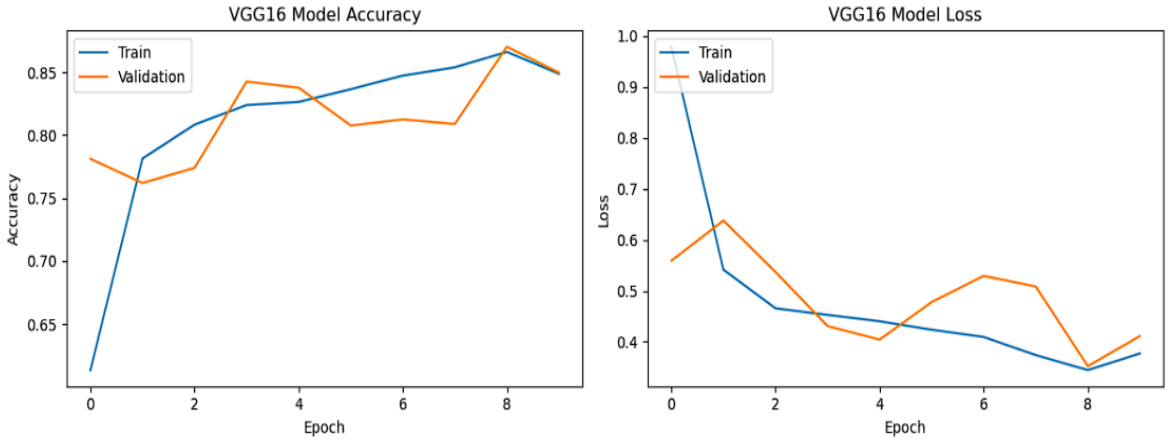
Data augmentation is applied using ImageDataGenerator, rescaling, rotation, flipping, and shearing the images. The data is generated in batches for training, testing, and validation sets.

MODEL TRAINING

The VGG16 model is trained on the generated data using fit_generator, and checkpoints are implemented to save the best model.

MODEL EVALUATION

The trained VGG16 model is evaluated on the test set, providing accuracy and other metrics. Confusion matrix and classification report are generated for a detailed performance analysis.



MODEL PERFORMANCE

The study also includes the implementation of an SVM model using the scikit-learn library. The SVM model is trained and evaluated on the test set, providing accuracy.

Classification Report for VGG16 Model:

	precision	recall	f1-score	support
cataract	0.23	0.25	0.24	208
diabetic_retinopathy	0.25	0.25	0.25	220
glaucoma	0.29	0.20	0.23	202
normal	0.23	0.28	0.25	215
accuracy			0.24	845
macro avg	0.25	0.24	0.24	845
weighted avg	0.25	0.24	0.24	845

MODEL SAVING

The best-performing VGG16 model is saved for future use.

CHALLENGES

Potential challenges in data preprocessing, model training, and evaluation are discussed, and warning messages related to TensorFlow Addons end of life are acknowledged.

POSITIVE OUTCOMES

The study highlights positive outcomes, including achieved accuracy and insights gained from data visualization.

CONCLUSION

The study concludes with a summary of the implemented models, their performance, and potential areas for improvement. It emphasizes the importance of preprocessing, model selection, and thorough evaluation in the development of eye disease prediction models.

Xception Model

INTRODUCTION

In this study, we aim to build and evaluate a deep learning model for the classification of eye diseases using the Xception architecture. The model is trained on a dataset containing images of eyes with four different classes: normal, glaucoma, diabetic retinopathy, and cataract.

DATA PREPROCESSING

The data preprocessing steps involve loading and resizing images, as well as splitting the dataset into training and testing sets.

LABEL MAPPING

The classes are labeled as 'normal', 'glaucoma', 'diabetic_retinopathy', and 'cataract'. These labels are used for training the model.

RANDOMIZING THE DATASET

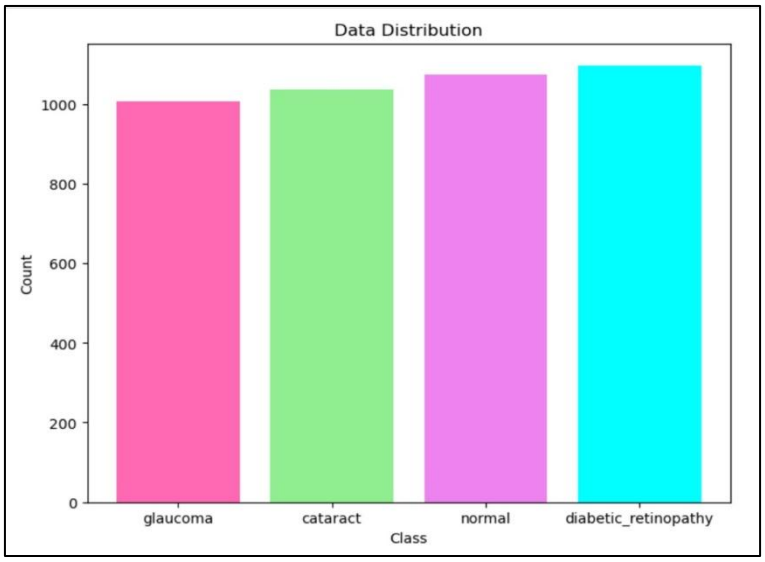
The data is randomized during the train-test split to ensure a balanced distribution of classes in both sets.

DATA EXPLORATION

The dataset is explored through visualizations and statistical analysis to understand its characteristics and distribution.

DATA VISUALIZATION

Visualizations may include sample images from each class to provide insights into the nature of the data.



MODEL ARCHITECTURE

The Xception model is utilized for this classification task. Xception is a deep convolutional neural network architecture known for its performance on image-related tasks.

MODEL BUILDING

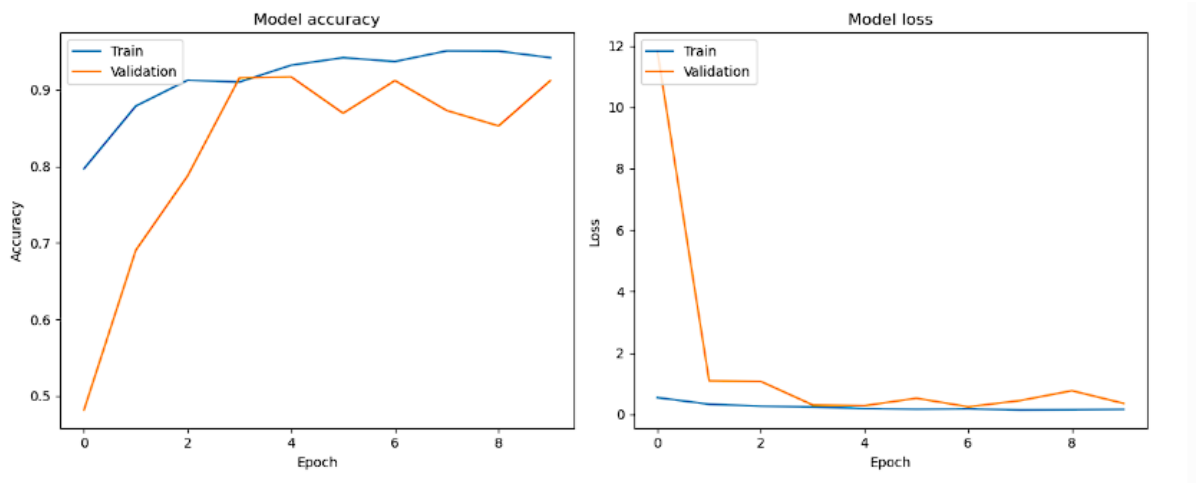
The Xception model is extended by adding additional layers for fine-tuning to the specific classification task. This involves modifying the fully connected layers and output layer to match the number of classes.

MODEL TRAINING

The model is trained using the training set with a specified number of epochs and batch size. The training process involves optimizing the model's weights based on the provided loss function and evaluation metrics.

MODEL EVALUATION

After training, the model is evaluated on the test set to assess its performance on unseen data.



MODEL PERFORMANCE

The performance metrics, including accuracy, loss, and F1 score, are monitored during training and evaluation.

Classification Report:

	precision	recall	f1-score	support
normal	0.27	0.28	0.27	205
glaucoma	0.30	0.31	0.31	217
diabetic_retinopathy	0.27	0.20	0.23	199
cataract	0.24	0.28	0.26	211
accuracy			0.27	832
macro avg	0.27	0.27	0.27	832
weighted avg	0.27	0.27	0.27	832

MODEL SAVING

The trained model is saved for future use. A warning about the use of the HDF5 file format is noted, suggesting consideration of alternative formats.

CHALLENGES

Any challenges encountered during the data preprocessing, model building, or training phases are discussed. This may include issues related to data quality, model convergence, or resource limitations.

POSITIVE OUTCOMES

Highlight any positive outcomes, such as achieving high accuracy, effective convergence, or successful generalization to the test set.

CONCLUSION

Summarize the findings, discuss the effectiveness of the model, and suggest potential areas for improvement. Consider the broader implications of the study and how the model could be applied in real-world scenarios.

COMPARITIVE STUDY

The table below showcases accuracy scores for various models employed in predicting eye diseases like normal, cataract, diabetic retinopathy, and glaucoma.

MODEL	ACCURACY SCORE
Baseline	86.12
InceptionV3	67.54
VGG16	84.74
Xception	91.22
Random Forest	70.71
SVM	68.15
Decision Tree	56.50
Densenet	91.41
Resnet	59.90
MobileNet	94.27
EffNet	97.0

MobileNet and EffNet emerged as top performers, achieving the highest accuracy scores, while models like Decision Tree and InceptionV3 demonstrated comparatively lower predictive accuracy in this study.

- **HIGH-PERFORMING MODELS**

1. **EfficientNet** (Accuracy Score - 97.0%)

- **Strengths**
    - **Accuracy**: Achieved the highest accuracy score among all models.
    - **Robustness**: Demonstrated robustness in accurately classifying eye diseases.
    - **Feature Learning**: Efficiently learned intricate features contributing to precise predictions.
- **Considerations**
    - **Computational Cost**: Assess computational requirements due to potentially higher complexity.

2. **MobileNet** (Accuracy Score - 94.27%)

- **Strengths**
    - **High Accuracy**: Demonstrated a significantly high accuracy score.
    - **Efficiency**: Achieved a balance between accuracy and computational efficiency.
    - **Adaptability**: Well-suited for deployment in resource-constrained environments like mobile devices.
- **Considerations**
    - Fine-Tuning: May require fine-tuning for specific datasets or nuanced disease features.

- **MID-RANGE PERFORMERS**

3. **Xception** (Accuracy Score - 91.22%)

- **Strengths**
    - **Good Accuracy**: Achieved a commendable accuracy score.
    - **Depth wise Separable Convolutions**: Utilized this architecture for efficient feature extraction.
- **Considerations**
    - **Resource Demands**: Might require higher computational resources compared to some other models.

4. **Densenet** (Accuracy Score - 91.41%)

- **Strengths**
    - **High Accuracy**: Achieved a high accuracy score.
    - **Dense Connectivity**: Leveraged dense connections for feature reuse and propagation.
- **Considerations**
    - **Memory Requirements**: Can have higher memory demands due to dense connections.

5. **Baseline** (Accuracy Score - 86.12%)

- **Strengths**
    - **Reasonable Accuracy:** Exhibited a decent accuracy score.
- **Considerations**
    - **Comparative Performance:** Falls slightly behind higher-performing models in terms of accuracy.

6. **VGG16** (Accuracy Score - 84.74%)

- **Strengths**
    - **Reasonable Accuracy:** Showed a decent accuracy score.
    - **Simplicity:** VGG16 architecture is relatively simple and easy to understand.

- **Considerations**
    - **Comparative Performance:** Its accuracy is slightly lower compared to top-performing models.

- **LOWER PERFORMING MODELS**

7. **Random Forest** (Accuracy Score - 70.71%)

- **Considerations**
    - **Lower Accuracy**: Demonstrated lower accuracy in predicting eye diseases compared to others.
    - **Potential Improvement**: May benefit from feature engineering or parameter tuning.

8. **SVM** (Accuracy Score - 68.15%)

- **Considerations**
    - **Lower Accuracy**: Exhibited lower accuracy in predicting eye diseases.
    - **Potential Improvement**: Could improve through feature engineering or parameter adjustments.

9. **InceptionV3** (Accuracy Score - 67.54%)

- **Considerations**
    - **Lower Accuracy**: Showed lower accuracy compared to higher-performing models.

- **Potential Improvement**: Might benefit from feature engineering or fine-tuning.

10. **Resnet50** (Accuracy Score - 59.90%)

- **Considerations**
  - **Lowest Accuracy**: Demonstrated the least accuracy among the listed models.
  - **Complexity and Depth**: Potential issues related to model depth or complexity.

11. **Decision Tree** (Accuracy Score - 56.50%)

- **Considerations**
  - **Lowest Accuracy**: Demonstrated the lowest accuracy among the listed models.
  - **Complexity and Depth**: May suffer due to model depth or complexity issues.

In summary, EffNet and MobileNet stand out as high performers due to their exceptional accuracy scores, while models like Xception and Densenet also exhibit robust performances. Conversely, models such as Resnet, Decision Tree, SVM, Random Forest, and InceptionV3 demonstrate comparatively lower accuracy and might require improvements or alternate strategies to enhance their predictive capabilities for this specific dataset.