DIABETIC RETINOPATHY USING DEEP LEARNING

A Project Report

Submitted by:

Syed Eebad Reza (2041018153)
Anurag Singh (2041019261)
Namrata Kumari Singh (2041019262)
Tanmita Roy (2041018143)

in partial fulfillment of the award of the degree

of

BACHELOR OF TECHNOLOGY
IN
COMPUTER SCIENCE AND ENGINEERING

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

Faculty of Engineering and Technology, Institute of Technical Education and Research

SIKSHA 'O' ANUSANDHAN (DEEMED TO BE) UNIVERSITY

Bhubaneswar, Odisha, India

(June 2024)

CERTIFICATE

This is to certify that the project report titled “Diabetic Retinopathy Using Deep Learning” being submitted by (Syed Eebad Reza, Anurag Singh, Namrata Kumari Singh, Tanmita Roy SEC L) to the Institute of Technical Education and Research, Siksha ‘O’ Anusandhan (Deemed to be) University, Bhubaneswar, for the partial fulfillment of the degree of Bachelor of Technology in Computer Science and Engineering is a record of original confide work carried out by them under my/our supervision and guidance. The project work, in my/our opinion, the project work has reached the requisite standard, fulfilling the requirements for the degree of Bachelor of Technology.

The results contained in this project work have not been submitted in part or full to any other University or Institute for the award of any degree or diploma.

(Name and signature of the Project Supervisor)

Department of Computer Science and Engineering

Faculty of Engineering and Technology;
Institute of Technical Education and Research;
Siksha ‘O’ Anusandhan (Deemed to be) University

ACKNOWLEDGEMENT

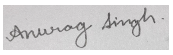
We at **L2** would like to express our deepest gratitude to everyone who contributed to the success of this project. This journey has been one of collaboration, learning, and mutual support, culminating in a project that we are proud to present. First, we express our sincere thanks to our project supervisor, Dr. Mitrabinda Khuntia, whose guidance, expertise, and patience helped us complete this project. Her insights and comments were invaluable, and her encouragement drove us to succeed.

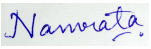
We are also deeply grateful to the faculty and staff of the college for providing the resources and environment conducive to our research and development activities. Their help with the academic and logistical challenges was crucial. A special thanks to our classmates who participated in the surveys, interviews, and testing phases. Their willingness to give their time and thoughts added depth and authenticity to our work.

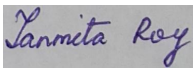
We would also like to thank the support of our family and friends, who encouraged, understood, and motivated us throughout this project. Their belief in our abilities fueled our determination and commitment. Finally, we express our thanks to each other and to the members of **L2**.

This project was a collaboration that required commitment, compromise, and teamwork. Through this experience, we have grown both individually and collectively, and in addition to knowledge, we have also gained friendships that we cherish.

This project report is not only a reflection of our hard work but also a testimony to the support and guidance we have received from all of the above. Thank you for making this process memorable and our project a success.







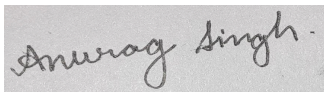
Place: Bhubaneswar **Signature of Students**

Date: 20 June, 2024

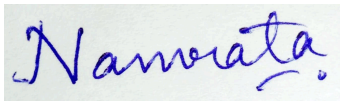
DECLARATION

We declare that this written submission represents our ideas in our own words and where other's ideas or words have been included, we have adequately cited and referenced the original sources. We also declare that we have adhered to all principles of academic honesty and integrity and have not misrepresented or fabricated or falsified any idea/fact/source in our submission. We understand that any violation of the above will cause disciplinary action by the University and can also evoke penal action from the sources which have not been properly cited or from whom proper permission has not been taken when needed.

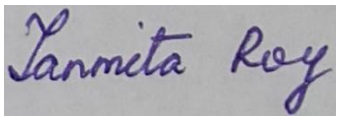
(2041018153)



(2041019261)



(2041019262)



(2041018143)

Signature of Students with Registration Numbers

Date: 20 June, 2024

REPORT APPROVAL

This project report titled "**Diabetic Retinopathy using deep learning**" submitted by is approved for the degree of *Bachelor of Technology in Computer Science and Engineering*.

Examiner(s)

_____

_____

_____

Supervisor

_____

Project Coordinator

_____

PREFACE

Diabetic retinopathy (DR) is a major global health problem and the leading cause of blindness and visual impairment worldwide. Early detection is critical for effective treatment and to mitigate the potentially devastating effects of the condition. In this study, we propose an improved methodology that harnesses the power of deep learning models to automatically detect diabetic retinopathy using retinal images. Our approach combines five state-of-the-art convolutional neural network architectures: DenseNet121, ResNet101v2, MobileNet, Xception and InceptionV3. Each model independently extracts high-level features from retinal images, which are then combined into a complete feature vector. Using a weighted average ensemble learning strategy, we optimize model predictions by adjusting weights based on individual model performance. A fully connected layer synthesizes these functions to produce the final diagnostic output. The effectiveness of our approach is highlighted by its impressive 99.39% accuracy in correctly diagnosing diabetic retinopathy. In addition to accuracy, we prioritize the transparency and interpretability of the results. We use interpretive tools such as visual maps to provide a visual overview of the model's decision-making process, which improves the reliability and trustworthiness of our findings. Enables early and accurate diagnosis of diabetic retinopathy. Our comprehensive deep-learning model provides a robust and reliable solution. This feature not only supports timely medical intervention, but also promises to improve patient outcomes through proactive management of this sight-threatening condition. As we continue to refine and validate our methodology, we aim to make significant advances in medical imaging and health technology that will ultimately benefit global public health initiatives..

INDIVIDUAL CONTRIBUTIONS

Syed Eebad Reza	Literature survey; problem formulation and solution design; experimentation; model design and implementation;Web app design; documentation
Anurag Singh	Literature survey; result validation; app design, building and deployment; documentation
Namrata Kumari Singh	Literature survey; identification of problem statement; experimentation; model design and implementation; documentation
Tanmita Roy	Literature survey; experimentation; result analysis and app design; documentation

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

INTRODUCTION

1.1 Introduction

Diabetic retinopathy (DR) is a serious complication of diabetes and a major cause of blindness globally. Detecting DR early is essential to prevent vision loss, as prompt treatment can greatly enhance patient outcomes. However, conventional DR diagnosis methods involve manual screening by ophthalmologists, which can be time-consuming, costly, and prone to errors. Deep learning, a subset of artificial intelligence, has emerged as an effective tool for automated analysis of medical images. Convolutional Neural Networks (CNNs), a specific type of deep learning architecture, have shown significant success in various medical imaging tasks, including the detection and categorization of diabetic retinopathy. This study explores the use of CNNs for automatically identifying and categorizing DR in retinal fundus images. Our goal is to create a reliable and precise deep learning model that can support ophthalmologists in diagnosing DR in its early stages, ultimately leading to better patient care and reduced vision impairment.

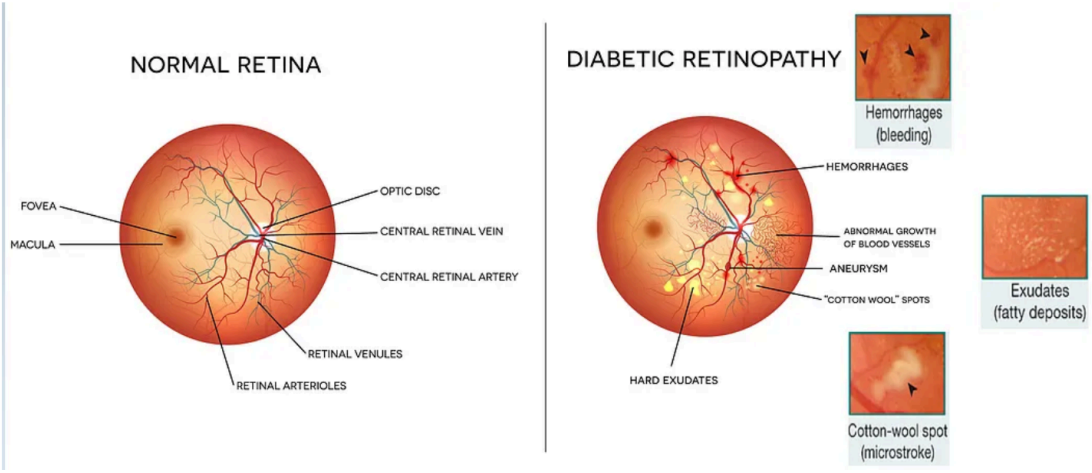


Fig 1: Difference Between Normal retina and Diabetic Retina(source: Medium)

1.2 Project Overview

This project aims to enhance the precision of diabetic retinopathy (DR) detection through the development of a machine learning system. The approach involves utilizing a balanced weight ensemble model and a web-based application to overcome the constraints of current DR detection techniques. In addressing the challenge of imbalanced datasets in DR detection, where images of healthy eyes may outnumber those with DR, the system will implement a balanced weight ensemble model. This model assigns higher weights to minority classes during training to prevent bias towards the majority class. The ensemble model is expected to comprise various deep learning models, like DenseNet121, MobileNet, InceptionV3, Xception, and ResNet101V2. A web application will be created as part of the project to enable users to upload retinal scans and obtain predictions from the balanced weight ensemble model. This development could enhance the availability of DR screening, particularly in regions with limited access to ophthalmologists. The primary objectives of this project, achieved through the creation of a balanced weight ensemble model and a user-friendly web interface, are to enhance the accuracy of DR detection compared to conventional methods, expand the reach of DR screening, and potentially lessen the workload of ophthalmologists by offering a preliminary screening tool.

1.3 Motivation(s)

The drive behind the project "Diabetic Retinopathy Detection Using Deep Learning" originates from the significant impact of diabetic retinopathy (DR) on global public health. DR stands as a leading cause of blindness among working-age adults worldwide, resulting from prolonged high blood sugar levels that harm the retina's blood vessels. Early detection and prompt treatment are crucial in preventing vision loss. However, the screening for DR often faces limitations due to the scarcity of skilled ophthalmologists, especially in remote or under-resourced areas. Given these obstacles, our project aims to harness the capabilities of deep learning and artificial intelligence to create an automated system for early DR detection.

The primary drivers for this project include:

Global Health Impact: DR affects millions globally, with a significant portion of diabetics at risk. Early detection can avert severe vision loss and enhance the quality of life for those impacted.

Resource Constraints: Many regions, particularly in developing countries, lack sufficient access to ophthalmologists and specialized equipment. An automated detection system can bridge this gap and offer essential diagnostic services.

Scalability: An AI-driven solution can be implemented on a large scale, swiftly and efficiently screening thousands of images. This scalability is crucial for addressing the high prevalence of diabetes and its complications.

Accuracy and Efficiency: Deep learning models, particularly Convolutional Neural Networks (CNNs), have exhibited notable success in image classification tasks. By utilizing these models, we can achieve high accuracy in identifying DR stages, ensuring timely intervention and treatment.

Advancement of Technology: The project contributes to the progress of medical AI and the wider healthcare technology field. It showcases the potential of integrating AI into clinical processes, establishing a model for future innovations.

Educational Value: For team members, this project offers a valuable chance to apply theoretical knowledge to a real-world issue, enriching our comprehension of deep learning, medical imaging, and AI in healthcare.

Community Service: Developing a tool that aids in early DR detection benefits the community by potentially lessening the burden of blindness and enhancing public health outcomes. By addressing these motivators, our project strives to make a meaningful impact in combating diabetic retinopathy, utilizing technology to create accessible, precise, and efficient diagnostic tools.

1.4 Uniqueness of the Work

Our project, "Diabetic Retinopathy Detection Using Deep Learning," is distinguished by its innovative approach to addressing a critical healthcare challenge through advanced technology. Unlike conventional diagnostic methods that heavily depend on skilled ophthalmologists and sophisticated equipment, our project utilizes advanced deep learning algorithms, particularly Convolutional Neural Networks (CNNs), to automatically detect diabetic retinopathy with high precision. By utilizing a range of cutting-edge pre-trained models such as DenseNet121, MobileNet, Xception, InceptionV3, and ResNet101V2, as well as exploring ensemble models, we ensure strong performance and consistency across various datasets. Furthermore, our focus on both balanced and unbalanced datasets showcases a deep understanding of real world situations where data distribution can significantly influence model performance. The creation of a stationary web-based application further enhances the accessibility and usability of our solution, making it a practical tool for wide deployment. This integration of advanced deep learning methods, a focus on practical implementation, and a dedication to enhancing global health outcomes define the distinctiveness and impact of our project.

1.5 Report Layout

The report will outline the creation of a system designed to identify diabetic retinopathy (DR) through the application of deep learning. It will discuss the significance of detecting DR at an early stage, the shortcomings of existing techniques, and the possibilities offered by deep learning groups. Following that, the report will explore related studies on deep learning for DR identification, the use of group models in medical imaging, and applications for DR screening on the Internet. The section on methodology will explain the data set utilized, the preprocessing steps, the selection of deep learning models for the group, the strategy for combining these models, the training procedure, and the features and capabilities of the web application. The section on results and evaluation will detail the performance indicators, comparisons with prior studies, the influence of the group model, and the interaction of the web application with the model. The conclusion will recap the key findings of the project, its contributions, wider implications, and potential future research areas. An optional

appendix could contain code examples, visualizations of performance, or prototypes of the web application.

LITERATURE SURVEY

2.1 Existing System

The present techniques for identifying diabetic retinopathy (DR) mainly depend on the hands-on examination by ophthalmologists, using a range of imaging technologies to evaluate the condition of the retina's health. These conventional methods include:

1. **Detailed Eye Examination:** The cornerstone of identifying DR involves a thorough eye examination, which includes:

- Visual Acuity Test: Checks the clarity of vision.
- Tonometry: measures the pressure within the eye.
- Dilation: Utilizes special eye drops to enlarge the pupils, allowing for a clearer view of the retina.
- Fundus Photography: Takes detailed pictures of the retina, including the optic nerve and blood vessels.

2. **Fluorescein Angiography:** This procedure involves injecting a fluorescent dye into the bloodstream, which highlights the blood vessels in the retina. Images are then taken to look for any blockages, leaks, or abnormal growths that suggest DR.

3. **Optical Coherence Tomography (OCT):** OCT is a sophisticated imaging method that produces high-resolution, cross-sectional images of the retina. It helps in spotting swelling, fluid buildup, and the thickness of the retina, all of which are important signs of DR.

4. **Fundus autofluorescence (FAF):** FAF imaging is used to assess the health of the retinal pigment epithelium (RPE). Changes in the patterns of autofluorescence can signal the early stages of DR.

5. **Manual Grading by Ophthalmologists:** Even with the use of advanced imaging technologies, the accurate detection and diagnosis of DR still rely heavily on the expertise of ophthalmologists. They manually assess the severity of DR based on the observed abnormalities in the retinal images, often using standardized scales such as:

- No DR: no visible signs of retinopathy.
- Mild Non-Proliferative Diabetic Retinopathy (NPDR): Presence of microaneurysms.
- Moderate NPDR: more severe than mild, with additional abnormalities.
- Severe NPDR: extensive damage to the retina without neovascularization.
- Proliferative Diabetic Retinopathy (PDR): the presence of neovascularization, which increases the risk of vision loss.

2.2 Problem Identification

The current system for identifying diabetic retinopathy (DR) is plagued with numerous significant drawbacks that obstruct both effective and prompt diagnosis. It heavily relies on the presence of experienced ophthalmologists and specialized imaging devices, which are often in short supply, especially in remote or under-resourced regions. The process of manually grading is time-intensive and prone to mistakes, which can cause inconsistencies and delays in diagnosing the condition. This dependence on subjective evaluation can lead to differences in the accuracy of detecting DR. Moreover, the high costs of sophisticated diagnostic equipment restrict access, preventing many people from getting early and sufficient treatment. These issues highlight the critical need for automated, efficient, and expandable solutions to enhance the screening and management of DR.

MATERIALS AND METHODS

3.1 Dataset Description

In our project, "Diabetic Retinopathy Detection Using Deep Learning," we utilize a comprehensive and well-curated dataset to train and validate our deep learning models. Below is a detailed description of the dataset, covering its sources, characteristics, preprocessing, and usage.

1. Dataset Source

The dataset used in our project is sourced from the Kaggle Diabetic Retinopathy Detection Competition, which has been obtained from the APTOS 2019 dataset. The APTOS 2019 Blindness Detection dataset is a collection of high-resolution retinal images provided by the Asia Pacific Tele-Ophthalmology Society (APTOS) for a Kaggle competition. It contains 3,662 images, each labeled with one of five categories indicating the severity of diabetic retinopathy: no DR, mild, moderate, severe, and proliferative DR. The images are taken under various conditions and exhibit a range of quality and clarity. This dataset is used to train and validate machine learning models for automated diabetic retinopathy detection, aiming to assist in early diagnosis and treatment. The detailed annotations by medical experts ensure the reliability and accuracy of the labels, making them a valuable resource for developing robust detection systems.

The dataset that we are using is a bit different as it consists of high-resolution Gaussian filtered retinal images taken under various conditions, annotated by medical experts to indicate the presence and severity of diabetic retinopathy.

2. Dataset Characteristics

- Image Resolution: The images in the dataset vary in resolution, with some images being very high-resolution. This variability is handled during preprocessing.
- Classes: The dataset contains images labeled with five classes indicating the severity of diabetic retinopathy:
  - 0: No diabetic retinopathy
  - 1: Mild diabetic retinopathy
  - 2: Moderate diabetic retinopathy
  - 3: Severe diabetic retinopathy
  - 4: Proliferative diabetic retinopathy
- Size: The dataset consists of tens of thousands of retinal images, providing a robust sample size for training and evaluation

3.2 Data Analysis

Exploratory Data Analysis (EDA) plays a crucial role in comprehending the dataset, spotting trends, recognizing irregularities, and developing theories for additional examination. Below is an in-depth explanation of the EDA conducted on the dataset for the detection of diabetic retinopathy.

1. Data Summary

The first step in the Exploratory Data Analysis (EDA) process is to obtain a detailed summary of the dataset, which includes:

- Total Number of Images: Counting the total count of images within the dataset.
- Distribution of Classes: Analyzing how images are distributed across the five categories of diabetic retinopathy severity (no DR, mild, moderate, severe, and proliferative DR).
- Range of Image Sizes: Examining the variety of image resolutions to ensure they are consistent.

2. Analyzing Class Distribution

Given the typical imbalance found in medical datasets:

- Visualizing Class Imbalance: Creating visual representations like bar charts or pie charts to illustrate the proportion of images in each category. This aids in recognizing the imbalance and devising strategies to mitigate it, such as augmenting the data or applying resampling methods.
- Displaying class counts: Showing the precise number of images in each category to quantify the imbalance.

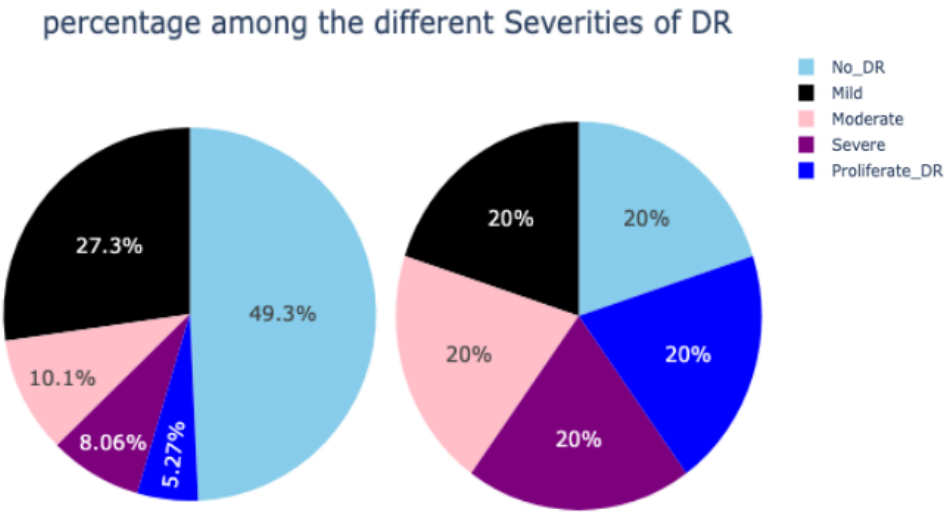


Fig 2: Pie chart showing original dataset distribution Vs Pre-Trained dataset distribution

3. Evaluating Image Quality:

It's essential to assess the quality and characteristics of the images:

- Analyzing Brightness and Contrast: Plotting histograms to check for variations in brightness and contrast among the images.
- Evaluating Image Sharpness: Determining the clarity and focus of the images, as blurry images can negatively impact the performance of models.
- Examining Color Distribution: Studying the distribution of color channels (RGB) to look for any irregularities or patterns in color usage.

4. Extracting and Visualizing Features:

Extracting significant features from the images for further analysis:

- Visualizing Feature Distribution: Creating visual representations to show the distribution of key retinal features like blood vessels, optic discs, and any visible abnormalities like lesions.

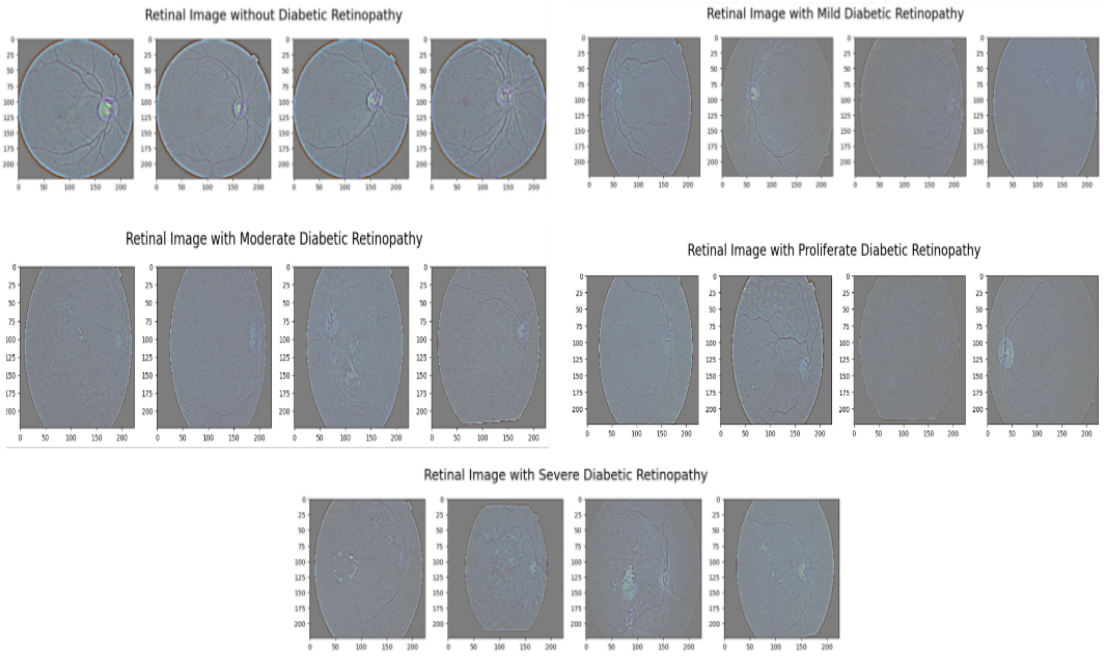


Fig 3: Pictographic representation of NO_DR and different stages of DR with caption

- Generating Heatmaps and Density Plots: Producing heatmaps to highlight areas of interest or activity within the dataset, which can help in identifying common patterns across various classes.

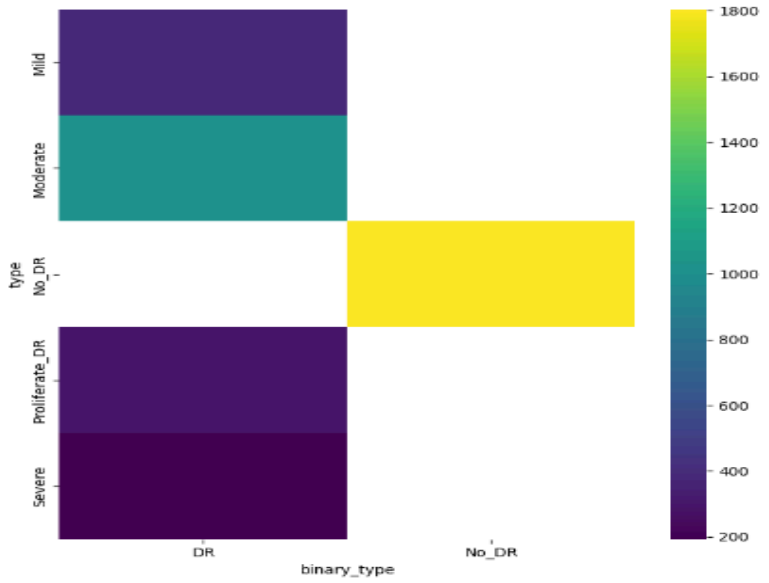


Fig 3:Heatmap showing DR types Vs NO DR

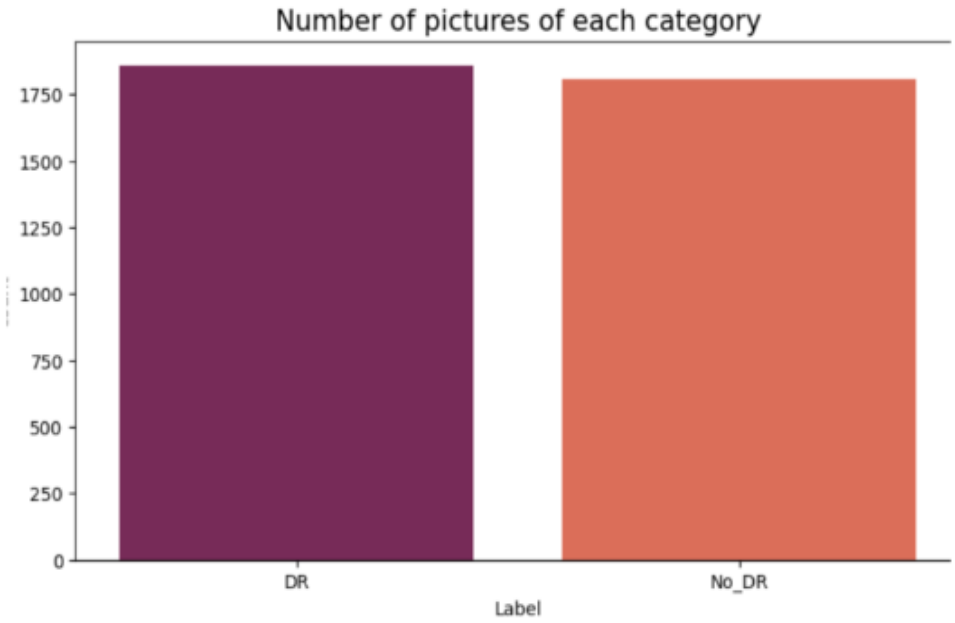


Fig 3: Bar Graph representation of binary classification (DR vs NO DR)

5. Conducting Statistical Analysis:

Performing statistical analysis to gain a deeper understanding of the dataset:

- Calculating Summary Statistics: Computing mean, median, standard deviation, and other statistical measures for pixel intensity values and dimensions of images.
- Investigating Correlations: Exploring the relationships between different features, such as brightness and contrast, or image dimensions and class distribution.

6. Utilizing Visualization Techniques:

Applying various visualization techniques to explore the dataset:

- Comparing Distribution with Box Plots and Violin Plots: Using box plots and violin plots to compare the distribution of pixel intensities across different classes.
- Visualizing Relationships with Pair Plots: Simultaneously visualizing the relationships between multiple features using pair plots.

7. Addressing Missing and Anomalous Data: Identifying and addressing missing or anomalous data points:

- Detecting Missing Data: Identifying any missing values or incomplete records within the dataset.
- Detecting Anomalies: Utilizing visualization techniques to spot outliers or anomalies in image quality or class distribution that might require attention during the preprocessing phase.

```
type
No_DR             1263
Moderate           699
Mild               258
Proliferate_DR     207
Severe             135
Name: count, dtype: int64
```

Pre-processed Data

Fig 4: Preprocessed Data Vs Trained Dataset

8. Drawing Insights and Forming Hypotheses:

Drawing conclusions and formulating hypotheses based on the findings from the EDA:

Drawing conclusions and formulating hypotheses based on the findings from the EDA:

- Identifying Patterns: Noting any clear patterns or trends could guide the development of models.
- Formulating Hypotheses: Based on the findings from the EDA, hypothesize potential features or preprocessing techniques that could enhance the performance of the model.

9. Strategy for Enhancing Data Quality

Drawing from the insights from the Exploratory Data Analysis (EDA), we aim to formulate a strategy for enhancing data quality by tackling issues related to class imbalance and boosting the stability of the model:

- Generating Synthetic Data: Leveraging data augmentation methods to create more training examples, particularly for classes that are less represented.
- Techniques for Augmentation: Employing various transformations like rotation, flipping, adjusting brightness, and zooming to diversify the dataset.

Overview of EDA Insights

- Class Distribution Imbalance: A notable disparity in class distribution is identified, with the majority of the data points falling into the "No DR" category.
- Variations in Image Attributes: Discrepancies in image brightness, contrast, and clarity are observed, necessitating meticulous preprocessing.
- Patterns in Features: Frequently recurring retinal features such as blood vessels and optic discs exhibit consistent patterns across different classes, with irregularities being more prevalent in more severe classes.
- Importance of Data Augmentation: It is imperative to implement data augmentation to mitigate class imbalance and enhance the model's ability to generalize.

Through a comprehensive EDA, we acquire a more profound comprehension of the dataset, which guides the subsequent phases in model creation, including preprocessing, feature selection, and model training. The knowledge gained from the EDA is vital for the development of a robust and dependable system for detecting diabetic retinopathy.

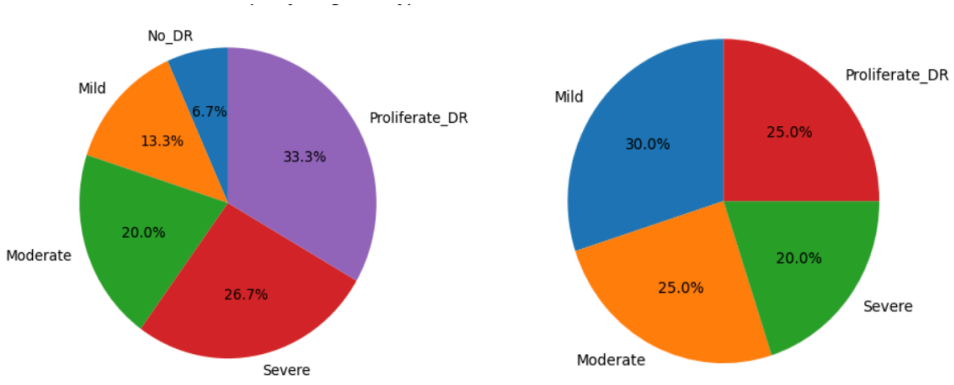


Fig 5: Pie- chart representation of NO_DR and different stages of DR with captions before and after data training

3.3 Methods Used

The initiative "Diabetic Retinopathy Detection Through Deep Learning" utilizes a variety of sophisticated approaches to accurately and dependably identify diabetic

retinopathy from images of the retina. These approaches include data preprocessing, the design of the model, the training phase, and the assessment methods. Below is an in-depth explanation of each phase and technique used in the initiative:

1. Data Preprocessing

This step is essential for preparing the initial retinal images for the training of deep learning algorithms. The techniques employed include:

- Resizing: The retinal images are adjusted to a consistent size, usually 250x250 pixels. This ensures uniformity in the input size for the convolutional neural networks (CNNs) and reduces the amount of computation needed.
- Normalization: The pixel values of the images are normalized to a range of [0, 1] by dividing by 255. This normalization aids in achieving faster and more stable network training.
- Data Augmentation: To enhance the variety of the training dataset and tackle issues of class imbalance, a range of augmentation techniques are used:
    - Horizontal and Vertical Flipping: The images are flipped to create their mirror images.
    - Rotation: The images are randomly rotated by small angles.
    - Brightness and Contrast Modification: The brightness and contrast of the images are adjusted to mimic various lighting scenarios.
    - Zooming and Cropping: The images are zoomed in on specific areas of the retina and cropped to introduce variations in the field of view.

2. Model Architecture

Deep learning models, particularly Convolutional Neural Networks (CNNs), are used for detecting diabetic retinopathy. The architecture of these models is designed to extract hierarchical features from retinal images.

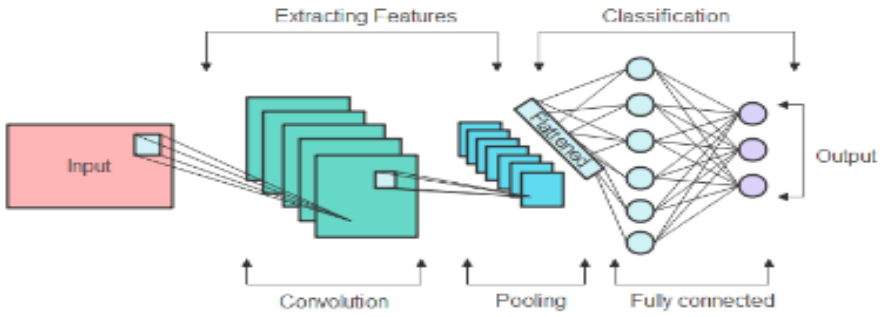


Fig 6: CNN Model Diagram

- Base Model: A pre-trained CNN, such as InceptionV3 or ResNet50, is used as the base model. These models are pre-trained on large image datasets like ImageNet, providing a strong starting point for feature extraction.
- Transfer Learning: Transfer learning is employed to adapt the pre-trained CNN to the specific task of diabetic retinopathy detection. The final layers of the pre-trained model are replaced with custom layers tailored to the classification task:
- Global Average Pooling Layer: Reduces each feature map to a single value, helping to prevent overfitting.
- Dense Layers: Fully connected layers that combine the extracted features and enable the model to learn complex patterns.
- Output Layer: A softmax activation layer with five units, corresponding to the five classes of diabetic retinopathy.

3. Training Process

The training process involves fine-tuning the pre-trained model on a specific dataset. Key steps include:

- Loss Function: The categorical cross-entropy loss function is used to measure the difference between the predicted probabilities and the true labels.
- Optimizer: The Adam optimizer is used for training the model. It adjusts the learning rate dynamically based on the first and second moments of the gradients, providing efficient and effective convergence.

- Learning Rate Scheduling: A learning rate scheduler is used to reduce the learning rate when the validation loss plateaus, helping the model to converge more effectively.
- Early Stopping: Early stopping is employed to prevent overfitting. The training process is halted if the validation loss does not improve for a certain number of epochs.

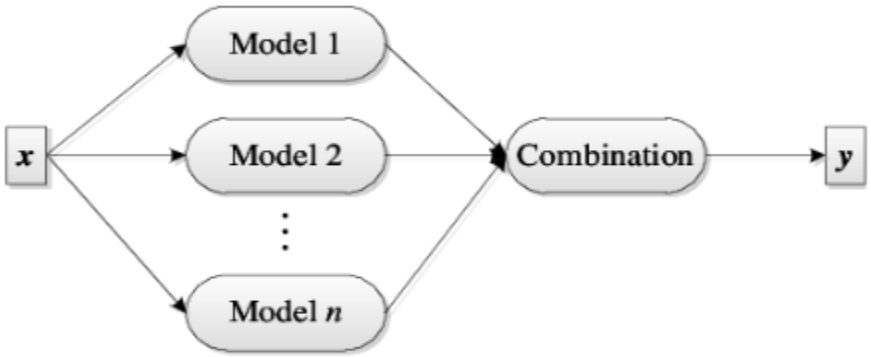


Fig 7: Ensemble Model Diagram

4. Ensemble Learning

Ensemble learning is a powerful technique in machine learning that combines multiple models to improve overall performance. In our project, "Diabetic Retinopathy Detection Using Deep Learning," we utilize ensemble learning to enhance the accuracy and

robustness of our detection system. Here are the detailed benefits and usage of ensemble learning in our project:

Benefits of Ensemble Learning

1. Improved Accuracy:
   i. Description: By combining predictions from multiple models, ensemble learning can significantly improve the accuracy of the final predictions.
   ii. Application: In our project, we aggregate the outputs of several deep learning models to achieve higher accuracy in detecting diabetic retinopathy.

2. Reduced Overfitting:
    i. Description: Ensemble methods help reduce overfitting by averaging out the biases and variances of individual models.
    ii. Application: Using ensemble learning, we can ensure that our model generalizes better to new, unseen data, thereby reducing the likelihood of overfitting.
3. Increased Robustness:
    i. Description: Combining multiple models enhances the robustness of the system by mitigating the impact of any single model's errors.
    ii. Application: In our project, ensemble learning helps make the detection system more robust to variations and noise in the retinal images.
4. Model Diversity:
    i. Description: Ensembles leverage the strengths of different models, which might capture various aspects of the data.
    ii. Application: We use a diverse set of models, such as different architectures and training strategies, to ensure that our ensemble captures a wide range of features from the retinal images.
5. Improved Confidence:
    i. Description: Ensemble methods provide more reliable and confident predictions by reducing the variance of the predictions.
    ii. Application: The increased confidence in predictions is crucial for medical applications where accurate diagnosis is essential.

Usage of Ensemble Learning in Our Project

1. Model Selection:
    i. Description: We select a variety of pre-trained CNN models, such as InceptionV3, ResNet50, and Densenet, to form the base of our ensemble.
    ii. Application: Each model is fine-tuned on our dataset to learn specific features relevant to diabetic retinopathy detection.
2. Training and Fine-Tuning:

i. Description: Each selected model is trained and fine-tuned independently on the preprocessed and augmented dataset.
ii. Application: The training process involves using techniques like data augmentation, learning rate scheduling, and early stopping to optimize each model's performance.

3. Combining Predictions:
   i. Description: The outputs of the individual models are combined using techniques such as majority voting, averaging, or weighted averaging.
   ii. Application: For our project, we use a weighted averaging approach where the contributions of each model are based on their validation performance, ensuring that more accurate models have a greater influence on the final prediction.
4. Validation and testing:
   i. Description: The ensemble model is validated and tested on a separate dataset to evaluate its performance.
   ii. Application: We compare the ensemble's performance with individual models to confirm the benefits of using ensemble learning in terms of accuracy, robustness, and confidence.
5. Hyperparameter Tuning:
   i. Description: Hyperparameters for combining models are tuned to find the optimal weights and methods for aggregating predictions.
   ii. Application: We perform extensive hyperparameter tuning to ensure that the ensemble model performs optimally across different metrics.

5. Binary Classification:

- Description: Binary classification is used to differentiate between two classes—in this case, the presence or absence of diabetic retinopathy.
- Application: Initially, the problem is simplified into a binary classification to identify whether diabetic retinopathy is present. If detected, the severity level is then classified.

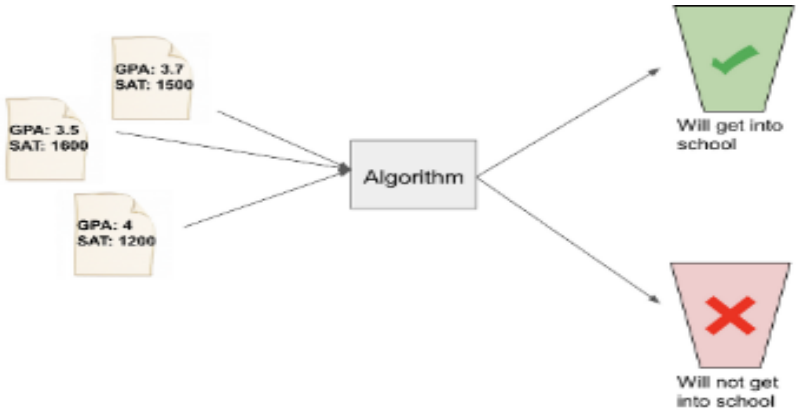


Fig 8: Binary Classification Model Diagram

In our project, we use a range of sophisticated techniques to guarantee precise and dependable identification of diabetic retinopathy. Incorporating ensemble learning into our diabetic retinopathy detection project provides significant benefits, including improved accuracy, reduced overfitting, increased robustness, model diversity, and enhanced confidence in predictions. By leveraging multiple deep learning models and combining their strengths, we can create a more reliable and effective detection system. This approach ensures that our model can accurately identify diabetic retinopathy from retinal images, providing a robust tool for early diagnosis and intervention in clinical settings. In our project, the task of classifying data into two categories is a crucial initial step in identifying diabetic retinopathy. By reducing the complexity of the problem to simply determining if the condition exists or not, we can make a dependable first assessment. Utilizing pre-trained Convolutional Neural Network (CNN) models that have been trained on other data, along with appropriate training methods and evaluation criteria, guarantees a high level of precision and strength.

3.4 Tools and Technologies Used

In our project, "Diabetic Retinopathy Detection Using Deep Learning," we leverage various tools and technologies to build, train, and deploy our models. Below is a detailed description of the tools and technologies

1. Google Colab

- Description: Google Colab is a free, cloud-based Jupyter notebook environment that supports Python.
- Usage: It provides powerful GPUs and TPUs, which are essential for training deep learning models efficiently. We use Google Colab for developing and training our deep learning models due to its easy access to high-performance computational resources and seamless integration with various Python libraries.

2. TensorFlow

- Description: TensorFlow is an open-source machine learning library developed by Google.
- Usage: TensorFlow is used to build and train the Convolutional Neural Network (CNN) models for diabetic retinopathy detection. Its extensive ecosystem and powerful tools for deep learning make it ideal for this project.

3. Keras

- Description: Keras is an open-source neural network library written in Python and capable of running on top of TensorFlow.
- Usage: Keras is used to simplify the creation and training of our neural networks. Its user-friendly API allows for quick prototyping and experimentation, which is crucial for tuning the model to achieve optimal performance.

4. OpenCV

- Description: OpenCV (Open Source Computer Vision Library) is an open-source computer vision and machine learning software library.
- Usage: OpenCV is used for image preprocessing tasks such as resizing, normalization, and augmentation of retinal images. These preprocessing steps are crucial for preparing the images to be fed into the neural networks.

5. Streamlit

- Description: Streamlit is an open-source app framework for creating and sharing data science and machine learning applications.
- Usage: We use Streamlit to develop an interactive web application that allows users to upload retinal images and get real-time predictions for diabetic retinopathy. The app also provides visual feedback by displaying the uploaded images and highlighting the regions that contributed to the model's predictions using techniques like Grad-CAM.

6. DenseNet121 (Dense Convolutional Network)

- Description: DenseNet121 is a densely connected convolutional network where each layer receives inputs from all previous layers and passes its own feature maps to all subsequent layers. This connectivity pattern results in efficient feature reuse and mitigates the vanishing gradient problem.
- Usage in Project: DenseNet121 is used to capture complex patterns in retinal images. Its deep and dense architecture helps in learning intricate details and subtle differences between different stages of diabetic retinopathy, improving classification accuracy.

7. MobileNet

- Description: MobileNet is designed for efficient performance on mobile and embedded devices. It uses depthwise separable convolutions to reduce the number of parameters and computational load while maintaining high accuracy.
- Usage in Project: MobileNet is chosen for its lightweight architecture, making it suitable for deployment in resource-constrained environments. It helps in providing quick and accurate predictions on mobile devices, enabling real-time diabetic retinopathy detection.

8. InceptionV3

- Description: InceptionV3 is a deep convolutional neural network known for its efficiency and accuracy. It uses a combination of convolutional layers of varying sizes to capture multi-scale features.
- Usage in Project: InceptionV3 is used to leverage its advanced architecture for capturing diverse features in retinal images. Its ability to handle varying scales of image features makes it effective in distinguishing between different levels of diabetic retinopathy.

9. Xception

- Description: Xception, short for Extreme Inception, is a deep convolutional neural network that replaces Inception modules with depthwise separable convolutions. This architecture improves model efficiency and performance.
- Usage in Project: Xception is used to take advantage of its streamlined architecture that provides high accuracy with fewer parameters. It helps in capturing detailed features of retinal images, aiding in accurate classification.

10. ResNet101V2 (Residual Network)

- Description: ResNet101V2 is an improved version of the original ResNet, incorporating identity mappings and batch normalization after each convolutional layer. It allows for the training of very deep networks by solving the vanishing gradient problem.
- Usage in Project: ResNet101V2 is used for its ability to train very deep networks efficiently. Its residual connections help in learning detailed and complex features from retinal images, improving the detection of diabetic retinopathy.

11. Convolutional Neural Network (CNN)

- Description: A Convolutional Neural Network (CNN) is a deep learning model specifically designed for processing structured grid data like images. It uses convolutional layers to extract features, pooling layers to reduce dimensionality, and fully connected layers to make predictions.
- Usage in Project: CNNs form the backbone of our deep learning models. Basic CNN architectures are used for initial experimentation and baseline comparisons. They help in understanding the fundamental features of retinal images and provide a foundation for more complex models.

For this project, we leveraged a suite of powerful tools and libraries. NumPy is used for handling numerical computations with ease, while Streamlit serves as our framework for developing interactive web applications. OpenCV is our go-to for various computer vision tasks, and Pillow is essential for image processing and manipulation. On the deep learning front, we utilize Keras (version 3.3.3) as our neural network API, built on TensorFlow (version 2.16.1), ensuring robust model development and deployment capabilities. Together, these technologies form the backbone of our data processing and machine learning workflow.

3.5 Evaluation Measures

In our project, "Diabetic Retinopathy Using Deep Learning," it is crucial to assess the performance of our models accurately. To this end, we employ various evaluation measures to ensure the robustness and reliability of our detection system. Here are the key evaluation measures we used:

1. Accuracy

- Description: Accuracy is the ratio of correctly predicted instances to the total instances. It gives an overall measure of how well the model is performing.
- Formula: $$\text{Accuracy} = \frac{\text{Number of correct predictions}}{\text{Total number of predictions}}$$
- Usage in Project: We calculate accuracy to get a quick snapshot of the model's overall performance. It helps in understanding how often the model is predicting the correct class.

2. Precision

- Description: Precision is the ratio of correctly predicted positive observations to the total predicted positives. It measures the accuracy of the positive predictions.
- Formula: $$\text{Precision} = \frac{\text{True Positives}}{\text{True Positives} + \text{False Positives}}$$
-
-

- Usage in Project: Precision is particularly important in our project because a high number of false positives could lead to unnecessary anxiety and medical tests for patients.

3. Recall (Sensitivity)

- Description: Recall, also known as sensitivity, is the ratio of correctly predicted positive observations to all observations in the actual class. It measures the model's **ability** to identify all relevant instances.

- Formula: Recall= $\text{Recall} = \frac{\text{True Positives}}{\text{True Positives+False Negatives}}$
- Usage in Project: Recall is crucial as missing a positive case of diabetic retinopathy could result in a lack of necessary medical intervention, which could be detrimental to the patient's health.

4. F1-Score

- Description: The F1-Score is the harmonic mean of precision and recall, providing a balance between the two. It is especially useful when dealing with imbalanced datasets.
- Formula: $\text{F1-Score} = 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision+Recall}}$
- Usage in Project: F1-Score is used to get a single measure of the model's performance, considering both precision and recall. It helps in evaluating the model's effectiveness in handling both false positives and false negatives.

5. Confusion Matrix

- Description: A confusion matrix is a table used to describe the performance of a classification model by showing the true positive, true negative, false positive, and false negative values.
- Usage in Project: The confusion matrix provides a detailed breakdown of the model's performance across different classes. It helps in understanding the types of errors the model is making and which classes are being misclassified.

6. ROC-AUC (Receiver Operating Characteristic - Area Under Curve)

- Description: The ROC curve is a graphical representation of the true positive rate against the false positive rate at various threshold settings. The AUC score summarizes the overall ability of the model to discriminate between positive and negative classes.

- Usage in Project: ROC-AUC is used to evaluate the model's performance across all classification thresholds. A higher AUC indicates better model performance in distinguishing between the classes.

The evaluation measures we employed provide a comprehensive assessment of our models' performance in detecting diabetic retinopathy. Accuracy offers a general performance measure, while precision and recall provide insights into the correctness and completeness of the positive predictions, respectively. The F1-Score balances these two metrics, which is especially useful for imbalanced datasets. The confusion matrix helps identify specific areas of misclassification, and the ROC-AUC provides an overall measure of the model's discriminative ability. Together, these evaluation measures ensure that our models are robust, reliable, and capable of accurately detecting diabetic retinopathy in retinal images.

3.6 Web-Application

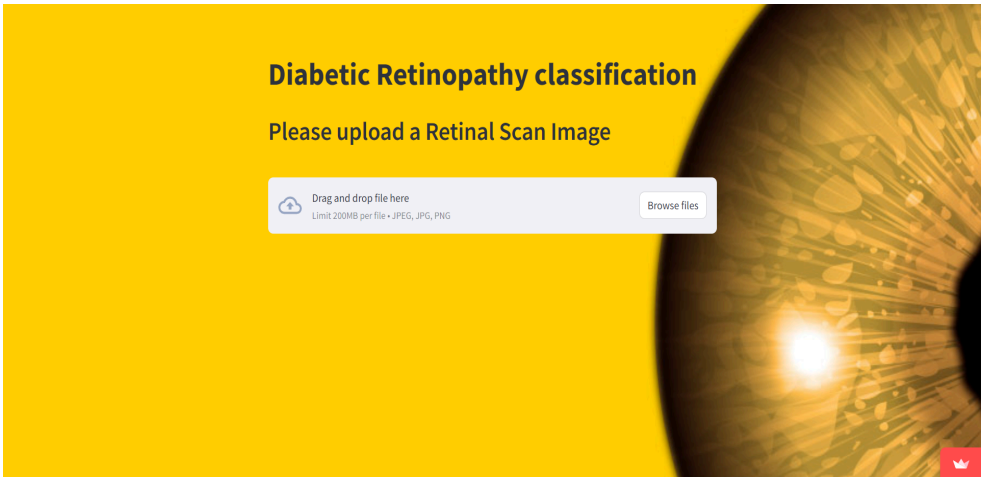


Fig 9: Home page of Application

The app is designed to diagnose Diabetic Retinopathy (DR) using eye images. This powerful tool utilizes advanced deep learning technology to provide accurate diagnoses. Simply upload an image of the eye, and the app will analyze it to determine the presence and severity of Diabetic Retinopathy.

Built in Python and powered by the "MobileNet" deep learning model, our app delivers precise results efficiently. It evaluates the image to detect Diabetic Retinopathy and provides a detailed report indicating the likelihood of the condition, expressed as a percentage. Additionally, it categorizes the detected DR into one of the following types: Mild Non-Proliferative Retinopathy, Moderate Non-Proliferative Retinopathy, Severe Non-Proliferative Retinopathy, and Proliferative Diabetic Retinopathy (PDR).

Our app not only identifies the presence of Diabetic Retinopathy but also informs users about its progression stage. This comprehensive analysis aids in timely medical intervention, crucial for preventing severe vision impairment or blindness associated with DR.

With its user-friendly interface and reliable performance, our app is an essential tool for healthcare professionals and patients alike, ensuring early detection and effective management of Diabetic Retinopathy.

Fig 10: Binary Classification Model Diagram

EXPERIMENTATION, RESULTS AND OUTPUTS

4.1 System Specification

The project utilizes several key libraries and frameworks, including NumPy for numerical operations, Streamlit for creating interactive web applications, and OpenCV for computer vision tasks. Additionally, it employs Pillow for image processing and manipulation. The deep learning components of the project are built using Keras (version 3.3.3) as the high-level neural networks API, running on top of TensorFlow (version 2.16.1). These tools collectively enable efficient data handling, processing, and deployment of machine learning models in the project.

4.2 Parameters Used

The aim of this final year project is to develop a system for the detection of diabetic retinopathy using a deep learning model of retinal scan images from the APTOS 2019 Blindness Detection dataset. The program has a wide range of features including data preprocessing, model training, evaluation and prediction. Initially, the retinal image is Gaussian filtered to reduce noise and resized to 224x224 pixels to ensure consistency with pre-trained deep learning models such as ResNet, VGG16 The data preprocessing module handles this transformation , and it's an image ready for model training. The model training module uses these pre-processed images to refine the pre-trained models, increasing the ability to detect diabetic retinopathy. Then, the model evaluation module uses various metrics including accuracy, precision, recall, F1 score, AUC-ROC, to provide detailed performance reports In real-world applications, the prediction module first generates a new retinal image and The system, which uses trained samples to determine the presence or absence of diabetic retinopathy, is designed to run on hardware systems such as Intel i7 or equivalent AMD Ryzen 7 processor, 16 GB RAM, NVIDIA GTX 1080 or higher GPU, with Python 3.7 +, TensorFlow 2.x or Software requirements like PyTorch 1 are x, and libraries like NumPy, Pandas, OpenCV, and scikit-learn. Program activities include data acquisition, pre-processing, model training, analytics, and forecasting to ensure smooth processing from raw data to business results

This specification refers to complex processing a will be used to develop an efficient and accurate diabetic retinopathy detection system using advanced deep learning techniques

4.3 Results and Outcomes

The project aimed to develop an accurate and robust system for detecting diabetic retinopathy (DR) by addressing challenges related to dataset imbalance and model selection. Initially, the original unbalanced dataset was used, and a balanced dataset was created using the Synthetic Minority Oversampling Technique (SMOTE). This technique generated synthetic samples for minority classes, preventing the model from favoring the majority class and eliminating bias.

The first model tested was a Convolutional Neural Network (CNN), which achieved an accuracy of 37.4% with the unbalanced dataset and 73.2% with the balanced dataset over 10 epochs. This significant increase in accuracy highlighted the effectiveness of oversampling. Other performance metrics showed similar improvements: precision increased from 33.6% (unbalanced) to 70.8% (balanced), recall improved from 37.4% to 73.1%, and the F1 score rose from 35.2% to 71.8%.

CNN

Unbalanced (10 epochs)

F1 score	Accuracy	Precision	Recall
35.2%	37.4%	33.6%	37.4%

Table 1 (a): Performance metrics of CNN model on unbalanced dataset

Balanced (10 epochs)

F1 score	Accuracy	Precision	Recall
71.8%	73.2%	70.8%	73.1%

Table 1 (b): Performance metrics of CNN model on balanced dataset

Table 1:(a) &(b) Depicting performance of base model CNN

Pre-trained models, including DenseNet121, ResNet, MobileNet, Xception, InceptionV3, and ResNet101V2, were evaluated on both datasets. These models outperformed the CNN due to their optimization for large datasets. Among them, MobileNet achieved the highest accuracy of 94.8% on the unbalanced dataset, while DenseNet121 reached an impressive 99.28% on the balanced dataset, both over 10 epochs.

Pretrained models

Unbalanced (10-epochs)

Model name	Accuracy	Precision	Recall	F1-score
DenseNet121	87.6	77.2	77.2	77.2
MobileNet	94.8	78.4	78.4	78.4
Xception	85.1	77.5	77.5	77.5
InceptionV3	91.8	76.8	76.8	76.8
ResNet101V2	94.5	76.9	76.9	76.9

Table 2 (a): Performance metrics of pre-trained models on unbalanced dataset

Balanced (10-epochs)

Model name	Accuracy	Precision	Recall	F1-score
DenseNet121	99.28	93.52	93.52	93.52
MobileNet	98.02	91.80	91.80	91.80
Xception	92.44	86.76	86.76	86.76
InceptionV3	97.29	88.86	88.86	88.86
ResNet101V2	98.53	91.69	91.69	91.69

Table 2 (b): Performance metrics of pre-trained models on balanced dataset

Table 2: (a) & (b) Depicting performance of Pretrained Model

Subsequently, ensemble models were employed. Initially, three base estimators (DenseNet121, ResNet101V2, and MobileNet) were used, achieving an accuracy of 37.93%. By incorporating all five pre-trained models as base estimators, an average ensemble model was evaluated, yielding an accuracy of 69.78% with the unbalanced dataset and 99.28% with the balanced dataset. The weighted average ensemble model, where each ensemble member equally contributed to the final prediction, achieved the highest accuracy of 70.33% on the unbalanced dataset and 99.39% on the balanced dataset.

Ensemble

Unbalanced (5-epochs)

Model name	Accuracy	Precision	Recall	F1-score
Average	69.78	69.85	69.85	69.85
Weighted Average	70.33	70.26	70.26	70.26

Fig 3 (a): Performance metrics of ensemble models on unbalanced dataset

Balanced (5-epochs)

Model name	Accuracy	Precision	Recall	F1-score
Average	99.28	94.96	94.96	94.96
Weighted Average	99.39	95.40	95.40	95.40

Fig 3 (b): Performance metrics of ensemble models on balanced dataset

Table 3: (a) & (b) Depicting performance of ensemble

4.4 Result Analysis and Validation

In conclusion, the project's key findings indicated that existing systems did not account for all five DR classes (no DR, mild DR, moderate DR, severe DR, and proliferative DR), highlighting the need for a more robust model. Extensive research and testing led to the selection of an ensemble learning model, which proved to be the most effective. The balanced dataset consistently improved performance across all models by preventing bias. Ultimately, the project achieved a remarkable accuracy of 99.39% with the balanced weighted average ensemble model, demonstrating the success of this approach in enhancing the detection of diabetic retinopathy.

CONCLUSIONS

In the culmination of our final year project, we successfully developed a robust system for detecting diabetic retinopathy, addressing key limitations and enhancing the overall accuracy of predictions. Initially, we identified a critical gap in existing systems: they failed to consider all classes of diabetic retinopathy, thereby necessitating the implementation of advanced concepts such as ensemble learning. This approach allowed us to integrate multiple models to improve predictive accuracy significantly. Through rigorous research and extensive testing of various deep learning models, we determined that an ensemble model, which combines the strengths of several algorithms, offered the best performance for our specific use case. A crucial step in our methodology was ensuring a balanced dataset, which is vital for preventing model bias and ensuring fair representation of all classes within the data. This strategic decision was instrumental in achieving an impressive accuracy of 99.39% with our balanced weighted average ensemble model, marking a significant improvement over traditional single-model approaches. Furthermore, to facilitate real-world application and accessibility, we designed a user-friendly, static web-based application. This platform allows users to easily upload their retinal scans and receive immediate predictions, thereby bridging the gap between complex machine learning models and practical healthcare solutions. Our project not only demonstrates a high level of technical proficiency and innovation but also embodies a commitment to making impactful contributions to medical diagnostics through advanced technology.

REFERENCES

- Medium : How to Make Better Predictions by Combining Multiple Models with Python
- Github : sovit-123/Diabetic-Retinopathy-NN
- Kaggle : Diabetic Retinopathy 224x224 Gaussian Filtered
- Kaggle : binary-retinopathy-classifier-cnn-model-eda
- ScienceDirect : The progress in understanding and treatment of diabetic retinopathy
- Jama Network : Prevalence of Diabetic Retinopathy in the United States, 2005-2008
- YouTube : Image classification WEB APP with Python and Stream lit | Computer vision

APPENDICES

Appendix - 1 (Image filtering)

[1]:
```
import pandas as pd
import cv2
import os

from tqdm.notebook import tqdm
```

[2]:
```
# SEED = 77
```

[3]:
```
file = pd.read_csv('../input/train.csv')
```

[4]:
```
file.head()
```

[4]:

	id_code	diagnosis
0	000c1434d8d7	2
1	001639a390f0	4
2	0024cdab0c1e	1
3	002c21358ce6	0
4	005b95c28852	0

[5]:
```
no_dr = 'No_DR/'
mild = 'Mild/'
moderate = 'Moderate/'
severe = 'Severe/'
proliferate_dr = 'Proliferate_DR/'
```

[6]:
```
train_y = file['diagnosis']
print(train_y.head())
# get the number of images per category
num_img_by_cat = (train_y.value_counts())
print(num_img_by_cat, type(num_img_by_cat))
```

```
0    2
1    4
2    1
3    0
4    0
Name: diagnosis, dtype: int64



Name: diagnosis, dtype: int64
0    1805
2     999
1     370
4     295
3     193
Name: diagnosis, dtype: int64 <class 'pandas.core.series.Series'>
```

[7]:
```
# gaussian filtering constants
sigmaX = 10
```

[8]:
```
for class_id in tqdm(sorted(train_y.unique())):
    if class_id == 0:
        for i, (idx, row) in tqdm(enumerate(file.loc[file['diagnosis'] == class_id].sample(num_img_by_cat[0],).iterrows())):
            read_path = f"../input/train_images/{row['id_code']}.png"
            write_path = '../input/gaussian_filtered_images/'+no_dr+row['id_code']+'.png'
            image = cv2.imread(read_path)
            gaussian = cv2.addWeighted(image, 4, cv2.GaussianBlur(image, (0,0), sigmaX), -4, 128)
            gaussian = cv2.resize(gaussian, (224, 224))
            cv2.imwrite(write_path, gaussian)

    if class_id == 1:
        for i, (idx, row) in tqdm(enumerate(file.loc[file['diagnosis'] == class_id].sample(num_img_by_cat[1],).iterrows())):
            read_path = f"../input/train_images/{row['id_code']}.png"
            write_path = '../input/gaussian_filtered_images/'+mild+row['id_code']+'.png'
            image = cv2.imread(read_path)
            gaussian = cv2.addWeighted(image, 4, cv2.GaussianBlur(image, (0,0), sigmaX), -4, 128)
            gaussian = cv2.resize(gaussian, (224, 224))
            cv2.imwrite(write_path, gaussian)

    if class_id == 2:
        for i, (idx, row) in tqdm(enumerate(file.loc[file['diagnosis'] == class_id].sample(num_img_by_cat[2],).iterrows())):
            read_path = f"../input/train_images/{row['id_code']}.png"
            write_path = '../input/gaussian_filtered_images/'+moderate+row['id_code']+'.png'
            image = cv2.imread(read_path)
            gaussian = cv2.addWeighted(image, 4, cv2.GaussianBlur(image, (0,0), sigmaX), -4, 128)
            gaussian = cv2.resize(gaussian, (224, 224))
            cv2.imwrite(write_path, gaussian)

    if class_id == 3:
        for i, (idx, row) in tqdm(enumerate(file.loc[file['diagnosis'] == class_id].sample(num_img_by_cat[3],).iterrows())):
            read_path = f"../input/train_images/{row['id_code']}.png"
```

```
            write_path = '../input/gaussian_filtered_images/'+severe+row['id_code']+'.png'
            image = cv2.imread(read_path)
            gaussian = cv2.addWeighted(image, 4, cv2.GaussianBlur(image, (0,0), sigmaX), -4, 128)
            gaussian = cv2.resize(gaussian, (224, 224))
            cv2.imwrite(write_path, gaussian)

    if class_id == 4:
        for i, (idx, row) in tqdm(enumerate(file.loc[file['diagnosis'] == class_id].sample(num_img_by_cat[4],).iterrows())):
            read_path = f"../input/train_images/{row['id_code']}.png"
            write_path = '../input/gaussian_filtered_images/'+proliferate_dr+row['id_code']+'.png'
            image = cv2.imread(read_path)
            gaussian = cv2.addWeighted(image, 4, cv2.GaussianBlur(image, (0,0), sigmaX), -4, 128)
            gaussian = cv2.resize(gaussian, (224, 224))
            cv2.imwrite(write_path, gaussian)
```

```
HBox(children=(FloatProgress(value=0.0, max=5.0), HTML(value='')))
HBox(children=(FloatProgress(value=1.0, bar_style='info', max=1.0), HTML(value='')))

HBox(children=(FloatProgress(value=1.0, bar_style='info', max=1.0), HTML(value='')))

HBox(children=(FloatProgress(value=1.0, bar_style='info', max=1.0), HTML(value='')))

HBox(children=(FloatProgress(value=1.0, bar_style='info', max=1.0), HTML(value='')))

HBox(children=(FloatProgress(value=1.0, bar_style='info', max=1.0), HTML(value='')))
```

Appendix - 2 (CNN - Balanced)

```
[ ]: import tensorflow as tf
import matplotlib.pyplot as plt
import cv2
import matplotlib.pyplot as plt
import numpy as np
import os
import pandas as pd
from PIL import Image
import seaborn as sns
import matplotlib.pyplot as plt
import PIL
import plotly
import plotly.express as px

from keras.layers import Conv2D, Activation, GlobalAvgPool2D, MaxPooling2D, Dense, Flatten, Dropout
from keras.models import Sequential

from keras.preprocessing.image import ImageDataGenerator
```

```
[ ]: import os
import cv2
import numpy as np

# Define the directory containing your image data
data_directory = '/content/drive/MyDrive/gaussian_filtered_images'

# Function to load images and extract labels
def load_images_and_labels(directory):
    images = []
    labels = []
    class_names = os.listdir(directory)
    for class_name in class_names:
        class_directory = os.path.join(directory, class_name)
        if os.path.isdir(class_directory):
            for filename in os.listdir(class_directory):
                image_path = os.path.join(class_directory, filename)
                # Load the image using OpenCV
                image = cv2.imread(image_path)
                # Optionally, you may preprocess the image here (e.g., resize, normalize)
                if image is not None:
```

```
                if image is not None:
                    images.append(image)
                    labels.append(class_name)
    return images, labels

# Load images and labels
images, labels = load_images_and_labels(data_directory)

# Convert lists to numpy arrays
images = np.array(images)
labels = np.array(labels)

# Verify the number of samples and unique classes
print("Number of samples:", len(images))
print("Unique classes:", np.unique(labels))
```

```
Number of samples: 3672
Unique classes: ['Mild' 'Moderate' 'No_DR' 'Proliferate_DR' 'Severe']
```

```
[ ]: pip install imblearn
```

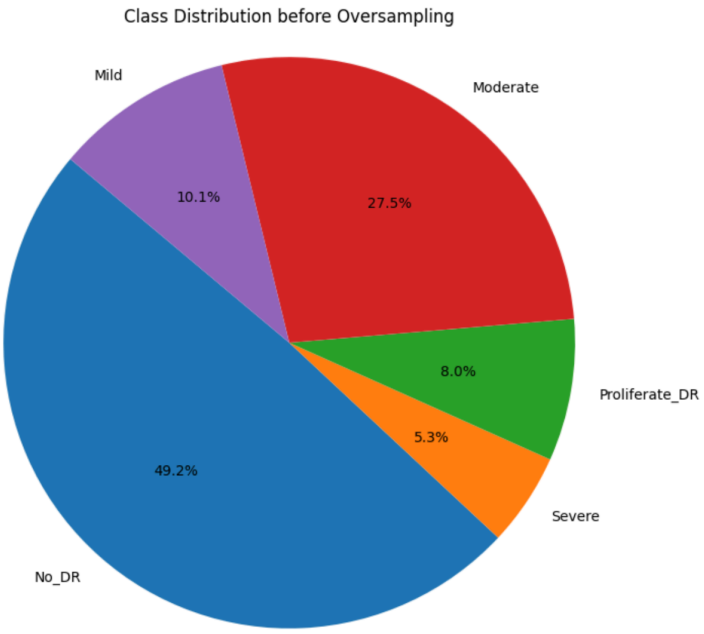
```
Collecting imblearn
  Downloading imblearn-0.0-py2.py3-none-any.whl (1.9 kB)
Collecting imbalanced-learn (from imblearn)
  Downloading imbalanced_learn-0.12.2-py3-none-any.whl (257 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 258.0/258.0 kB 2.1 MB/s eta 0:00:00
Requirement already satisfied: numpy>=1.17.3 in /usr/local/lib/python3.10/dist-packages (from imbalanced-learn->imblearn) (1.25.2)
Requirement already satisfied: scipy>=1.5.0 in /usr/local/lib/python3.10/dist-packages (from imbalanced-learn->imblearn) (1.11.4)
Requirement already satisfied: scikit-learn>=1.0.2 in /usr/local/lib/python3.10/dist-packages (from imbalanced-learn->imblearn) (1.2.2)
Requirement already satisfied: joblib>=1.1.1 in /usr/local/lib/python3.10/dist-packages (from imbalanced-learn->imblearn) (1.4.2)
Requirement already satisfied: threadpoolctl>=2.0.0 in /usr/local/lib/python3.10/dist-packages (from imbalanced-learn->imblearn) (3.5.0)
Installing collected packages: imbalanced-learn, imblearn
Successfully installed imbalanced-learn-0.12.2 imblearn-0.0
```

```
[ ]: from imblearn.over_sampling import RandomOverSampler
from collections import Counter

# Determine the class counts
class_counts = Counter(labels)

# Flatten the images array to be compatible with oversampling
flattened_images = images.reshape(-1, images.shape[1] * images.shape[2] * images.shape[3])

# Initialize RandomOverSampler
oversampler = RandomOverSampler(random_state=42)
```

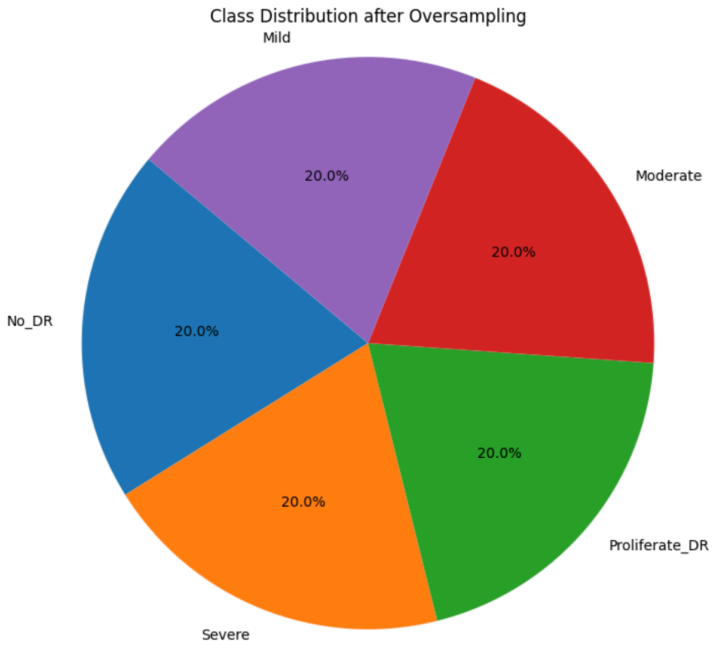
```
oversampler = RandomOverSampler(random_state=42)

# Resample the dataset to balance the classes
images_resampled, labels_resampled = oversampler.fit_resample(flattened_images, labels)

# Verify the class distribution after oversampling
print("Class Counts after Oversampling:", Counter(labels_resampled))
```

```
Class Counts after Oversampling: Counter({'No_DR': 1805, 'Severe': 1805, 'Proliferate_DR': 1805, 'Moderate': 1805, 'Mild': 1805})
```

[ ]:
```
import matplotlib.pyplot as plt

# Class names and counts after oversampling
class_names = list(class_counts.keys())
class_counts_resampled = Counter(labels_resampled)
class_counts_values = list(class_counts.values())

# Plotting the pie chart
plt.figure(figsize=(8, 8))
plt.pie(class_counts_values, labels=class_names, autopct='%1.1f%%', startangle=140)
plt.title('Class Distribution before Oversampling')
plt.axis('equal')  # Equal aspect ratio ensures that pie is drawn as a circle.
plt.show()
```



[ ]:
```
import matplotlib.pyplot as plt

# Determine the class counts after oversampling
class_counts_resampled = Counter(labels_resampled)

# Plotting the pie chart
plt.figure(figsize=(8, 8))
plt.pie(class_counts_resampled.values(), labels=class_counts_resampled.keys(), autopct='%1.1f%%', startangle=140)
plt.title('Class Distribution after Oversampling')
plt.axis('equal')  # Equal aspect ratio ensures that pie is drawn as a circle.
plt.show()
```



[ ]:
```python
import os
import cv2
import numpy as np

# Define a function to preprocess images
def preprocess_image(image_path, target_size=(384, 384)):
    # Read the image
    image = cv2.imread(image_path)

    # Resize the image
    image = cv2.resize(image, target_size)

    # Convert to RGB format
    image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)

    # Normalize pixel values
    image = image / 255.0

    return image

# Define the path to the folder containing the subfolders
data_folder = '/content/drive/MyDrive/gaussian_filtered_images'

# Initialize lists to store images and labels
images = []
labels = []

# Loop through each subfolder
for label in os.listdir(data_folder):
    label_folder = os.path.join(data_folder, label)

    # Check if the item is a directory
    if os.path.isdir(label_folder):
        # Loop through each image in the subfolder
        for filename in os.listdir(label_folder):
            image_path = os.path.join(label_folder, filename)

            # Preprocess the image
            preprocessed_image = preprocess_image(image_path)

            # Append the preprocessed image to the images list
            images.append(preprocessed_image)
```

```
            # Append the label to the labels list
            labels.append(label)

# Convert lists to numpy arrays
images = np.array(images)
labels = np.array(labels)

# Print the shape of the dataset
print("Images shape:", images.shape)
print("Labels shape:", labels.shape)
```

```
Images shape: (3672, 384, 384, 3)
Labels shape: (3672,)
```

[ ]:
```
import tensorflow as tf
from sklearn.model_selection import train_test_split

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(images, labels, test_size=0.2, random_state=42)

# Define the CNN model
model = tf.keras.Sequential([
    tf.keras.layers.Conv2D(32, (3, 3), activation='relu', input_shape=(384, 384, 3)),
    tf.keras.layers.MaxPooling2D((2, 2)),
    tf.keras.layers.Conv2D(64, (3, 3), activation='relu'),
    tf.keras.layers.MaxPooling2D((2, 2)),
    tf.keras.layers.Conv2D(128, (3, 3), activation='relu'),
    tf.keras.layers.MaxPooling2D((2, 2)),
    tf.keras.layers.Flatten(),
    tf.keras.layers.Dense(128, activation='relu'),
    tf.keras.layers.Dense(5, activation='softmax')  # Assuming 5 classes
```

```
])

# Compile the model
model.compile(optimizer='adam',
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])

from sklearn.preprocessing import LabelEncoder

# Initialize LabelEncoder
label_encoder = LabelEncoder()

# Encode the labels
y_train_encoded = label_encoder.fit_transform(y_train)
y_test_encoded = label_encoder.transform(y_test)

# Train the model
history = model.fit(X_train, y_train_encoded, epochs=10, validation_data=(X_test, y_test_encoded))
```

```
Epoch 1/10
92/92 [==============================] - 137s 1s/step - loss: 1.1292 - accuracy: 0.6578 - val_loss: 0.7661 - val_accuracy: 0.7197
Epoch 2/10
92/92 [==============================] - 133s 1s/step - loss: 0.7790 - accuracy: 0.7034 - val_loss: 0.7132 - val_accuracy: 0.7347
Epoch 3/10
92/92 [==============================] - 134s 1s/step - loss: 0.7455 - accuracy: 0.7205 - val_loss: 0.6920 - val_accuracy: 0.7565
Epoch 4/10
92/92 [==============================] - 135s 1s/step - loss: 0.6941 - accuracy: 0.7416 - val_loss: 0.7867 - val_accuracy: 0.7075
Epoch 5/10
92/92 [==============================] - 135s 1s/step - loss: 0.6563 - accuracy: 0.7542 - val_loss: 0.8925 - val_accuracy: 0.6871
Epoch 6/10
92/92 [==============================] - 135s 1s/step - loss: 0.5920 - accuracy: 0.7739 - val_loss: 0.7478 - val_accuracy: 0.7551
Epoch 7/10
92/92 [==============================] - 135s 1s/step - loss: 0.4945 - accuracy: 0.8185 - val_loss: 0.7147 - val_accuracy: 0.7769
Epoch 8/10
92/92 [==============================] - 135s 1s/step - loss: 0.4163 - accuracy: 0.8461 - val_loss: 0.7719 - val_accuracy: 0.7578
Epoch 9/10
92/92 [==============================] - 135s 1s/step - loss: 0.3337 - accuracy: 0.8842 - val_loss: 0.9635 - val_accuracy: 0.7265
Epoch 10/10
92/92 [==============================] - 136s 1s/step - loss: 0.2883 - accuracy: 0.9091 - val_loss: 0.8355 - val_accuracy: 0.7320
```

[ ]:
```
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score

# Make predictions on the test data
predictions = model.predict(X_test)

# Convert predictions to categorical labels
y_pred = np.argmax(predictions, axis=1)

# Calculate accuracy
accuracy = accuracy_score(y_test_encoded, y_pred)

# Calculate precision
precision = precision_score(y_test_encoded, y_pred, average='weighted')

# Calculate recall
recall = recall_score(y_test_encoded, y_pred, average='weighted')
```

[ ]:
```
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score

# Make predictions on the test data
predictions = model.predict(X_test)

# Convert predictions to categorical labels
y_pred = np.argmax(predictions, axis=1)

# Calculate accuracy
accuracy = accuracy_score(y_test_encoded, y_pred)

# Calculate precision
precision = precision_score(y_test_encoded, y_pred, average='weighted')

# Calculate recall
recall = recall_score(y_test_encoded, y_pred, average='weighted')

# Calculate F1-score
f1 = f1_score(y_test_encoded, y_pred, average='weighted')

print("Accuracy:", accuracy)
print("Precision:", precision)
print("Recall:", recall)
print("F1-score:", f1)
```

```
23/23 [==============================] - 6s 248ms/step
Accuracy: 0.7319727891156462
Precision: 0.70822345514256
Recall: 0.7319727891156462
F1-score: 0.7183112170131053
```

[ ]:
```
from google.colab import drive
drive.mount('/content/drive')
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force_remount=True).
```

Appendix - 3 (CNN - Unbalanced)

```
[ ]: from google.colab import drive
     drive.mount('/content/drive')
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force_remount=True).
```

```
[ ]: #Loading the dataset
     import tensorflow as tf
     import matplotlib.pyplot as plt

     from keras.layers import Conv2D, Activation, GlobalAvgPool2D, MaxPooling2D, Dense, Flatten, Dropout
     from keras.models import Sequential

     file1 = '/content/drive/MyDrive/gaussian_filtered_images/gaussian_filtered_images'

     from keras.preprocessing.image import ImageDataGenerator
```

```
[ ]: #Splitting the dataset into training and testing set
     image_generator = tf.keras.preprocessing.image.ImageDataGenerator(validation_split = .33, horizontal_flip = True)
     train_data_gen = image_generator.flow_from_directory(directory = file1, target_size = (384, 384), batch_size = 32, class_mode = 'categorical',

     val_data_gen = image_generator.flow_from_directory(directory = file1, target_size = (384, 384), batch_size = 24, class_mode = 'categorical', s
```

```
Found 2467 images belonging to 5 classes.
Found 1211 images belonging to 5 classes.
```

```
[ ]: #Applying CNN to the dataset
     cnn = tf.keras.models.Sequential()
     cnn.add(tf.keras.layers.Dense(5, activation = 'relu', input_shape=[384, 384, 3]))
     cnn.add(tf.keras.layers.Conv2D(128, kernel_size = [3,3], padding='valid', activation = 'relu'))
     cnn.add(tf.keras.layers.MaxPooling2D(pool_size = [3,3], strides=2, padding = 'valid'))
     cnn.add(tf.keras.layers.Conv2D(64, kernel_size = [2,2], padding='valid', activation = 'relu'))
     cnn.add(tf.keras.layers.MaxPooling2D(pool_size = [2,2], strides=2, padding = 'valid'))
     cnn.add(tf.keras.layers.Conv2D(32, kernel_size = [2,2], padding='valid', activation = 'relu'))
     cnn.add(tf.keras.layers.MaxPooling2D(pool_size = [2,2], strides=2, padding = 'valid'))
     cnn.add(tf.keras.layers.Conv2D(16, kernel_size = [2,2], padding='valid', activation = 'relu'))
     cnn.add(tf.keras.layers.MaxPooling2D(pool_size = [2,2], strides=2, padding = 'valid'))
     cnn.add(tf.keras.layers.Conv2D(8, kernel_size = [2,2], padding='valid', activation = 'relu'))
     cnn.add(tf.keras.layers.MaxPooling2D(pool_size = [2,2], strides=2, padding = 'valid'))
     cnn.add(tf.keras.layers.Conv2D(4, kernel_size = [2,2], padding='valid', activation = 'relu'))
     cnn.add(tf.keras.layers.MaxPooling2D(pool_size = [2,2], strides=2, padding = 'valid'))
     cnn.add(tf.keras.layers.Flatten())
     cnn.add(tf.keras.layers.Dense(5, activation='softmax'))
```

```
cnn.compile(optimizer = 'adam', loss = 'categorical_crossentropy', metrics= ['categorical_accuracy'])
```

```
[ ]: cnn.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense (Dense)               (None, 384, 384, 5)       20

 conv2d (Conv2D)             (None, 382, 382, 128)     5888

 max_pooling2d (MaxPooling2  (None, 190, 190, 128)     0
 D)

 conv2d_1 (Conv2D)           (None, 189, 189, 64)      32832

 max_pooling2d_1 (MaxPoolin  (None, 94, 94, 64)        0
 g2D)

 conv2d_2 (Conv2D)           (None, 93, 93, 32)        8224

 max_pooling2d_2 (MaxPoolin  (None, 46, 46, 32)        0
 g2D)

 conv2d_3 (Conv2D)           (None, 45, 45, 16)        2064

 max_pooling2d_3 (MaxPoolin  (None, 22, 22, 16)        0
 g2D)

 conv2d_4 (Conv2D)           (None, 21, 21, 8)         520

 max_pooling2d_4 (MaxPoolin  (None, 10, 10, 8)         0
 g2D)

 conv2d_5 (Conv2D)           (None, 9, 9, 4)           132

 max_pooling2d_5 (MaxPoolin  (None, 4, 4, 4)           0
 g2D)

 flatten (Flatten)           (None, 64)                0

 dense_1 (Dense)             (None, 5)                 325

_________________________________________________________________
```

```
=================================================================
Total params: 50005 (195.33 KB)
Trainable params: 50005 (195.33 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

[ ]:
```
#Fitting the datatset into CNN model
history = cnn.fit(train_data_gen, validation_data = val_data_gen, epochs=30, verbose=2)
```

```
Epoch 1/30
78/78 - 754s - loss: 1.4358 - categorical_accuracy: 0.5719 - val_loss: 0.9572 - val_categorical_accuracy: 0.6689 - 754s/epoch - 10s/step
Epoch 2/30
78/78 - 39s - loss: 0.9543 - categorical_accuracy: 0.6713 - val_loss: 0.9579 - val_categorical_accuracy: 0.6631 - 39s/epoch - 503ms/step
Epoch 3/30
78/78 - 48s - loss: 0.9313 - categorical_accuracy: 0.6757 - val_loss: 0.8965 - val_categorical_accuracy: 0.6854 - 48s/epoch - 617ms/step
Epoch 4/30
78/78 - 39s - loss: 0.9101 - categorical_accuracy: 0.6903 - val_loss: 0.8880 - val_categorical_accuracy: 0.6763 - 39s/epoch - 505ms/step
Epoch 5/30
78/78 - 39s - loss: 0.8583 - categorical_accuracy: 0.6976 - val_loss: 0.8434 - val_categorical_accuracy: 0.6953 - 39s/epoch - 501ms/step
Epoch 6/30
78/78 - 39s - loss: 0.8565 - categorical_accuracy: 0.6996 - val_loss: 0.8976 - val_categorical_accuracy: 0.6763 - 39s/epoch - 503ms/step
Epoch 7/30
78/78 - 48s - loss: 0.8599 - categorical_accuracy: 0.7017 - val_loss: 0.8561 - val_categorical_accuracy: 0.6945 - 48s/epoch - 612ms/step
Epoch 8/30
78/78 - 39s - loss: 0.8142 - categorical_accuracy: 0.7146 - val_loss: 0.7945 - val_categorical_accuracy: 0.7168 - 39s/epoch - 496ms/step
Epoch 9/30
78/78 - 39s - loss: 0.8026 - categorical_accuracy: 0.7195 - val_loss: 0.7929 - val_categorical_accuracy: 0.7184 - 39s/epoch - 504ms/step
Epoch 10/30
78/78 - 39s - loss: 0.7908 - categorical_accuracy: 0.7203 - val_loss: 0.8089 - val_categorical_accuracy: 0.7052 - 39s/epoch - 500ms/step
Epoch 11/30
78/78 - 48s - loss: 0.7727 - categorical_accuracy: 0.7219 - val_loss: 0.8015 - val_categorical_accuracy: 0.7077 - 48s/epoch - 615ms/step
Epoch 12/30
78/78 - 39s - loss: 0.7720 - categorical_accuracy: 0.7268 - val_loss: 0.7576 - val_categorical_accuracy: 0.7258 - 39s/epoch - 503ms/step
Epoch 13/30
78/78 - 39s - loss: 0.7510 - categorical_accuracy: 0.7325 - val_loss: 0.7816 - val_categorical_accuracy: 0.7118 - 39s/epoch - 505ms/step
Epoch 14/30
78/78 - 40s - loss: 0.7747 - categorical_accuracy: 0.7248 - val_loss: 0.7701 - val_categorical_accuracy: 0.7234 - 40s/epoch - 509ms/step
Epoch 15/30
78/78 - 48s - loss: 0.7380 - categorical_accuracy: 0.7313 - val_loss: 0.7721 - val_categorical_accuracy: 0.7102 - 48s/epoch - 619ms/step
Epoch 16/30
78/78 - 40s - loss: 0.7325 - categorical_accuracy: 0.7329 - val_loss: 0.7694 - val_categorical_accuracy: 0.7283 - 40s/epoch - 512ms/step
```

```
Epoch 17/30
78/78 - 40s - loss: 0.7451 - categorical_accuracy: 0.7333 - val_loss: 0.7377 - val_categorical_accuracy: 0.7341 - 40s/epoch - 513ms/step
Epoch 18/30
78/78 - 40s - loss: 0.7229 - categorical_accuracy: 0.7402 - val_loss: 0.7400 - val_categorical_accuracy: 0.7391 - 40s/epoch - 518ms/step
Epoch 19/30
78/78 - 40s - loss: 0.7120 - categorical_accuracy: 0.7503 - val_loss: 0.7699 - val_categorical_accuracy: 0.7275 - 40s/epoch - 515ms/step
Epoch 20/30
78/78 - 49s - loss: 0.7273 - categorical_accuracy: 0.7385 - val_loss: 0.7650 - val_categorical_accuracy: 0.7209 - 49s/epoch - 622ms/step
Epoch 21/30
78/78 - 40s - loss: 0.7157 - categorical_accuracy: 0.7475 - val_loss: 0.7735 - val_categorical_accuracy: 0.7341 - 40s/epoch - 514ms/step
Epoch 22/30
78/78 - 40s - loss: 0.7312 - categorical_accuracy: 0.7410 - val_loss: 0.7576 - val_categorical_accuracy: 0.7333 - 40s/epoch - 509ms/step
Epoch 23/30
78/78 - 40s - loss: 0.7122 - categorical_accuracy: 0.7438 - val_loss: 0.7399 - val_categorical_accuracy: 0.7341 - 40s/epoch - 507ms/step
Epoch 24/30
78/78 - 40s - loss: 0.6952 - categorical_accuracy: 0.7410 - val_loss: 0.7320 - val_categorical_accuracy: 0.7391 - 40s/epoch - 511ms/step
Epoch 25/30
78/78 - 49s - loss: 0.7018 - categorical_accuracy: 0.7422 - val_loss: 0.7383 - val_categorical_accuracy: 0.7341 - 49s/epoch - 624ms/step
Epoch 26/30
78/78 - 48s - loss: 0.7040 - categorical_accuracy: 0.7527 - val_loss: 0.7405 - val_categorical_accuracy: 0.7308 - 48s/epoch - 620ms/step
Epoch 27/30
78/78 - 48s - loss: 0.6906 - categorical_accuracy: 0.7487 - val_loss: 0.7552 - val_categorical_accuracy: 0.7209 - 48s/epoch - 618ms/step
Epoch 28/30
78/78 - 40s - loss: 0.6748 - categorical_accuracy: 0.7503 - val_loss: 0.7394 - val_categorical_accuracy: 0.7267 - 40s/epoch - 507ms/step
Epoch 29/30
78/78 - 39s - loss: 0.6714 - categorical_accuracy: 0.7580 - val_loss: 0.7576 - val_categorical_accuracy: 0.7333 - 39s/epoch - 505ms/step
Epoch 30/30
78/78 - 39s - loss: 0.6803 - categorical_accuracy: 0.7507 - val_loss: 0.7276 - val_categorical_accuracy: 0.7440 - 39s/epoch - 504ms/step
```

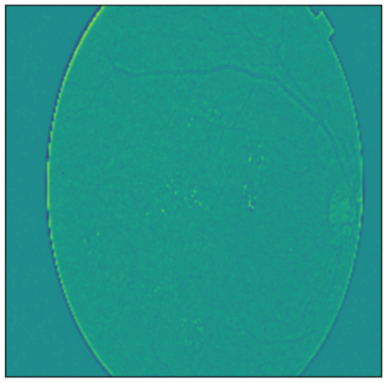
[ ]:
```
import os

# List subdirectories (classes) within the directory
subdirectories = os.listdir(file1)

# Count the number of samples in each class
class_counts = {}
for subdir in subdirectories:
    subdir_path = os.path.join(file1, subdir)
    if os.path.isdir(subdir_path):
        num_samples = len(os.listdir(subdir_path))
        class_counts[subdir] = num_samples
```

```
print("Class Counts:", class_counts)
```

```
Class Counts: {'Severe': 193, 'No_DR': 1815, 'Moderate': 1005, 'Mild': 370, 'Proliferate_DR': 295}
```

[ ]:
```
import cv2
import matplotlib.pyplot as plt
import numpy as np
import cv2

def prepare(image):
  IMG_SIZE=384
  img_array = cv2.imread('/content/drive/MyDrive/gaussian_filtered_images/gaussian_filtered_images/Moderate/000c1434d8d7.png', cv2.IMREAD_COLO
  new_array = cv2.resize(img_array, (IMG_SIZE, IMG_SIZE))
  return new_array.reshape(-1, IMG_SIZE, IMG_SIZE, 3)

y=cnn.predict([prepare('/content/drive/MyDrive/gaussian_filtered_images/gaussian_filtered_images/Moderate/000c1434d8d7.png')])
print('Retinal image with Moderate DR')
print(y)

img = cv2.imread('/content/drive/MyDrive/gaussian_filtered_images/gaussian_filtered_images/Moderate/000c1434d8d7.png',0)
plt.imshow(img, interpolation = 'bicubic')
plt.xticks([]), plt.yticks([])
plt.show()
```

```
1/1 [==============================] - 0s 475ms/step
Retinal image with Moderate DR
[[0.01261268 0.095369   0.49285436 0.22713201 0.17203192]]
```



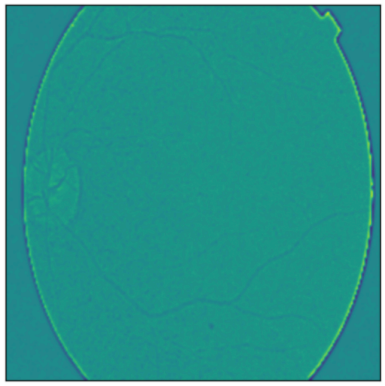
[ ]:
```
np.argmax(y)
```

[9]:
```
2
```

[ ]:
```
def prepare(image):
  IMG_SIZE=384
  img_array = cv2.imread('/content/drive/MyDrive/gaussian_filtered_images/gaussian_filtered_images/Mild/0024cdab0c1e.png', cv2.IMREAD_COLOR)
  new_array = cv2.resize(img_array, (IMG_SIZE, IMG_SIZE))
  return new_array.reshape(-1, IMG_SIZE, IMG_SIZE, 3)

x=cnn.predict([prepare('/content/drive/MyDrive/gaussian_filtered_images/gaussian_filtered_images/Mild/0024cdab0c1e.png')])
print('Retinal image with Mild DR')
```

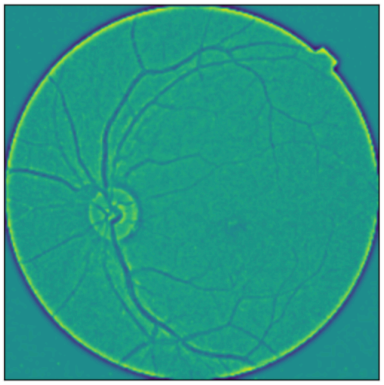
```
img = cv2.imread('/content/drive/MyDrive/gaussian_filtered_images/gaussian_filtered_images/Mild/0024cdab0c1e.png',0)
plt.imshow(img, interpolation = 'bicubic')
plt.xticks([]), plt.yticks([])
plt.show()
```

```
1/1 [==============================] - 0s 20ms/step
Retinal image with Mild DR
[[0.01291931 0.10616565 0.7590089  0.09557297 0.02633307]]
```



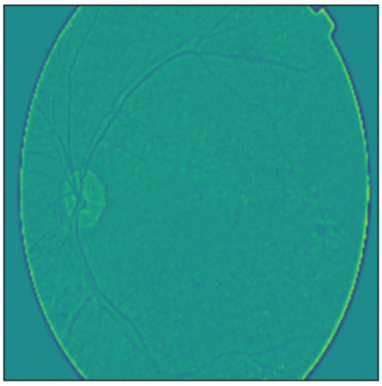
```
[ ]: def prepare(image):
       IMG_SIZE=384
       img_array = cv2.imread('/content/drive/MyDrive/gaussian_filtered_images/gaussian_filtered_images/No_DR/002c21358ce6.png', cv2.IMREAD_COLOR)
       new_array = cv2.resize(img_array, (IMG_SIZE, IMG_SIZE))
       return new_array.reshape(-1, IMG_SIZE, IMG_SIZE, 3)

     a=cnn.predict([prepare('/content/drive/MyDrive/gaussian_filtered_images/gaussian_filtered_images/No_DR/002c21358ce6.png')])
     print('Retinal image with No DR')
     print(a)

     img = cv2.imread('/content/drive/MyDrive/gaussian_filtered_images/gaussian_filtered_images/No_DR/002c21358ce6.png',0)
     plt.imshow(img, interpolation = 'bicubic')
     plt.xticks([]), plt.yticks([])
     plt.show()
```

```
1/1 [==============================] - 0s 20ms/step
Retinal image with No DR
[[0.00287436 0.17381945 0.5713271  0.06850819 0.18347098]]
```



```
[ ]: def prepare(image):
       IMG_SIZE=384
       img_array = cv2.imread('/content/drive/MyDrive/gaussian_filtered_images/gaussian_filtered_images/Severe/03c85870824c.png', cv2.IMREAD_COLOR)
       new_array = cv2.resize(img_array, (IMG_SIZE, IMG_SIZE))
       return new_array.reshape(-1, IMG_SIZE, IMG_SIZE, 3)

     b=cnn.predict([prepare('/content/drive/MyDrive/gaussian_filtered_images/gaussian_filtered_images/Severe/03c85870824c.png')])
     print('Retinal image with Severe DR')
     print(b)

     img = cv2.imread('/content/drive/MyDrive/gaussian_filtered_images/gaussian_filtered_images/Severe/03c85870824c.png',0)
     plt.imshow(img, interpolation = 'bicubic')
     plt.xticks([]), plt.yticks([])
     plt.show()
```

```
1/1 [==============================] - 0s 18ms/step
Retinal image with Severe DR
[[0.01014982 0.10296008 0.7444074  0.09289111 0.04959162]]
```



Appendix - 4

(Ensemble Learning - Unbalanced/Balanced)

Note: SMOTE was not used in Unbalanced Case (refer to Dataset analysis on page no. 52)

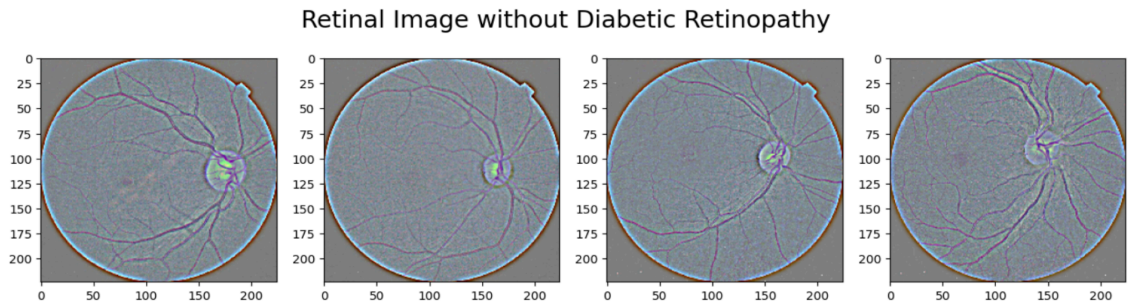
Imports

[6]:
```
import cv2
import matplotlib.pyplot as plt
import numpy as np
import os
import pandas as pd
from PIL import Image
import seaborn as sns
import matplotlib.pyplot as plt
import PIL
import plotly
import plotly.express as px
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers
from tensorflow.keras.models import Sequential
from tensorflow.keras.models import Model, load_model
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Dense, Flatten, Input, ZeroPadding2D, BatchNormalization, Activation
from tensorflow.keras.metrics import BinaryAccuracy, Precision, Recall
from tensorflow.keras import preprocessing
from tensorflow.keras.applications import DenseNet121, MobileNet, Xception, InceptionV3, ResNet101V2, VGG16
from tensorflow.keras.layers import Dropout, Activation, Lambda
from tensorflow.keras.callbacks import EarlyStopping
from keras.callbacks import ModelCheckpoint, EarlyStopping
from imblearn.over_sampling import SMOTE
import gc
from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix, classification_report, accuracy_score, recall_score, precision_score, f1_score
from sklearn.metrics import classification_report
import ssl
ssl._create_default_https_context = ssl._create_unverified_context
import warnings
warnings.filterwarnings("ignore")
```

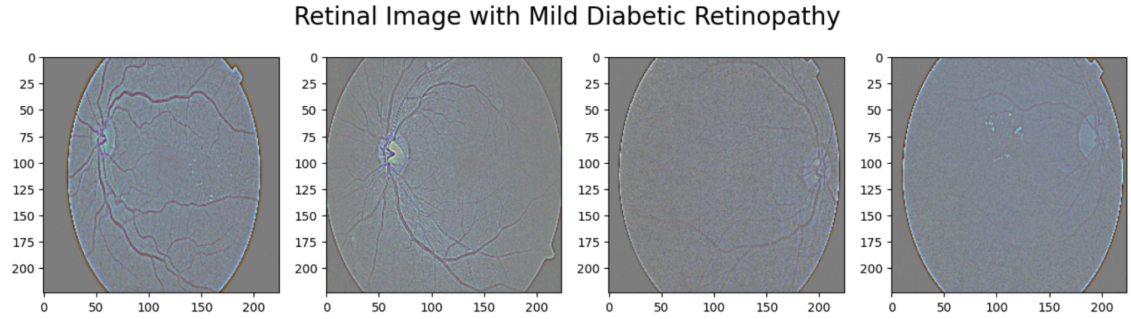
[3]:
```
#Locate the file in Personal Machine
path = '/Users/apple/Downloads/SDP/images'
```

[4]:
```
No_DR_Img = os.listdir(path + '/No_DR/')
Mild_Img = os.listdir(path + '/Mild/')
Moderate_Img = os.listdir(path + '/Moderate/')
Proliferate_DR_Img = os.listdir(path + '/Proliferate_DR/')
Severe_Img = os.listdir(path + '/Severe/')
```

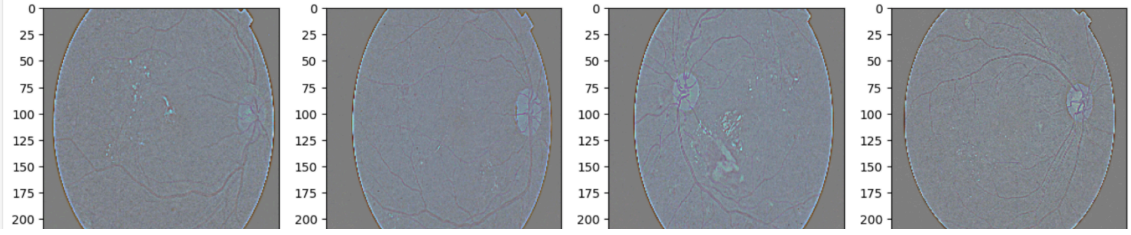
[5]:
```
#Viewing the dataset
fig = plt.figure(figsize=(16,4))
for i in range(4):
    plt.subplot(1, 4, i+1)
    img = cv2.imread(path+'/No_DR/'+ No_DR_Img[i])
    plt.imshow(img)
plt.suptitle("Retinal Image without Diabetic Retinopathy",fontsize=20)
plt.show()
```

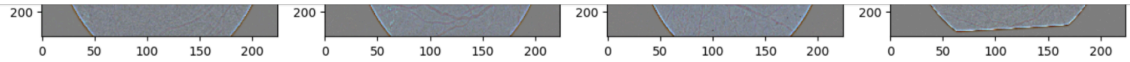


[6]:
```
#Viewing the dataset
fig = plt.figure(figsize=(16,4))
for i in range(4):
    plt.subplot(1, 4, i+1)
    img = cv2.imread(path+'/Mild/'+ Mild_Img[i])
    plt.imshow(img)
plt.suptitle("Retinal Image with Mild Diabetic Retinopathy",fontsize=20)
plt.show()
```

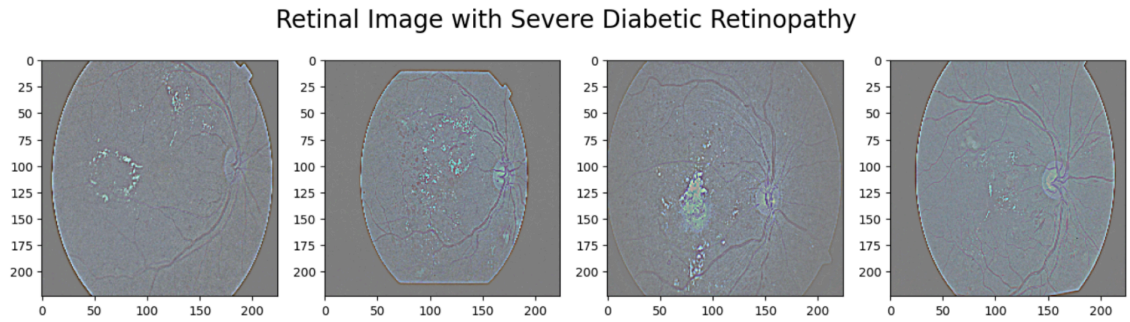


[7]:
```
#Viewing the dataset
fig = plt.figure(figsize=(16,4))
for i in range(4):
    plt.subplot(1, 4, i+1)
    img = cv2.imread(path+'/Moderate/'+ Moderate_Img[i])
    plt.imshow(img)
plt.suptitle("Retinal Image with Moderate Diabetic Retinopathy",fontsize=20)
plt.show()
```

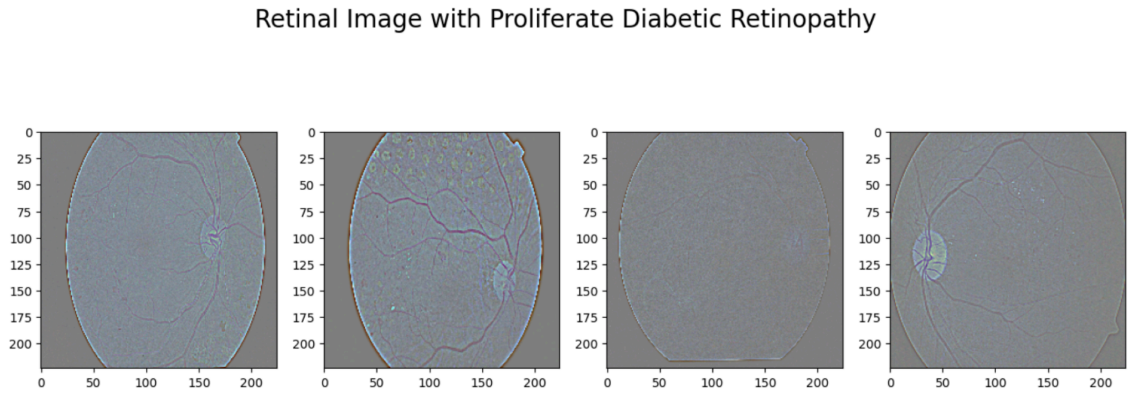




[8]:
```
#Viewing the dataset
fig = plt.figure(figsize=(16,4))
for i in range(4):
    plt.subplot(1, 4, i+1)
    img = cv2.imread(path+'/Severe/'+ Severe_Img[i])
    plt.imshow(img)
plt.suptitle("Retinal Image with Severe Diabetic Retinopathy",fontsize=20)
plt.show()
```



[9]:
```
#Viewing the dataset
fig = plt.figure(figsize=(16,4))
for i in range(4):
    plt.subplot(1, 4, i+1)
    img = cv2.imread(path+'/Proliferate_DR/'+ Proliferate_DR_Img[i])
    plt.imshow(img)
plt.suptitle("Retinal Image with Proliferate Diabetic Retinopathy",fontsize=20)
plt.show()
```



[10]:
```
Classes = {'No_DR':0, 'Mild':1, 'Moderate':2, 'Severe':3, 'Proliferate_DR':4}
```

Data Preprocessing

[11]:
```
for i in Classes:
    print(i)
```

```
No_DR
Mild
Moderate
Severe
Proliferate_DR
```

[12]:
```
#Normalizing pixel values
X = []
y = []

for i in Classes:
    folder_path ='/Users/apple/Downloads/SDP/images/' + i
    for j in os.listdir(folder_path):
        img = cv2.imread(folder_path + '/' + j)
        # normalize values
        img = img / 255 #-->Apply normalization because we want pixel values to be scaled to the range 0-1
        X.append(img) #-->Image Array
        y.append(Classes[i]) #-->Label Array
```

[14]:
```
X = np.array(X)
y = np.array(y)

X.shape, y.shape
```

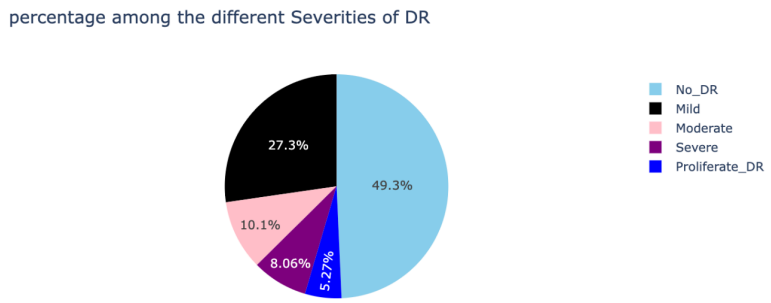
[14]:
```
((3662, 224, 224, 3), (3662,))
```

[15]:
```
gc.collect()
```

[15]:
```
23990
```

Dataset Analysis

[16]:
```
#Class division
Class_series=pd.Series(y)
lis=["No_DR","Mild","Moderate","Severe","Proliferate_DR"]
DR_or_not = Class_series.value_counts().tolist()
values = [DR_or_not[0], DR_or_not[1], DR_or_not[2], DR_or_not[3], DR_or_not[4]]
fig = px.pie(values=Class_series.value_counts(), names=lis , width=800, height=400, color_discrete_sequence=["skyblue","black","pink","purple"
             ,title="percentage among the different Severities of DR")
fig.show()
```



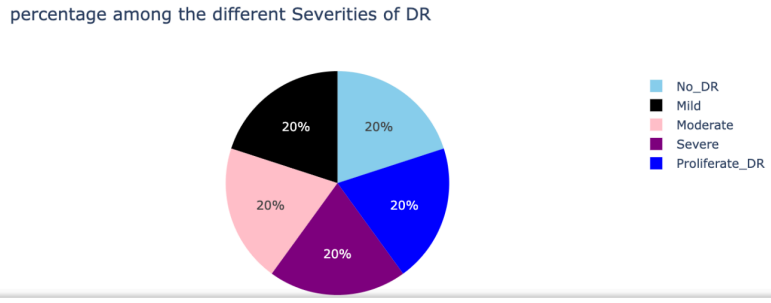
[19]:
```
#Synthetic Minority Over-sampling Technique (SMOTE) to address class imbalance in the dataset
#Using SMOTE to oversample the minority class(Edema) to avoid class imbalance
sm = SMOTE(random_state = 2)
# print(sm)
sm = SMOTE(random_state = 2)
# print(sm)
X, y = sm.fit_resample(X.reshape(X.shape[0], -1), y.ravel()) #-->This line applies the SMOTE algorithm to oversample the minority class (Edema
```

[20]:
```
X.shape[0]
```

[20]:
```
9025
```

[21]:
```
# reshaping is done on the oversampled dataset.
X = X.reshape(X.shape[0], 224, 224, 3)
```

[22]:
```
#Class division
Class_series=pd.Series(y)
lis=["No_DR","Mild","Moderate","Severe","Proliferate_DR"]
DR_or_not = Class_series.value_counts().tolist()
values = [DR_or_not[0], DR_or_not[1], DR_or_not[2], DR_or_not[3], DR_or_not[4]]
fig = px.pie(values=Class_series.value_counts(), names=lis , width=800, height=400, color_discrete_sequence=["skyblue","black","pink","purple"
             ,title="percentage among the different Severities of DR")
fig.show()
```



[24]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.20, random_state=32, stratify=y) # used stratify to balance the number o
```

[25]:
```
X.shape
```

[25]:
```
(9025, 224, 224, 3)
```

[26]:
```
X_train.shape
```

[26]:
```
(7220, 224, 224, 3)
```

[27]:
```
X_test.shape
```

[27]:
```
(1805, 224, 224, 3)
```

[28]:
```
y_train, y_test
```

[28]:
```
(array([4, 1, 4, ..., 1, 0, 0]), array([0, 1, 4, ..., 1, 0, 2]))
```

[29]:
```
X_train.shape, X_test.shape
```

[29]:
```
((7220, 224, 224, 3), (1805, 224, 224, 3))
```

[30]:
```
# Reshaping of X_train and X_test after splitting the dataset ensures that
# both the training and testing data are in the appropriate format for training and evaluating the model
X_train = X_train.reshape( (-1,224,224,3))
X_test = X_test.reshape((-1,224,224,3))
```

[31]:
```
X_train.shape, X_test.shape
```

[31]:
```
((7220, 224, 224, 3), (1805, 224, 224, 3))
```

Predefined Models

1. Densenet121

[32]:
```
# DenseNet121, MobileNet, Xception, InceptionV3, ResNet101V2
```

[33]:
```
# Define DenseNet121 model with pre-trained weights from ImageNet dataset,
# excluding the top fully connected layers.
Densenet_Model = DenseNet121(input_shape=(224,224,3),weights='imagenet',include_top=False)

# Allow the layers of DenseNet121 to be trainable.
Densenet_Model.trainable = True
# Flag to selectively set layers as trainable or not based on condition.
set_trainable = False

# Iterate through each layer in the DenseNet121 model.
for layer in Densenet_Model.layers:
    if layer.name == 'conv5_block16_0_bn':
        set_trainable = True
    if set_trainable:
        layer.trainable = True
    else:
        layer.trainable = False

for layer in Densenet_Model.layers[:]:
  if ('bn' in layer.name):
    trainable = False

# Flatten the output of DenseNet121 to prepare for additional dense layers.
x = Flatten()(Densenet_Model.output)
x = tf.keras.layers.Dense(128,activation='relu')(x)
x = tf.keras.layers.Dropout(0.4)(x)
prediction = Dense(5,activation='softmax')(x)

model = Model(inputs=Densenet_Model.input, outputs=prediction)
```

[34]:
```
model.summary()
```

```
Model: "functional_1"
```

[47]:
```
gc.collect()
```

[47]:
```
32439
```

[38]:
```
# Initialize empty lists to store features and labels for the dataset.
X=[]
y=[]

# Early stopping to prevent overfitting by monitoring validation loss.
es = EarlyStopping(monitor='val_accuracy', min_delta = 0.005, patience=10, verbose=1, mode='auto')
#Model Check Point
# Model checkpoint to save the best model based on validation accuracy.
# The model will be saved to the specified filepath whenever a new best validation accuracy is achieved.
mc = ModelCheckpoint(monitor='val_accuracy', filepath = '/Users/apple/Downloads/SDP/pretrained_models/B/model_densenet.keras', verbose=1, save
cd = [es,mc]
adam = keras.optimizers.Adam(learning_rate=0.00001)
model.compile(loss='sparse_categorical_crossentropy',
              optimizer = adam,
              metrics=['accuracy']
)
```

[39]:
```
# Train the model with the specified training and validation data.
model.fit(x=X_train,y=y_train,
          validation_data=(X_test,y_test),
          epochs=10,
          callbacks=cd,
          batch_size = 32,
          shuffle=True)
```

```
Epoch 1/10
226/226 ━━━━━━━━━━━━━━━━━━━━ 0s 972ms/step - accuracy: 0.9119 - loss: 0.2772
Epoch 1: val_accuracy improved from -inf to 0.89695, saving model to /Users/apple/Downloads/SDP/pretrained_models/B/model_densenet.keras
226/226 ━━━━━━━━━━━━━━━━━━━━ 291s 1s/step - accuracy: 0.9119 - loss: 0.2772 - val_accuracy: 0.8970 - val_loss: 0.3106
Epoch 2/10
226/226 ━━━━━━━━━━━━━━━━━━━━ 0s 1s/step - accuracy: 0.9278 - loss: 0.2449
Epoch 2: val_accuracy improved from 0.89695 to 0.90471, saving model to /Users/apple/Downloads/SDP/pretrained_models/B/model_densenet.keras
226/226 ━━━━━━━━━━━━━━━━━━━━ 295s 1s/step - accuracy: 0.9278 - loss: 0.2449 - val_accuracy: 0.9047 - val_loss: 0.2952
Epoch 3/10
226/226 ━━━━━━━━━━━━━━━━━━━━ 0s 1s/step - accuracy: 0.9327 - loss: 0.2197
Epoch 3: val_accuracy did not improve from 0.90471
226/226 ━━━━━━━━━━━━━━━━━━━━ 294s 1s/step - accuracy: 0.9328 - loss: 0.2197 - val_accuracy: 0.9047 - val_loss: 0.2778
Epoch 4/10
226/226 ━━━━━━━━━━━━━━━━━━━━ 0s 1s/step - accuracy: 0.9533 - loss: 0.1802
```

```
Epoch 4: val_accuracy improved from 0.90471 to 0.90859, saving model to /Users/apple/Downloads/SDP/pretrained_models/B/model_densenet.keras
226/226 ━━━━━━━━━━━━━━━━━━━━ 295s 1s/step - accuracy: 0.9532 - loss: 0.1802 - val_accuracy: 0.9086 - val_loss: 0.2573
Epoch 5/10
226/226 ━━━━━━━━━━━━━━━━━━━━ 0s 1s/step - accuracy: 0.9496 - loss: 0.1779
Epoch 5: val_accuracy improved from 0.90859 to 0.91025, saving model to /Users/apple/Downloads/SDP/pretrained_models/B/model_densenet.keras
226/226 ━━━━━━━━━━━━━━━━━━━━ 298s 1s/step - accuracy: 0.9496 - loss: 0.1779 - val_accuracy: 0.9102 - val_loss: 0.2596
Epoch 6/10
226/226 ━━━━━━━━━━━━━━━━━━━━ 0s 1s/step - accuracy: 0.9592 - loss: 0.1573
Epoch 6: val_accuracy improved from 0.91025 to 0.92133, saving model to /Users/apple/Downloads/SDP/pretrained_models/B/model_densenet.keras
226/226 ━━━━━━━━━━━━━━━━━━━━ 299s 1s/step - accuracy: 0.9592 - loss: 0.1573 - val_accuracy: 0.9213 - val_loss: 0.2348
Epoch 7/10
226/226 ━━━━━━━━━━━━━━━━━━━━ 0s 1s/step - accuracy: 0.9642 - loss: 0.1386
Epoch 7: val_accuracy improved from 0.92133 to 0.92742, saving model to /Users/apple/Downloads/SDP/pretrained_models/B/model_densenet.keras
226/226 ━━━━━━━━━━━━━━━━━━━━ 303s 1s/step - accuracy: 0.9642 - loss: 0.1386 - val_accuracy: 0.9274 - val_loss: 0.2297
Epoch 8/10
226/226 ━━━━━━━━━━━━━━━━━━━━ 0s 1s/step - accuracy: 0.9678 - loss: 0.1284
Epoch 8: val_accuracy improved from 0.92742 to 0.93573, saving model to /Users/apple/Downloads/SDP/pretrained_models/B/model_densenet.keras
226/226 ━━━━━━━━━━━━━━━━━━━━ 297s 1s/step - accuracy: 0.9678 - loss: 0.1284 - val_accuracy: 0.9357 - val_loss: 0.2106
Epoch 9/10
226/226 ━━━━━━━━━━━━━━━━━━━━ 0s 1s/step - accuracy: 0.9731 - loss: 0.1222
Epoch 9: val_accuracy did not improve from 0.93573
226/226 ━━━━━━━━━━━━━━━━━━━━ 304s 1s/step - accuracy: 0.9731 - loss: 0.1221 - val_accuracy: 0.9280 - val_loss: 0.2223
Epoch 10/10
226/226 ━━━━━━━━━━━━━━━━━━━━ 0s 1s/step - accuracy: 0.9778 - loss: 0.1055
Epoch 10: val_accuracy did not improve from 0.93573
226/226 ━━━━━━━━━━━━━━━━━━━━ 300s 1s/step - accuracy: 0.9778 - loss: 0.1055 - val_accuracy: 0.9352 - val_loss: 0.1995
```

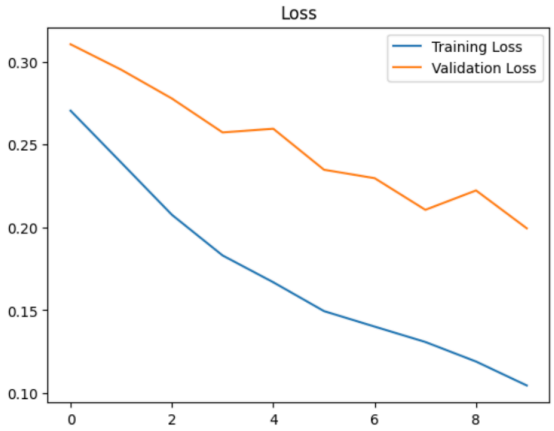
[39]:
```
<keras.src.callbacks.history.History at 0x31d9a9fa0>
```

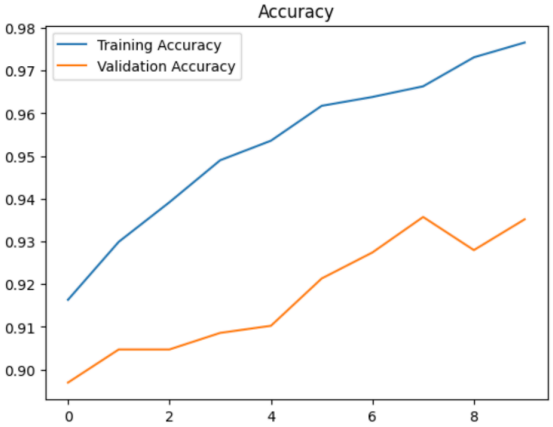
[40]:
```
history = model.history.history
model_loss=pd.DataFrame(model.history.history)
```

[41]:
```
def plot_metrics(history):
    train_loss = history['loss']
    val_loss = history['val_loss']
    train_acc = history['accuracy']
    val_acc = history['val_accuracy']
    # Loss
    plt.figure()
    plt.plot(train_loss, label='Training Loss')
    plt.plot(val_loss, label='Validation Loss')
    plt.title('Loss')
    plt.legend()
    plt.show()
     # Accuracy
```

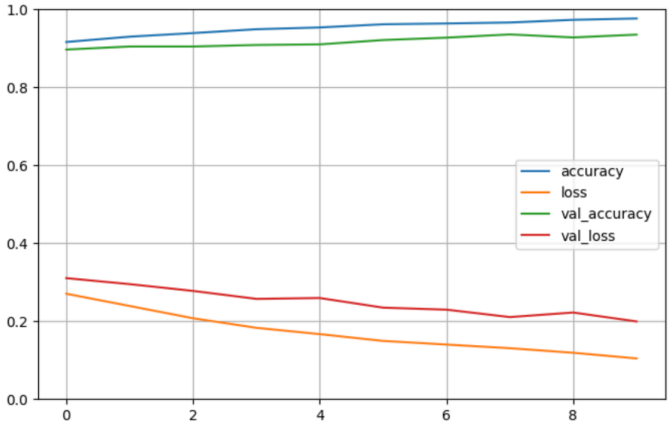
```
    plt.figure()
    plt.plot(train_acc, label='Training Accuracy')
    plt.plot(val_acc, label='Validation Accuracy')
    plt.title('Accuracy')
    plt.legend()
    plt.show()
```

[42]:
```
plot_metrics(history)
```





[43]:
```
model_loss.plot (figsize = (8,5))
plt.grid(True)
plt.gca().set_ylim(0,1)
plt.show()
```



[44]:
```
Model_evaluation = model.evaluate(X_train, y_train)
print(f"%\nAccuracy: {Model_evaluation[1]*100:.4f} %")
```

```
226/226 ━━━━━━━━━━━━━━━━━━━━ 226s 997ms/step - accuracy: 0.9909 - loss: 0.0512
%
Accuracy: 99.2798 %
```

[45]:
```
y_predicted_tf = model.predict(X_test)
```

```
57/57 ━━━━━━━━━━━━━━━━━━━━ 60s 1s/step
```

[46]:
```
y_predicted_tf
```

[46]:
```
array([[9.9912256e-01, 3.8380356e-04, 4.9240387e-04, 3.6907224e-07,
        1.0046249e-06],
       [1.2144869e-03, 9.8021567e-01, 1.9861399e-03, 2.2969423e-04,
        1.6354043e-02],
       [2.5322510e-05, 2.6985753e-04, 4.2581651e-02, 3.4953530e-03,
        9.5362777e-01],
       ...,
       [8.4456727e-03, 9.3981463e-01, 4.9706057e-02, 2.8865703e-04,
        1.7450625e-03],
       [9.9436474e-01, 1.2076830e-04, 5.1217782e-03, 7.7288765e-05,
        3.1543305e-04],
       [4.0182088e-02, 2.8892276e-01, 6.5446162e-01, 8.3057713e-03,
        8.1277499e-03]], dtype=float32)
```

[48]:
```
y_test
```

[48]:
```
array([0, 1, 4, ..., 1, 0, 2])
```

[49]:
```
y_predicted =[]
for i in range(1805):
    max_val = y_predicted_tf[i][0]
    classify = 0
    for j in range(5):
        if max_val<y_predicted_tf[i][j]:
            max_val = y_predicted_tf[i][j]
            classify = j
    y_predicted.append(classify)
```

[50]:
```
# y_predicted
```

[51]:
```
len(y_predicted)
```

[51]:
```
1805
```

[52]:
```
len(y_test)
```

[52]:
```
1805
```

[53]:

	Y_test	Y_predicted
0	0	0
1	1	1
2	4	4
3	4	4
4	0	0
5	1	1
6	1	1
7	3	3

[54]:
```
f1_score(y_test, y_predicted, average='micro')
```

[54]:
```
0.935180055401662
```

[55]:
```
recall_score(y_test, y_predicted, average='micro')
```

[55]:
```
0.935180055401662
```

[56]:
```
 precision_score(y_test, y_predicted, average='micro')
```

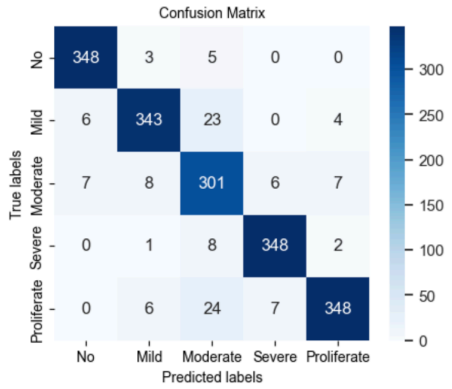
[56]:
```
0.935180055401662
```

[57]:
```
cm = confusion_matrix(y_test, y_predicted)
print(cm)
```

```
[[348   6   7   0   0]
 [  3 343   8   1   6]
 [  5  23 301   8  24]
 [  0   0   6 348   7]
 [  0   4   7   2 348]]
```

[58]:
```
cm = confusion_matrix(y_predicted,y_test)

plt.figure(figsize=(5, 4))
ax = plt.subplot()
sns.set(font_scale=1.0)
sns.heatmap(cm, annot=True, fmt='g', cmap="Blues", ax=ax);
```

```
# labels, title and ticks
ax.set_xlabel('Predicted labels', fontsize=10);ax.set_ylabel('True labels', fontsize=10);
ax.set_title('Confusion Matrix', fontsize=10);
ax.xaxis.set_ticklabels(['No','Mild','Moderate','Severe','Proliferate'], fontsize=10); ax.yaxis.set_ticklabels(['No','Mild','Moderate','Severe
```



2. MobileNet

[59]:
```
# from tensorflow.keras.applications import DenseNet121, MobileNet, Xception, InceptionV3, ResNet101V2
# from tensorflow.keras.models import Model
```

[120]:
```
# Choose the model
Mobilenet_model = MobileNet(input_shape=(224,224,3), weights='imagenet', include_top=False)  # For MobileNet

# Freeze layers up to a certain point
set_trainable = False
for layer in Mobilenet_model.layers:
    if layer.name == 'conv_pw_13_relu':  # Example layer name to start training from
        set_trainable = True
    if set_trainable:
        layer.trainable = True
    else:
        layer.trainable = False

# Add custom layers
x = Flatten()(Mobilenet_model.output)
x = Dense(128, activation='relu')(x)
x = Dropout(0.4)(x)
prediction = Dense(5, activation='softmax')(x)

# Create model
model = Model(inputs=Mobilenet_model.input, outputs=prediction)
```

[121]:
```
model.summary()
```

[122]:
```
gc.collect()
```

[122]:
```
3905
```

[123]:
```
X=[]
y=[]
early_stop = EarlyStopping(monitor='val_loss',mode='min',verbose=1,patience=5)
#Early Stopping
es = EarlyStopping(monitor='val_accuracy', min_delta = 0.005, patience=10, verbose=1, mode='auto')
#Model Check Point
mc = ModelCheckpoint(monitor='val_accuracy', filepath = '/Users/apple/Downloads/SDP/pretrained_models/B/model_mobilenet.keras', verbose=1, sav
cd = [es,mc]
adam = keras.optimizers.Adam(learning_rate=0.00001)
model.compile(loss='sparse_categorical_crossentropy',
              optimizer = adam,
              metrics=['accuracy']
)
```

[125]:
```
model.fit(x=X_train,y=y_train,
          validation_data=(X_test,y_test),
          epochs=10,
          callbacks=cd,
          batch_size = 32,
          shuffle=True)
```

```
Epoch 1/10
226/226 ━━━━━━━━━━━━━━━━━━━━ 0s 228ms/step - accuracy: 0.5558 - loss: 1.0913
Epoch 1: val_accuracy improved from -inf to 0.69751, saving model to /Users/apple/Downloads/SDP/pretrained_models/B/model_mobilenet.keras
226/226 ━━━━━━━━━━━━━━━━━━━━ 74s 325ms/step - accuracy: 0.5559 - loss: 1.0909 - val_accuracy: 0.6975 - val_loss: 0.7786
Epoch 2/10
226/226 ━━━━━━━━━━━━━━━━━━━━ 0s 236ms/step - accuracy: 0.7004 - loss: 0.7900
Epoch 2: val_accuracy improved from 0.69751 to 0.76787, saving model to /Users/apple/Downloads/SDP/pretrained_models/B/model_mobilenet.keras
226/226 ━━━━━━━━━━━━━━━━━━━━ 66s 292ms/step - accuracy: 0.7004 - loss: 0.7898 - val_accuracy: 0.7679 - val_loss: 0.6462
Epoch 3/10
226/226 ━━━━━━━━━━━━━━━━━━━━ 0s 238ms/step - accuracy: 0.7650 - loss: 0.6213
Epoch 3: val_accuracy improved from 0.76787 to 0.81773, saving model to /Users/apple/Downloads/SDP/pretrained_models/B/model_mobilenet.keras
226/226 ━━━━━━━━━━━━━━━━━━━━ 66s 292ms/step - accuracy: 0.7650 - loss: 0.6212 - val_accuracy: 0.8177 - val_loss: 0.5323
Epoch 4/10
226/226 ━━━━━━━━━━━━━━━━━━━━ 0s 231ms/step - accuracy: 0.8159 - loss: 0.4999
Epoch 4: val_accuracy improved from 0.81773 to 0.85485, saving model to /Users/apple/Downloads/SDP/pretrained_models/B/model_mobilenet.keras
226/226 ━━━━━━━━━━━━━━━━━━━━ 65s 285ms/step - accuracy: 0.8159 - loss: 0.4999 - val_accuracy: 0.8548 - val_loss: 0.4504
Epoch 5/10
226/226 ━━━━━━━━━━━━━━━━━━━━ 0s 233ms/step - accuracy: 0.8599 - loss: 0.4121
```

```
Epoch 6/10
226/226 ━━━━━━━━━━━━━━━━━━━━ 0s 234ms/step - accuracy: 0.8876 - loss: 0.3363
Epoch 6: val_accuracy improved from 0.86537 to 0.88864, saving model to /Users/apple/Downloads/SDP/pretrained_models/B/model_mobilenet.keras
226/226 ━━━━━━━━━━━━━━━━━━━━ 65s 289ms/step - accuracy: 0.8877 - loss: 0.3363 - val_accuracy: 0.8886 - val_loss: 0.3530
Epoch 7/10
226/226 ━━━━━━━━━━━━━━━━━━━━ 0s 233ms/step - accuracy: 0.9071 - loss: 0.2883
Epoch 7: val_accuracy improved from 0.88864 to 0.89917, saving model to /Users/apple/Downloads/SDP/pretrained_models/B/model_mobilenet.keras
226/226 ━━━━━━━━━━━━━━━━━━━━ 65s 287ms/step - accuracy: 0.9071 - loss: 0.2882 - val_accuracy: 0.8992 - val_loss: 0.3280
Epoch 8/10
226/226 ━━━━━━━━━━━━━━━━━━━━ 0s 232ms/step - accuracy: 0.9228 - loss: 0.2488
Epoch 8: val_accuracy improved from 0.89917 to 0.90970, saving model to /Users/apple/Downloads/SDP/pretrained_models/B/model_mobilenet.keras
226/226 ━━━━━━━━━━━━━━━━━━━━ 65s 286ms/step - accuracy: 0.9228 - loss: 0.2488 - val_accuracy: 0.9097 - val_loss: 0.3021
Epoch 9/10
226/226 ━━━━━━━━━━━━━━━━━━━━ 0s 235ms/step - accuracy: 0.9372 - loss: 0.2172
Epoch 9: val_accuracy improved from 0.90970 to 0.91634, saving model to /Users/apple/Downloads/SDP/pretrained_models/B/model_mobilenet.keras
226/226 ━━━━━━━━━━━━━━━━━━━━ 66s 290ms/step - accuracy: 0.9372 - loss: 0.2171 - val_accuracy: 0.9163 - val_loss: 0.2745
Epoch 10/10
226/226 ━━━━━━━━━━━━━━━━━━━━ 0s 235ms/step - accuracy: 0.9426 - loss: 0.1945
Epoch 10: val_accuracy improved from 0.91634 to 0.91801, saving model to /Users/apple/Downloads/SDP/pretrained_models/B/model_mobilenet.keras
226/226 ━━━━━━━━━━━━━━━━━━━━ 66s 290ms/step - accuracy: 0.9426 - loss: 0.1945 - val_accuracy: 0.9180 - val_loss: 0.2653
```

[125]:
```
<keras.src.callbacks.history.History at 0x375642b40>
```

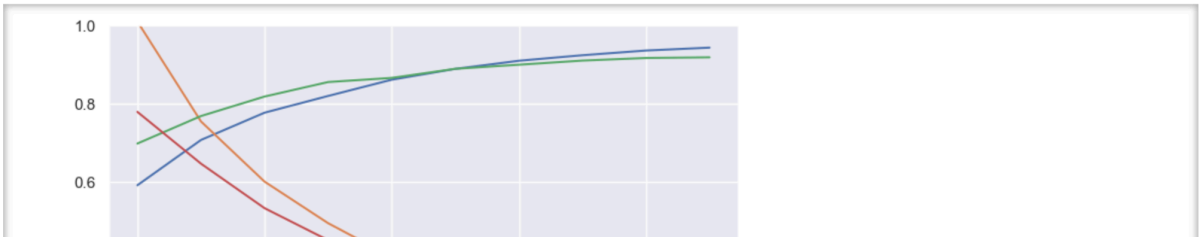
[126]:
```
history = model.history.history
model_loss=pd.DataFrame(model.history.history)
```

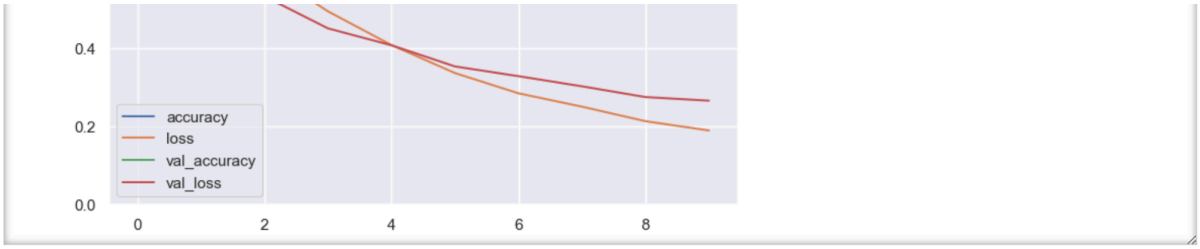
[127]:
```
def plot_metrics(history):

    train_loss = history['loss']
    val_loss = history['val_loss']
    train_acc = history['accuracy']
    val_acc = history['val_accuracy']
    # Loss
    plt.figure()
    plt.plot(train_loss, label='Training Loss')
    plt.plot(val_loss, label='Validation Loss')
    plt.title('Loss')
    plt.legend()
    plt.show()
     # Accuracy
    plt.figure()
    plt.plot(train_acc, label='Training Accuracy')
    plt.plot(val_acc, label='Validation Accuracy')
    plt.title('Accuracy')
```

```
    plt.legend()
    plt.show()
```

[128]:
```
plot_metrics(history)
```

[129]:
```
model_loss.plot (figsize = (8,5))
plt.grid(True)
plt.gca().set_ylim(0,1)
plt.show()
```





[130]:
```
Model_evaluation = model.evaluate(X_train, y_train)
print(f"%\nAccuracy: {Model_evaluation[1]*100:.4f} %")
```

```
226/226 ━━━━━━━━━━━━━━━━━━━━ 53s 235ms/step - accuracy: 0.9803 - loss: 0.1006
%
Accuracy: 98.0194 %
```

[131]:
```
y_predicted_tf = model.predict(X_test)
```

```
57/57 ━━━━━━━━━━━━━━━━━━━━ 13s 220ms/step
```

[132]:
```
y_predicted_tf
```

[132]:
```
array([[9.9944162e-01, 2.7261436e-04, 2.8361334e-04, 4.7180791e-07,
        1.7746205e-06],
       [2.5809961e-03, 8.4853768e-01, 1.9857649e-02, 2.3000722e-03,
        1.2672362e-01],
       [1.5860324e-04, 3.2464801e-03, 1.1495941e-01, 5.4707821e-02,
        8.2692772e-01],
       ...,
       [6.8370779e-03, 9.6252108e-01, 2.9182920e-02, 1.1209081e-03,
        3.3807810e-04],
       [9.9854124e-01, 1.5934835e-04, 1.2952015e-03, 2.0912969e-06,
        2.1131214e-06],
       [1.4097995e-02, 8.2215589e-01, 1.3111894e-01, 4.6122307e-03,
        2.8014958e-02]], dtype=float32)
```

[133]:
```
y_test
```

[133]:
```
array([0, 1, 4, ..., 1, 0, 2])
```

[134]:
```
y_predicted =[]
for i in range(1805):
    max_val = y_predicted_tf[i][0]
    classify = 0
    for j in range(5):
        if max_val<y_predicted_tf[i][j]:
            max_val = y_predicted_tf[i][j]
            classify = j
    y_predicted.append(classify)
```

[135]:
```
# y_predicted
```

[136]:
```
len(y_predicted)
```

[136]:
```
1805
```

[137]:
```
len(y_test)
```

[137]:
```
1805
```

[138]:
```
df = pd.DataFrame({"Y_test": y_test , "Y_predicted" : y_predicted})
df.head(8)
```

[138]:

	Y_test	Y_predicted
0	0	0
1	1	1
2	4	4
3	4	4
4	0	0
5	1	1
6	1	1
7	3	3

[139]:
```
f1_score(y_test, y_predicted, average='micro')
```

[139]:
```
0.918005540166205
```

[140]:
```
recall_score(y_test, y_predicted, average='micro')
```

[140]:
```
0.918005540166205
```

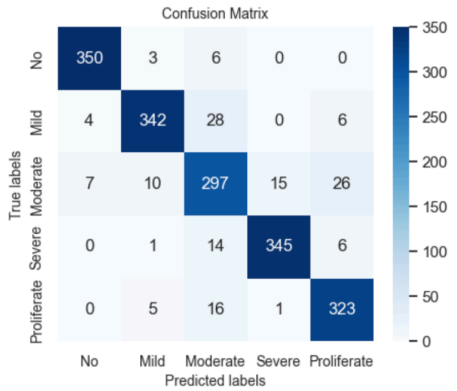
[141]:
```
 precision_score(y_test, y_predicted, average='micro')
```

[141]:
```
0.918005540166205
```

[142]:
```
cm = confusion_matrix(y_test, y_predicted)
print(cm)
```

```
[[350   4   7   0   0]
 [  3 342  10   1   5]
 [  6  28 297  14  16]
 [  0   0  15 345   1]
 [  0   6  26   6 323]]
```

[143]:
```
cm = confusion_matrix(y_predicted,y_test)

plt.figure(figsize=(5, 4))
ax = plt.subplot()
sns.set(font_scale=1.0)
sns.heatmap(cm, annot=True, fmt='g', cmap="Blues", ax=ax);

# labels, title and ticks
ax.set_xlabel('Predicted labels', fontsize=10);ax.set_ylabel('True labels', fontsize=10);
ax.set_title('Confusion Matrix', fontsize=10);
ax.xaxis.set_ticklabels(['No','Mild','Moderate','Severe','Proliferate'], fontsize=10); ax.yaxis.set_ticklabels(['No','Mild','Moderate','Severe
```



3. Xception

[85]:
```
# DenseNet121, MobileNet, Xception, InceptionV3, ResNet101V2
```

[145]:
```
# Choose the model
# Xception_model = MobileNet(input_shape=(224,224,3), weights='imagenet', include_top=False)  # For MobileNet
Xception_model = Xception(input_shape=(224,224,3), weights='imagenet', include_top=False)  # For Xception
# Xception_model = InceptionV3(weights='imagenet', include_top=False)  # For InceptionV3
# Xception_model = ResNet101V2(weights='imagenet', include_top=False)  # For ResNet101V2

# Freeze layers up to a certain point
set_trainable = False
for layer in Xception_model.layers:
    if layer.name == 'conv_pw_13_relu':  # Example layer name to start training from
        set_trainable = True
    if set_trainable:
        layer.trainable = True
    else:
        layer.trainable = False

# Add custom layers
x = Flatten()(Xception_model.output)
x = Dense(128, activation='relu')(x)
x = Dropout(0.4)(x)
prediction = Dense(5, activation='softmax')(x)

# Create model
model = Model(inputs=Xception_model.input, outputs=prediction)
```

[146]:
```
model.summary()
```

[147]:
```
gc.collect()
```

[147]:
```
15468
```

[148]:
```
X=[]
y=[]
early_stop = EarlyStopping(monitor='val_loss',mode='min',verbose=1,patience=5)
#Early Stopping
es = EarlyStopping(monitor='val_accuracy', min_delta = 0.005, patience=10, verbose=1, mode='auto')
#Model Check Point
mc = ModelCheckpoint(monitor='val_accuracy', filepath = '/Users/apple/Downloads/SDP/pretrained_models/B/model_xception.keras', verbose=1, save
cd = [es,mc]
adam = keras.optimizers.Adam(learning_rate=0.00001)
model.compile(loss='sparse_categorical_crossentropy',
              optimizer = adam,
              metrics=['accuracy']
)
```

[149]:
```
model.fit(x=X_train,y=y_train,
          validation_data=(X_test,y_test),
          epochs=10,
          callbacks=cd,
          batch_size = 32,
          shuffle=True)
```

```
Epoch 1/10
226/226 ━━━━━━━━━━━━━━━━━━━━ 0s 1s/step - accuracy: 0.4843 - loss: 1.2379
Epoch 1: val_accuracy improved from -inf to 0.64654, saving model to /Users/apple/Downloads/SDP/pretrained_models/B/model_xception.keras
226/226 ━━━━━━━━━━━━━━━━━━━━ 322s 1s/step - accuracy: 0.4846 - loss: 1.2372 - val_accuracy: 0.6465 - val_loss: 0.8910
Epoch 2/10
226/226 ━━━━━━━━━━━━━━━━━━━━ 0s 1s/step - accuracy: 0.6530 - loss: 0.8774
Epoch 2: val_accuracy improved from 0.64654 to 0.71357, saving model to /Users/apple/Downloads/SDP/pretrained_models/B/model_xception.keras
226/226 ━━━━━━━━━━━━━━━━━━━━ 307s 1s/step - accuracy: 0.6531 - loss: 0.8773 - val_accuracy: 0.7136 - val_loss: 0.7708
Epoch 3/10
226/226 ━━━━━━━━━━━━━━━━━━━━ 0s 1s/step - accuracy: 0.7228 - loss: 0.7514
Epoch 3: val_accuracy improved from 0.71357 to 0.74737, saving model to /Users/apple/Downloads/SDP/pretrained_models/B/model_xception.keras
226/226 ━━━━━━━━━━━━━━━━━━━━ 306s 1s/step - accuracy: 0.7228 - loss: 0.7513 - val_accuracy: 0.7474 - val_loss: 0.6901
Epoch 4/10
226/226 ━━━━━━━━━━━━━━━━━━━━ 0s 1s/step - accuracy: 0.7582 - loss: 0.6613
Epoch 4: val_accuracy improved from 0.74737 to 0.77064, saving model to /Users/apple/Downloads/SDP/pretrained_models/B/model_xception.keras
226/226 ━━━━━━━━━━━━━━━━━━━━ 306s 1s/step - accuracy: 0.7582 - loss: 0.6612 - val_accuracy: 0.7706 - val_loss: 0.6255
Epoch 5/10
226/226 ━━━━━━━━━━━━━━━━━━━━ 0s 1s/step - accuracy: 0.7914 - loss: 0.5948
Epoch 5: val_accuracy improved from 0.77064 to 0.79391, saving model to /Users/apple/Downloads/SDP/pretrained_models/B/model_xception.keras
```

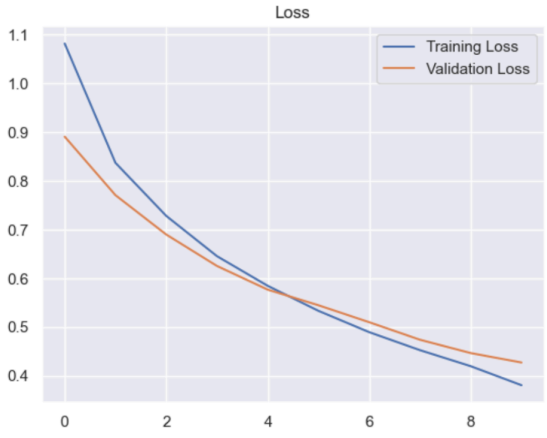
```
Epoch 6/10
226/226 ━━━━━━━━━━━━━━━━━━━━ 0s 1s/step - accuracy: 0.8217 - loss: 0.5288
Epoch 6: val_accuracy improved from 0.79391 to 0.80831, saving model to /Users/apple/Downloads/SDP/pretrained_models/B/model_xception.keras
226/226 ━━━━━━━━━━━━━━━━━━━━ 306s 1s/step - accuracy: 0.8217 - loss: 0.5289 - val_accuracy: 0.8083 - val_loss: 0.5448
Epoch 7/10
226/226 ━━━━━━━━━━━━━━━━━━━━ 0s 1s/step - accuracy: 0.8322 - loss: 0.4966
Epoch 7: val_accuracy improved from 0.80831 to 0.82161, saving model to /Users/apple/Downloads/SDP/pretrained_models/B/model_xception.keras
226/226 ━━━━━━━━━━━━━━━━━━━━ 308s 1s/step - accuracy: 0.8322 - loss: 0.4966 - val_accuracy: 0.8216 - val_loss: 0.5099
Epoch 8/10
226/226 ━━━━━━━━━━━━━━━━━━━━ 0s 1s/step - accuracy: 0.8463 - loss: 0.4563
Epoch 8: val_accuracy improved from 0.82161 to 0.83878, saving model to /Users/apple/Downloads/SDP/pretrained_models/B/model_xception.keras
226/226 ━━━━━━━━━━━━━━━━━━━━ 307s 1s/step - accuracy: 0.8463 - loss: 0.4563 - val_accuracy: 0.8388 - val_loss: 0.4740
Epoch 9/10
226/226 ━━━━━━━━━━━━━━━━━━━━ 0s 1s/step - accuracy: 0.8706 - loss: 0.4061
Epoch 9: val_accuracy improved from 0.83878 to 0.85485, saving model to /Users/apple/Downloads/SDP/pretrained_models/B/model_xception.keras
226/226 ━━━━━━━━━━━━━━━━━━━━ 307s 1s/step - accuracy: 0.8705 - loss: 0.4061 - val_accuracy: 0.8548 - val_loss: 0.4466
Epoch 10/10
226/226 ━━━━━━━━━━━━━━━━━━━━ 0s 1s/step - accuracy: 0.8823 - loss: 0.3867
Epoch 10: val_accuracy improved from 0.85485 to 0.86759, saving model to /Users/apple/Downloads/SDP/pretrained_models/B/model_xception.keras
226/226 ━━━━━━━━━━━━━━━━━━━━ 306s 1s/step - accuracy: 0.8823 - loss: 0.3867 - val_accuracy: 0.8676 - val_loss: 0.4272
```

[149]:
```
<keras.src.callbacks.history.History at 0x385a87f50>
```

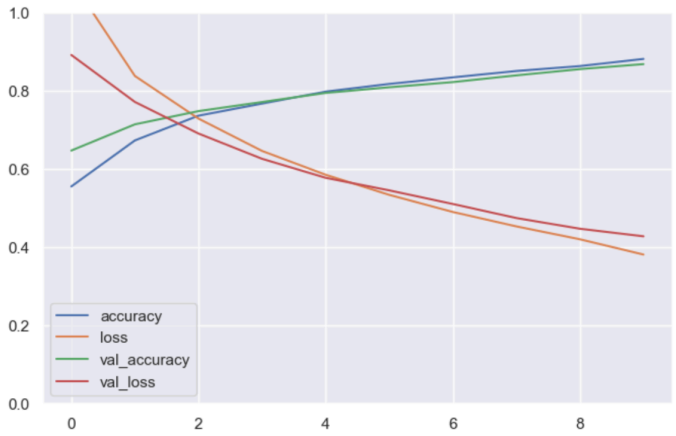
[150]:
```
history = model.history.history
model_loss=pd.DataFrame(model.history.history)
```

[151]:
```
def plot_metrics(history):

    train_loss = history['loss']
    val_loss = history['val_loss']
    train_acc = history['accuracy']
    val_acc = history['val_accuracy']
    # Loss
    plt.figure()
    plt.plot(train_loss, label='Training Loss')
    plt.plot(val_loss, label='Validation Loss')
    plt.title('Loss')
    plt.legend()
    plt.show()
     # Accuracy
    plt.figure()
     # Accuracy
    plt.figure()
    plt.plot(train_acc, label='Training Accuracy')
    plt.plot(val_acc, label='Validation Accuracy')
    plt.title('Accuracy')
    plt.legend()
    plt.show()
```

[152]:
```
plot_metrics(history)
```



[153]:
```
model_loss.plot (figsize = (8,5))
plt.grid(True)
plt.gca().set_ylim(0,1)
plt.show()
```



[154]:
```
Model_evaluation = model.evaluate(X_train, y_train)
print(f"%\nAccuracy: {Model_evaluation[1]*100:.4f} %")
```

```
226/226 ━━━━━━━━━━━━━━━━━━━━ 241s 1s/step - accuracy: 0.9260 - loss: 0.2998
%
Accuracy: 92.4377 %
```

[155]:
```
y_predicted_tf = model.predict(X_test)
```

```
57/57 ━━━━━━━━━━━━━━━━━━━━ 62s 1s/step
```

[156]:
```
y_predicted_tf
```

[156]:
```
array([[9.9587953e-01, 1.1804141e-03, 2.2603339e-03, 1.0831324e-04,
        5.7139754e-04],
       [2.3311370e-03, 8.9764434e-01, 4.2685814e-02, 1.8443197e-02,
        3.8895383e-02],
       [5.8563529e-03, 2.0777671e-02, 1.2036220e-01, 2.1425965e-01,
        6.3874418e-01],
```

```
       ...,
       [1.2538700e-01, 6.1943555e-01, 2.1087125e-01, 6.9831996e-03,
        3.7322875e-02],
       [9.9060810e-01, 6.1555845e-03, 2.8526315e-03, 3.2662796e-05,
        3.5105803e-04],
       [2.1131815e-02, 8.4472179e-02, 8.3458275e-01, 1.3941426e-02,
        4.5871813e-02]], dtype=float32)
```

[157]:
```
y_test
```

[157]:
```
array([0, 1, 4, ..., 1, 0, 2])
```

[158]:
```
y_predicted =[]
for i in range(1805):
    max_val = y_predicted_tf[i][0]
    classify = 0
    for j in range(5):
        if max_val<y_predicted_tf[i][j]:
            max_val = y_predicted_tf[i][j]
            classify = j
    y_predicted.append(classify)
```

[159]:
```
len(y_predicted)
```

[159]:
```
1805
```

[160]:
```
len(y_test)
```

[160]:
```
1805
```

[161]:
```
df = pd.DataFrame({"Y_test": y_test , "Y_predicted" : y_predicted})
df.head(8)
```

[161]:

	Y_test	Y_predicted
0	0	0
1	1	1
2	4	4
3	4	4
4	0	0
5	1	1
6	1	1
7	3	3

[162]:
```
f1_score(y_test, y_predicted, average='micro')
```

[162]:
```
0.867590027700831
```

[163]:
```
recall_score(y_test, y_predicted, average='micro')
```

[163]:
```
0.867590027700831
```

[164]:
```
precision_score(y_test, y_predicted, average='micro')
```

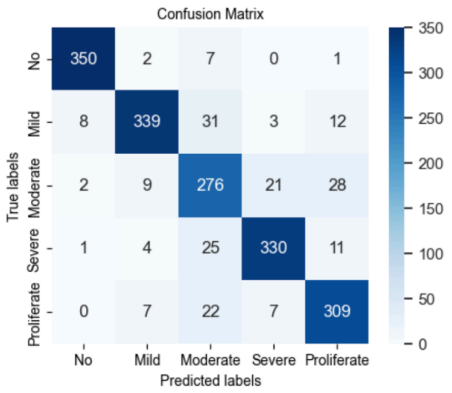
[164]:
```
0.867590027700831
```

[165]:
```
cm = confusion_matrix(y_test, y_predicted)
print(cm)
```

```
[[347   7   6   0   1]
 [  6 339   8   3   5]
 [  4  47 252  24  34]
 [  0   4  20 326  11]
 [  0  20  26  13 302]]
```

[166]:
```
cm = confusion_matrix(y_predicted,y_test)

plt.figure(figsize=(5, 4))
ax = plt.subplot()
sns.set(font_scale=1.0)
sns.heatmap(cm, annot=True, fmt='g', cmap="Blues", ax=ax);
```

```
# labels, title and ticks
ax.set_xlabel('Predicted labels', fontsize=10);ax.set_ylabel('True labels', fontsize=10);
ax.set_title('Confusion Matrix', fontsize=10);
ax.xaxis.set_ticklabels(['No','Mild','Moderate','Severe','Proliferate'], fontsize=10); ax.yaxis.set_ticklabels(['No','Mild','Moderate','Severe
```



4. InceptionV3

[167]:
```
# DenseNet121, MobileNet, Xception, InceptionV3, ResNet101V2
```

[60]:
```
# Choose the model
from keras.layers import Dropout
Inception_model = InceptionV3(input_shape=(224,224,3), weights='imagenet', include_top=False)  # For InceptionV3
# Inception_model = ResNet101V2(weights='imagenet', include_top=False)  # For ResNet101V2

# Freeze layers up to a certain point
set_trainable = False
for layer in Inception_model.layers:
    if layer.name == 'conv_pw_13_relu':  # Example layer name to start training from
        set_trainable = True
    if set_trainable:
        layer.trainable = True
    else:
        layer.trainable = False

# Add custom layers
x = Flatten()(Inception_model.output)
x = Dense(128, activation='relu')(x)
x = Dropout(0.4)(x)
prediction = Dense(5, activation='softmax')(x)

# Create model
model = Model(inputs=Inception_model.input, outputs=prediction)
```

[61]:
```
model.summary()
```

[62]:
```
gc.collect()
```

[62]:
```
1899
```

[63]:
```
X=[]
y=[]
early_stop = EarlyStopping(monitor='val_loss',mode='min',verbose=1,patience=5)
#Early Stopping
es = EarlyStopping(monitor='val_accuracy', min_delta = 0.005, patience=10, verbose=1, mode='auto')
#Model Check Point
mc = ModelCheckpoint(monitor='val_accuracy', filepath = '/Users/apple/Downloads/SDP/pretrained_models/B/model_inception.keras', verbose=1, sav
cd = [es,mc]
adam = keras.optimizers.Adam(learning_rate=0.00001)
model.compile(loss='sparse_categorical_crossentropy',
              optimizer = adam,
              metrics=['accuracy']
)
```

[64]:
```
model.fit(x=X_train,y=y_train,
          validation_data=(X_test,y_test),
          epochs=10,
          callbacks=cd,
          batch_size = 32,
          shuffle=True)
```

```
Epoch 1/10
226/226 ━━━━━━━━━━━━━━━━━━━━ 0s 636ms/step - accuracy: 0.4446 - loss: 1.3977
Epoch 1: val_accuracy improved from -inf to 0.63047, saving model to /Users/apple/Downloads/SDP/pretrained_models/B/model_inception.keras
226/226 ━━━━━━━━━━━━━━━━━━━━ 192s 837ms/step - accuracy: 0.4449 - loss: 1.3968 - val_accuracy: 0.6305 - val_loss: 0.9194
Epoch 2/10
226/226 ━━━━━━━━━━━━━━━━━━━━ 0s 643ms/step - accuracy: 0.6431 - loss: 0.9088
Epoch 2: val_accuracy improved from 0.63047 to 0.72022, saving model to /Users/apple/Downloads/SDP/pretrained_models/B/model_inception.keras
226/226 ━━━━━━━━━━━━━━━━━━━━ 181s 801ms/step - accuracy: 0.6432 - loss: 0.9087 - val_accuracy: 0.7202 - val_loss: 0.7506
Epoch 3/10
226/226 ━━━━━━━━━━━━━━━━━━━━ 0s 678ms/step - accuracy: 0.7168 - loss: 0.7327
Epoch 3: val_accuracy improved from 0.72022 to 0.77839, saving model to /Users/apple/Downloads/SDP/pretrained_models/B/model_inception.keras
226/226 ━━━━━━━━━━━━━━━━━━━━ 190s 842ms/step - accuracy: 0.7169 - loss: 0.7326 - val_accuracy: 0.7784 - val_loss: 0.6408
Epoch 4/10
226/226 ━━━━━━━━━━━━━━━━━━━━ 0s 666ms/step - accuracy: 0.7749 - loss: 0.6225
Epoch 4: val_accuracy improved from 0.77839 to 0.81939, saving model to /Users/apple/Downloads/SDP/pretrained_models/B/model_inception.keras
226/226 ━━━━━━━━━━━━━━━━━━━━ 188s 833ms/step - accuracy: 0.7749 - loss: 0.6224 - val_accuracy: 0.8194 - val_loss: 0.5526
Epoch 5/10
226/226 ━━━━━━━━━━━━━━━━━━━━ 0s 684ms/step - accuracy: 0.8204 - loss: 0.5245



Epoch 5/10
226/226 ━━━━━━━━━━━━━━━━━━━━ 0s 684ms/step - accuracy: 0.8204 - loss: 0.5245
Epoch 5: val_accuracy improved from 0.81939 to 0.83380, saving model to /Users/apple/Downloads/SDP/pretrained_models/B/model_inception.keras
226/226 ━━━━━━━━━━━━━━━━━━━━ 193s 852ms/step - accuracy: 0.8204 - loss: 0.5244 - val_accuracy: 0.8338 - val_loss: 0.5076
Epoch 6/10
226/226 ━━━━━━━━━━━━━━━━━━━━ 0s 679ms/step - accuracy: 0.8513 - loss: 0.4465
Epoch 6: val_accuracy improved from 0.83380 to 0.85983, saving model to /Users/apple/Downloads/SDP/pretrained_models/B/model_inception.keras
226/226 ━━━━━━━━━━━━━━━━━━━━ 191s 845ms/step - accuracy: 0.8513 - loss: 0.4464 - val_accuracy: 0.8598 - val_loss: 0.4586
Epoch 7/10
226/226 ━━━━━━━━━━━━━━━━━━━━ 0s 684ms/step - accuracy: 0.8817 - loss: 0.3878
Epoch 7: val_accuracy did not improve from 0.85983
226/226 ━━━━━━━━━━━━━━━━━━━━ 192s 852ms/step - accuracy: 0.8817 - loss: 0.3878 - val_accuracy: 0.8582 - val_loss: 0.4289
Epoch 8/10
226/226 ━━━━━━━━━━━━━━━━━━━━ 0s 685ms/step - accuracy: 0.8924 - loss: 0.3453
Epoch 8: val_accuracy improved from 0.85983 to 0.87645, saving model to /Users/apple/Downloads/SDP/pretrained_models/B/model_inception.keras
226/226 ━━━━━━━━━━━━━━━━━━━━ 193s 852ms/step - accuracy: 0.8924 - loss: 0.3453 - val_accuracy: 0.8765 - val_loss: 0.3868
Epoch 9/10
226/226 ━━━━━━━━━━━━━━━━━━━━ 0s 695ms/step - accuracy: 0.9109 - loss: 0.3039
Epoch 9: val_accuracy did not improve from 0.87645
226/226 ━━━━━━━━━━━━━━━━━━━━ 195s 862ms/step - accuracy: 0.9110 - loss: 0.3039 - val_accuracy: 0.8765 - val_loss: 0.3644
Epoch 10/10
226/226 ━━━━━━━━━━━━━━━━━━━━ 0s 688ms/step - accuracy: 0.9267 - loss: 0.2699
Epoch 10: val_accuracy improved from 0.87645 to 0.88864, saving model to /Users/apple/Downloads/SDP/pretrained_models/B/model_inception.keras
226/226 ━━━━━━━━━━━━━━━━━━━━ 194s 859ms/step - accuracy: 0.9267 - loss: 0.2699 - val_accuracy: 0.8886 - val_loss: 0.3374
```

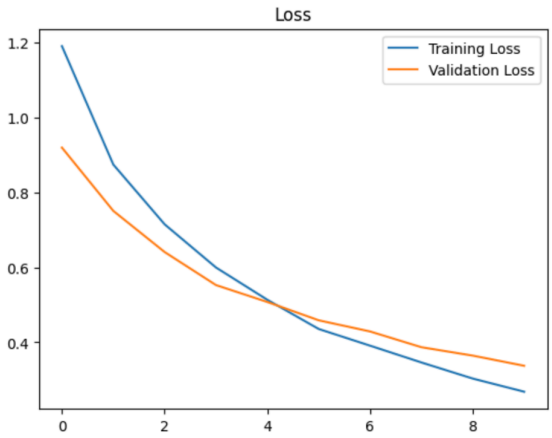
[64]:
```
<keras.src.callbacks.history.History at 0x312dab140>
```

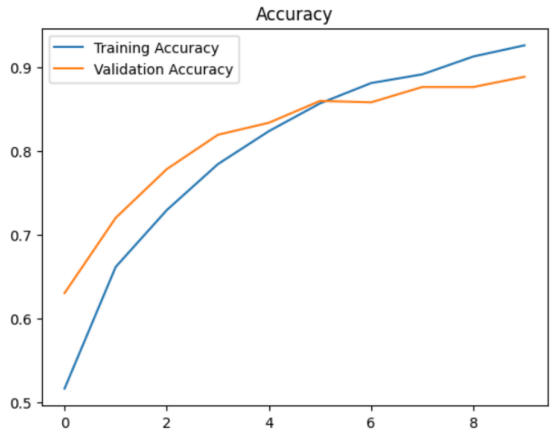
[65]:
```
history = model.history.history
model_loss=pd.DataFrame(model.history.history)
```

[66]:
```
def plot_metrics(history):

    train_loss = history['loss']
    val_loss = history['val_loss']
    train_acc = history['accuracy']
    val_acc = history['val_accuracy']
    # Loss
    plt.figure()
    plt.plot(train_loss, label='Training Loss')
    plt.plot(val_loss, label='Validation Loss')
    plt.title('Loss')
    plt.legend()
    plt.show()
```

```
    # Accuracy
    plt.figure()
    plt.plot(train_acc, label='Training Accuracy')
    plt.plot(val_acc, label='Validation Accuracy')
    plt.title('Accuracy')
    plt.legend()
    plt.show()
```

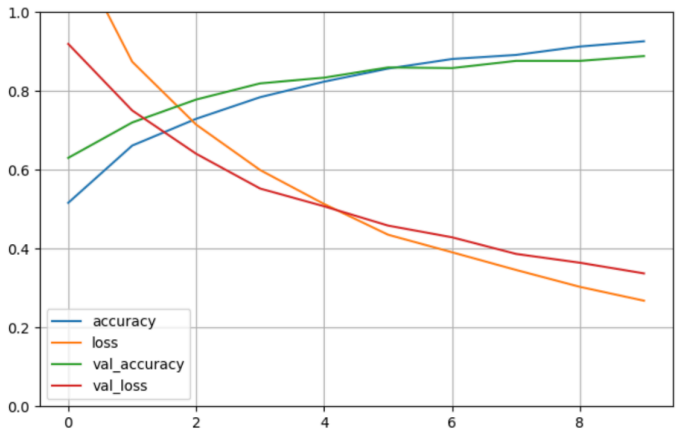
[67]:
```
plot_metrics(history)
```





[68]:
```
model_loss.plot (figsize = (8,5))
plt.grid(True)
plt.gca().set_ylim(0,1)
plt.show()
```

[68]:
```
model_loss.plot (figsize = (8,5))
plt.grid(True)
plt.gca().set_ylim(0,1)
plt.show()
```



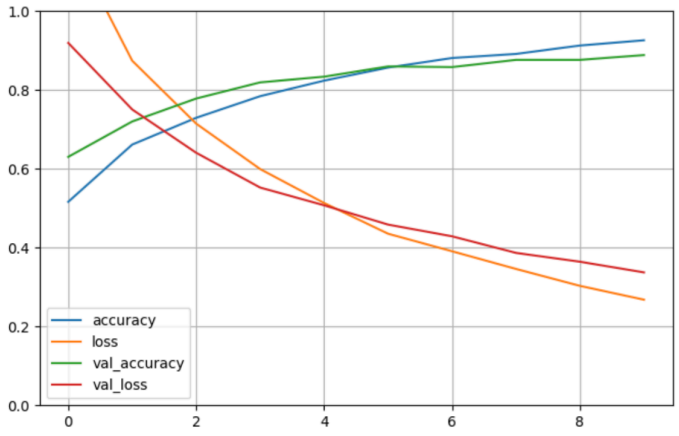
[69]:
```
Model_evaluation = model.evaluate(X_train, y_train)
print(f"%\nAccuracy: {Model_evaluation[1]*100:.4f} %")
```

```
226/226 ━━━━━━━━━━━━━━━━━━━━ 144s 637ms/step - accuracy: 0.9719 - loss: 0.1754
%
Accuracy: 97.2853 %
```

[70]:
```
y_predicted_tf = model.predict(X_test)
```

```
57/57 ━━━━━━━━━━━━━━━━━━━━ 37s 635ms/step
```

[68]:
```
model_loss.plot (figsize = (8,5))
plt.grid(True)
plt.gca().set_ylim(0,1)
plt.show()
```



[69]:
```
Model_evaluation = model.evaluate(X_train, y_train)
print(f"%\nAccuracy: {Model_evaluation[1]*100:.4f} %")
```

```
226/226 ━━━━━━━━━━━━━━━━━━━━ 144s 637ms/step - accuracy: 0.9719 - loss: 0.1754
%
Accuracy: 97.2853 %
```

[70]:
```
y_predicted_tf = model.predict(X_test)
```

```
57/57 ━━━━━━━━━━━━━━━━━━━━ 37s 635ms/step
```

[71]:
```
y_predicted_tf
```

[71]:
```
array([[9.89679635e-01, 7.61477044e-03, 2.58323574e-03, 6.95830677e-05,
        5.27166048e-05],
       [4.93013998e-03, 8.65904689e-01, 1.01492461e-02, 1.70141663e-02,
        1.02001727e-01],
       [1.23661547e-03, 2.23396556e-03, 7.43748397e-02, 1.14736974e-01,
        8.07417691e-01],
       ...,
       [5.71343079e-02, 7.66627133e-01, 1.45848289e-01, 1.71743997e-03,
        2.86727399e-02],
       [9.89474833e-01, 8.25096387e-03, 2.09794077e-03, 5.17732005e-05,
        1.24611979e-04],
       [1.13709964e-01, 4.11455110e-02, 6.54027164e-01, 3.43718380e-02,
        1.56745493e-01]], dtype=float32)
```

[72]:
```
y_test
```

[72]:
```
array([0, 1, 4, ..., 1, 0, 2])
```

[73]:
```
y_predicted =[]
for i in range(1805):
    max_val = y_predicted_tf[i][0]
    classify = 0
    for j in range(5):
        if max_val<y_predicted_tf[i][j]:
            max_val = y_predicted_tf[i][j]
            classify = j
    y_predicted.append(classify)
```

[74]:
```
len(y_predicted)
```

[74]:
```
1805
```

[75]:
```
len(y_test)
```

[75]:
```
1805
```

[76]:
```
df = pd.DataFrame({"Y_test": y_test , "Y_predicted" : y_predicted})
df.head(8)
```

[76]:

	Y_test	Y_predicted
0	0	0
1	1	1
2	4	4
3	4	4
4	0	0
5	1	2
6	1	1
7	3	3

[77]:
```
f1_score(y_test, y_predicted, average='micro')
```

[77]:
```
0.8886426592797784
```

[78]:
```
recall_score(y_test, y_predicted, average='micro')
```

[78]:
```
0.8886426592797784
```

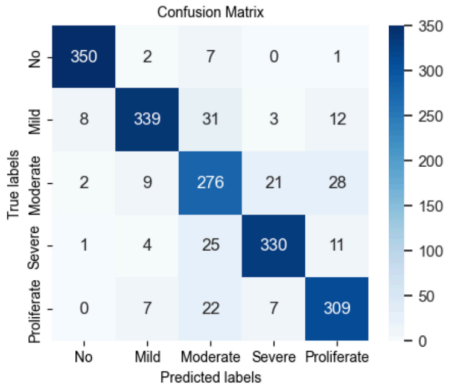
[79]:
```
precision_score(y_test, y_predicted, average='micro')
```

[79]:
```
0.8886426592797784
```

[80]:
```
cm = confusion_matrix(y_test, y_predicted)
print(cm)
```

```
[[350   8   2   1   0]
 [  2 339   9   4   7]
 [  7  31 276  25  22]
 [  0   3  21 330   7]
 [  1  12  28  11 309]]
```

[81]:
```
cm = confusion_matrix(y_predicted,y_test)

plt.figure(figsize=(5, 4))
ax = plt.subplot()
sns.set(font_scale=1.0)
sns.heatmap(cm, annot=True, fmt='g', cmap="Blues", ax=ax);

# labels, title and ticks
ax.set_xlabel('Predicted labels', fontsize=10);ax.set_ylabel('True labels', fontsize=10);
ax.set_title('Confusion Matrix', fontsize=10);
ax.xaxis.set_ticklabels(['No','Mild','Moderate','Severe','Proliferate'], fontsize=10); ax.yaxis.set_ticklabels(['No','Mild','Moderate','Severe
```



5. ResNet101V2

[83]:
```
# DenseNet121, MobileNet, Xception, InceptionV3, ResNet101V2
```

[84]:
```
# Choose the model
# Resnet_model = MobileNet(input_shape=(224,224,3), weights='imagenet', include_top=False)  # For MobileNet
# Resnet_model = Xception(input_shape=(224,224,3), weights='imagenet', include_top=False)  # For Xception
# Resnet_model = InceptionV3(weights='imagenet', include_top=False)  # For InceptionV3
Resnet_model = ResNet101V2(input_shape=(224,224,3), weights='imagenet', include_top=False)  # For ResNet101V2

# Freeze layers up to a certain point
set_trainable = False
for layer in Resnet_model.layers:
    if layer.name == 'conv_pw_13_relu':  # Example layer name to start training from
        set_trainable = True
    if set_trainable:
        layer.trainable = True
    else:
        layer.trainable = False

# Add custom layers
x = Flatten()(Resnet_model.output)
x = Dense(128, activation='relu')(x)
x = Dropout(0.4)(x)
prediction = Dense(5, activation='softmax')(x)

# Create model
model = Model(inputs=Resnet_model.input, outputs=prediction)
```

[ ]:
```
model.summary()
```

[45]:
```
gc.collect()
```

[45]:
```
147380
```

[85]:
```
X=[]
y=[]
early_stop = EarlyStopping(monitor='val_loss',mode='min',verbose=1,patience=5)
#Early Stopping
es = EarlyStopping(monitor='val_accuracy', min_delta = 0.005, patience=10, verbose=1, mode='auto')
#Model Check Point
mc = ModelCheckpoint(monitor='val_accuracy', filepath = '/Users/apple/Downloads/SDP/pretrained_models/B/model_resnet.keras', verbose=1, save_b
cd = [es,mc]
adam = keras.optimizers.Adam(learning_rate=0.00001)
model.compile(loss='sparse_categorical_crossentropy',
              optimizer = adam,
              metrics=['accuracy']
)
```

[86]:
```
model.fit(x=X_train,y=y_train,
          validation_data=(X_test,y_test),
          epochs=10,
          callbacks=cd,
          batch_size = 32,
          shuffle=True)
```

```
Epoch 1/10
226/226 ━━━━━━━━━━━━━━━━━━━━ 0s 2s/step - accuracy: 0.4417 - loss: 1.4630
Epoch 1: val_accuracy improved from -inf to 0.69806, saving model to /Users/apple/Downloads/SDP/pretrained_models/B/model_resnet.keras
226/226 ━━━━━━━━━━━━━━━━━━━━ 479s 2s/step - accuracy: 0.4421 - loss: 1.4618 - val_accuracy: 0.6981 - val_loss: 0.8134
Epoch 2/10
226/226 ━━━━━━━━━━━━━━━━━━━━ 0s 2s/step - accuracy: 0.6888 - loss: 0.8176
Epoch 2: val_accuracy improved from 0.69806 to 0.77175, saving model to /Users/apple/Downloads/SDP/pretrained_models/B/model_resnet.keras
226/226 ━━━━━━━━━━━━━━━━━━━━ 483s 2s/step - accuracy: 0.6888 - loss: 0.8174 - val_accuracy: 0.7717 - val_loss: 0.6335
Epoch 3/10
226/226 ━━━━━━━━━━━━━━━━━━━━ 0s 2s/step - accuracy: 0.7752 - loss: 0.6123
Epoch 3: val_accuracy improved from 0.77175 to 0.83102, saving model to /Users/apple/Downloads/SDP/pretrained_models/B/model_resnet.keras
226/226 ━━━━━━━━━━━━━━━━━━━━ 490s 2s/step - accuracy: 0.7752 - loss: 0.6122 - val_accuracy: 0.8310 - val_loss: 0.5170
Epoch 4/10
226/226 ━━━━━━━━━━━━━━━━━━━━ 0s 2s/step - accuracy: 0.8315 - loss: 0.4870
Epoch 4: val_accuracy improved from 0.83102 to 0.86150, saving model to /Users/apple/Downloads/SDP/pretrained_models/B/model_resnet.keras
226/226 ━━━━━━━━━━━━━━━━━━━━ 488s 2s/step - accuracy: 0.8316 - loss: 0.4869 - val_accuracy: 0.8615 - val_loss: 0.4403
Epoch 5/10
226/226 ━━━━━━━━━━━━━━━━━━━━ 0s 2s/step - accuracy: 0.8837 - loss: 0.3695
Epoch 5: val_accuracy improved from 0.86150 to 0.87701, saving model to /Users/apple/Downloads/SDP/pretrained_models/B/model_resnet.keras
226/226 ━━━━━━━━━━━━━━━━━━━━ 486s 2s/step - accuracy: 0.8837 - loss: 0.3694 - val_accuracy: 0.8770 - val_loss: 0.3894
```

```
Epoch 6/10
226/226 ━━━━━━━━━━━━━━━━━━━━ 0s 2s/step - accuracy: 0.9088 - loss: 0.3088
Epoch 6: val_accuracy improved from 0.87701 to 0.89030, saving model to /Users/apple/Downloads/SDP/pretrained_models/B/model_resnet.keras
226/226 ━━━━━━━━━━━━━━━━━━━━ 485s 2s/step - accuracy: 0.9088 - loss: 0.3088 - val_accuracy: 0.8903 - val_loss: 0.3495
Epoch 7/10
226/226 ━━━━━━━━━━━━━━━━━━━━ 0s 2s/step - accuracy: 0.9276 - loss: 0.2519
Epoch 7: val_accuracy improved from 0.89030 to 0.90914, saving model to /Users/apple/Downloads/SDP/pretrained_models/B/model_resnet.keras
226/226 ━━━━━━━━━━━━━━━━━━━━ 492s 2s/step - accuracy: 0.9276 - loss: 0.2519 - val_accuracy: 0.9091 - val_loss: 0.3129
Epoch 8/10
226/226 ━━━━━━━━━━━━━━━━━━━━ 0s 2s/step - accuracy: 0.9324 - loss: 0.2284
Epoch 8: val_accuracy did not improve from 0.90914
226/226 ━━━━━━━━━━━━━━━━━━━━ 491s 2s/step - accuracy: 0.9324 - loss: 0.2284 - val_accuracy: 0.9036 - val_loss: 0.3179
Epoch 9/10
226/226 ━━━━━━━━━━━━━━━━━━━━ 0s 2s/step - accuracy: 0.9488 - loss: 0.1925
Epoch 9: val_accuracy improved from 0.90914 to 0.91302, saving model to /Users/apple/Downloads/SDP/pretrained_models/B/model_resnet.keras
226/226 ━━━━━━━━━━━━━━━━━━━━ 487s 2s/step - accuracy: 0.9488 - loss: 0.1924 - val_accuracy: 0.9130 - val_loss: 0.2792
Epoch 10/10
226/226 ━━━━━━━━━━━━━━━━━━━━ 0s 2s/step - accuracy: 0.9580 - loss: 0.1571
Epoch 10: val_accuracy improved from 0.91302 to 0.91690, saving model to /Users/apple/Downloads/SDP/pretrained_models/B/model_resnet.keras
226/226 ━━━━━━━━━━━━━━━━━━━━ 491s 2s/step - accuracy: 0.9580 - loss: 0.1571 - val_accuracy: 0.9169 - val_loss: 0.2580
```

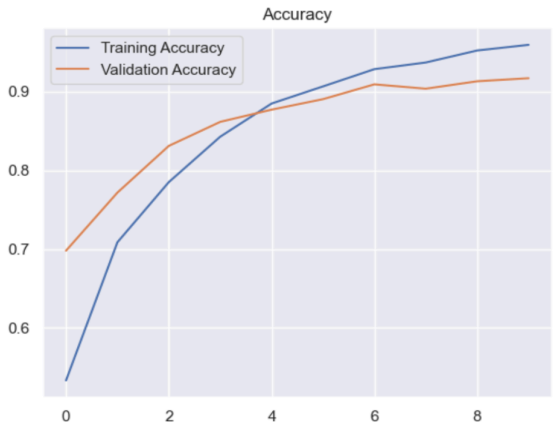
[86]:
```
<keras.src.callbacks.history.History at 0x352c07230>
```

[87]:
```
history = model.history.history
model_loss=pd.DataFrame(model.history.history)
```

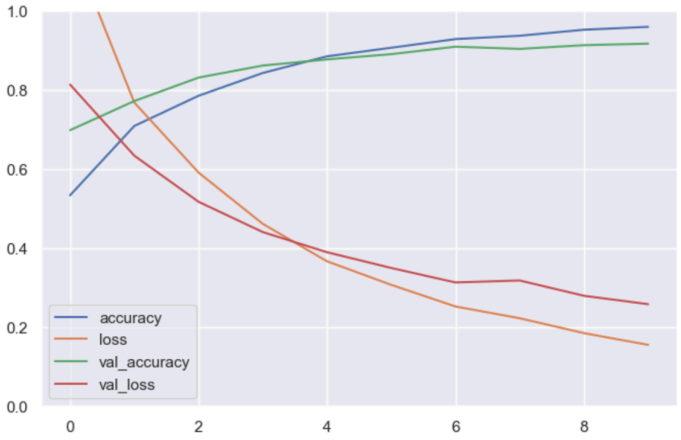
[88]:
```
def plot_metrics(history):

    train_loss = history['loss']
    val_loss = history['val_loss']
    train_acc = history['accuracy']
    val_acc = history['val_accuracy']
    # Loss
    plt.figure()
    plt.plot(train_loss, label='Training Loss')
    plt.plot(val_loss, label='Validation Loss')
    plt.title('Loss')
    plt.legend()
    plt.show()
```

```
    # Accuracy
    plt.figure()
    plt.plot(train_acc, label='Training Accuracy')
    plt.plot(val_acc, label='Validation Accuracy')
    plt.title('Accuracy')
    plt.legend()
    plt.show()
```

[89]:
```
plot_metrics(history)
```



[90]:
```
model_loss.plot (figsize = (8,5))
plt.grid(True)
plt.gca().set_ylim(0,1)
plt.show()
```



[91]:
```
Model_evaluation = model.evaluate(X_train, y_train)
print(f"%\nAccuracy: {Model_evaluation[1]*100:.4f} %")
```

```
226/226 ━━━━━━━━━━━━━━━━━━━━ 361s 2s/step - accuracy: 0.9835 - loss: 0.0887
%
Accuracy: 98.5319 %
```

[92]:
```
y_predicted_tf = model.predict(X_test)
```

```
57/57 ━━━━━━━━━━━━━━━━━━━━ 98s 2s/step
```

[93]:
```
y_predicted_tf
```

[93]:
```
array([[9.9608147e-01, 3.3999933e-03, 4.8067656e-04, 2.8147068e-07,
        3.7623835e-05],
       [9.1877882e-04, 9.5867139e-01, 1.3917095e-02, 2.5897162e-04,
        2.6233625e-02],
       [4.9304706e-04, 9.8027603e-04, 3.0084729e-02, 1.0242186e-02,
        9.5819974e-01],
       ...,
       [2.4239751e-02, 9.1462964e-01, 5.5412829e-02, 9.1396476e-04,
        4.8038266e-03],
       [9.8992997e-01, 2.2073961e-03, 7.6520662e-03, 2.5656777e-06,
        2.0799896e-04],
       [2.4213581e-03, 1.4301576e-01, 8.5339499e-01, 8.1282662e-04,
        3.5502721e-04]], dtype=float32)
```

[94]:
```
y_test
```

[94]:
```
array([0, 1, 4, ..., 1, 0, 2])
```

[95]:
```
y_predicted =[]
for i in range(1805):
    max_val = y_predicted_tf[i][0]
    classify = 0
    for j in range(5):
        if max_val<y_predicted_tf[i][j]:
            max_val = y_predicted_tf[i][j]
            classify = j
    y_predicted.append(classify)
```

[96]:
```
len(y_predicted)
```

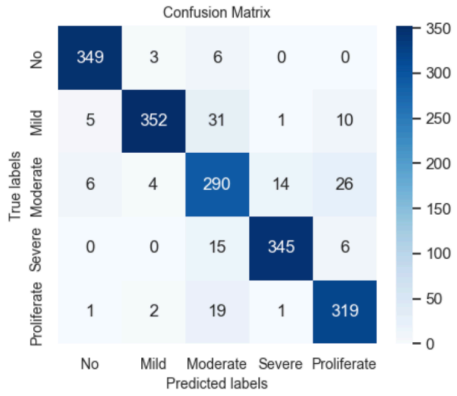
[96]:
```
1805
```

[97]:
```
len(y_test)
```

[97]:
```
1805
```

[98]:
```
df = pd.DataFrame({"Y_test": y_test , "Y_predicted" : y_predicted})
df.head(8)
```

[103]:
```
cm = confusion_matrix(y_predicted,y_test)

plt.figure(figsize=(5, 4))
ax = plt.subplot()
sns.set(font_scale=1.0)
sns.heatmap(cm, annot=True, fmt='g', cmap="Blues", ax=ax);

# labels, title and ticks
ax.set_xlabel('Predicted labels', fontsize=10);ax.set_ylabel('True labels', fontsize=10);
ax.set_title('Confusion Matrix', fontsize=10);
ax.xaxis.set_ticklabels(['No','Mild','Moderate','Severe','Proliferate'], fontsize=10); ax.yaxis.set_ticklabels(['No','Mild','Moderate','Severe
```



Ensemble Models

1. Average

[104]:
```
import tensorflow as tf

# Load individual pre-trained models without compiling them
keras_model1 = tf.keras.models.load_model('/Users/apple/Downloads/SDP/pretrained_models/B/model_densenet.keras', compile=False)
keras_model1._name = 'model1'
keras_model2 = tf.keras.models.load_model('/Users/apple/Downloads/SDP/pretrained_models/B/model_resnet.keras', compile=False)
keras_model2._name = 'model2'
keras_model3 = tf.keras.models.load_model('/Users/apple/Downloads/SDP/pretrained_models/B/model_mobilenet.keras', compile=False)
keras_model3._name = 'model3'
keras_model4 = tf.keras.models.load_model('/Users/apple/Downloads/SDP/pretrained_models/B/model_inception.keras', compile=False)
keras_model4._name = 'model4'
keras_model5 = tf.keras.models.load_model('/Users/apple/Downloads/SDP/pretrained_models/B/model_xception.keras', compile=False)
keras_model5._name = 'model5'

# Store all individual models in a list
models = [keras_model1, keras_model2, keras_model3, keras_model4, keras_model5]  # Stacking individual models in a list

# Stack models
model_input = tf.keras.Input(shape=(224, 224, 3))  # Input shape for the ensemble model
# Get the outputs from each model for the given input
model_outputs = [model(model_input) for model in models]  # Collect outputs of models in a list
ensemble_output = tf.keras.layers.Average()(model_outputs)  # Average outputs
ensemble_model = tf.keras.Model(inputs=model_input, outputs=ensemble_output)
ensemble_model.summary()
```

[107]:
```
early_stop = EarlyStopping(monitor='val_loss',mode='min',verbose=1,patience=5)
#Early Stopping
es = EarlyStopping(monitor='val_accuracy', min_delta = 0.005, patience=10, verbose=1, mode='auto')
#Model Check Point
mc = ModelCheckpoint(monitor='val_accuracy', filepath = '/Users/apple/Downloads/SDP/pretrained_models/B/model_avg.keras', verbose=1, save_best
cd = [es,mc]
adam = keras.optimizers.Adam(learning_rate=0.00001)
```

[108]:
```
ensemble_model.compile(loss='sparse_categorical_crossentropy', optimizer=adam, metrics=['accuracy'])
```

[109]:
```
# history = ensemble_model.fit(x=X_train, y=y_train, validation_data=(X_test, y_test), epochs=10, callbacks=cd, batch_size = 32, shuffle=True)
ensemble_model.fit(x=X_train, y=y_train, validation_data=(X_test, y_test), epochs=5, batch_size=32, shuffle=True, callbacks=cd)
```

```
Epoch 1/5
226/226 ━━━━━━━━━━━━━━━━━━━━ 0s 5s/step - accuracy: 0.9772 - loss: 0.1832
Epoch 1: val_accuracy improved from -inf to 0.93407, saving model to /Users/apple/Downloads/SDP/pretrained_models/B/model_avg.keras
226/226 ━━━━━━━━━━━━━━━━━━━━ 1344s 6s/step - accuracy: 0.9772 - loss: 0.1832 - val_accuracy: 0.9341 - val_loss: 0.2509
Epoch 2/5
226/226 ━━━━━━━━━━━━━━━━━━━━ 0s 5s/step - accuracy: 0.9819 - loss: 0.1568
Epoch 2: val_accuracy improved from 0.93407 to 0.94294, saving model to /Users/apple/Downloads/SDP/pretrained_models/B/model_avg.keras
226/226 ━━━━━━━━━━━━━━━━━━━━ 1328s 6s/step - accuracy: 0.9819 - loss: 0.1568 - val_accuracy: 0.9429 - val_loss: 0.2277
Epoch 3/5
226/226 ━━━━━━━━━━━━━━━━━━━━ 0s 5s/step - accuracy: 0.9850 - loss: 0.1448
Epoch 3: val_accuracy improved from 0.94294 to 0.94404, saving model to /Users/apple/Downloads/SDP/pretrained_models/B/model_avg.keras
226/226 ━━━━━━━━━━━━━━━━━━━━ 1319s 6s/step - accuracy: 0.9850 - loss: 0.1448 - val_accuracy: 0.9440 - val_loss: 0.2209
Epoch 4/5
226/226 ━━━━━━━━━━━━━━━━━━━━ 0s 5s/step - accuracy: 0.9879 - loss: 0.1281
Epoch 4: val_accuracy improved from 0.94404 to 0.94792, saving model to /Users/apple/Downloads/SDP/pretrained_models/B/model_avg.keras
226/226 ━━━━━━━━━━━━━━━━━━━━ 1415s 6s/step - accuracy: 0.9879 - loss: 0.1281 - val_accuracy: 0.9479 - val_loss: 0.2167
Epoch 5/5
226/226 ━━━━━━━━━━━━━━━━━━━━ 0s 5s/step - accuracy: 0.9903 - loss: 0.1143
Epoch 5: val_accuracy improved from 0.94792 to 0.94958, saving model to /Users/apple/Downloads/SDP/pretrained_models/B/model_avg.keras
226/226 ━━━━━━━━━━━━━━━━━━━━ 1263s 6s/step - accuracy: 0.9903 - loss: 0.1143 - val_accuracy: 0.9496 - val_loss: 0.2021
```

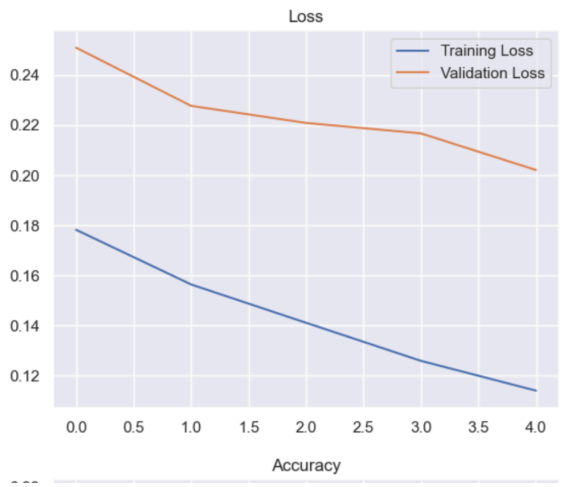
[109]:
```
<keras.src.callbacks.history.History at 0x3570f18b0>
```

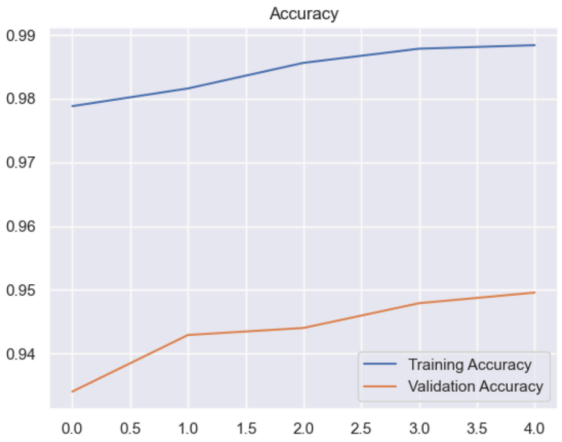
[110]:
```
history = ensemble_model.history.history
model_loss=pd.DataFrame(ensemble_model.history.history)
```

[111]:
```
def plot_metrics(history):
    train_loss = history['loss']
    val_loss = history['val_loss']
    train_acc = history['accuracy']
    val_acc = history['val_accuracy']
    # Loss
    plt.figure()
    plt.plot(train_loss, label='Training Loss')
    plt.plot(val_loss, label='Validation Loss')
    plt.title('Loss')
    plt.legend()
    plt.show()
     # Accuracy
    plt.figure()
    plt.plot(train_acc, label='Training Accuracy')
    plt.plot(val_acc, label='Validation Accuracy')
    plt.title('Accuracy')
    plt.legend()
    plt.show()
```

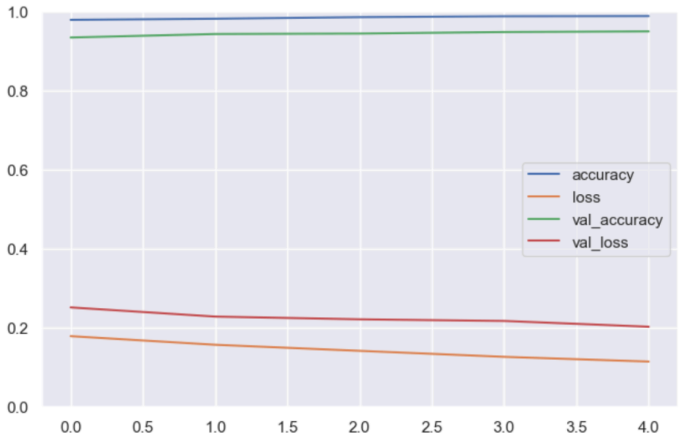
[112]:
```
plot_metrics(history)
```

[112]:
```
plot_metrics(history)
```





[113]:
```
model_loss.plot (figsize = (8,5))
plt.grid(True)
plt.gca().set_ylim(0,1)
plt.show()
```



[116]:
```
Model_evaluation = ensemble_model.evaluate(X_train, y_train)
print(f"%\nAccuracy: {Model_evaluation[1]*100:.4f} %")
```

```
226/226 ━━━━━━━━━━━━━━━━━━━━ 936s 4s/step - accuracy: 0.9917 - loss: 0.0714
%
Accuracy: 99.2798 %
```

[117]:
```
y_predicted_tf = ensemble_model.predict(X_test)
```

```
57/57 ━━━━━━━━━━━━━━━━━━━━ 243s 4s/step
```

[118]:
```
y_predicted_tf
```

[118]:
```
array([[9.9869776e-01, 1.0282127e-03, 2.5799242e-04, 1.2253332e-06,
        1.4808655e-05],
       [4.2746341e-04, 9.6979696e-01, 5.6315861e-03, 1.0902864e-03,
        2.3053711e-02],
       [4.2938898e-04, 1.2569998e-03, 2.9583532e-02, 2.5458997e-02,
        9.4327110e-01],
```

```
       ...,
       [1.6294371e-02, 9.0314370e-01, 7.3599383e-02, 3.4114518e-04,
        6.6214236e-03],
       [9.9759972e-01, 8.9603610e-04, 1.4438559e-03, 1.6043372e-06,
        5.8880145e-05],
       [3.9259572e-02, 2.6819393e-01, 6.6321772e-01, 2.1046705e-03,
        2.7224189e-02]], dtype=float32)
```

[119]:
```
len(y_test)
```

[119]:
```
1805
```

[120]:
```
y_predicted =[]
for i in range(1805):
    max_val = y_predicted_tf[i][0]
    classify = 0
    for j in range(5):
        if max_val<y_predicted_tf[i][j]:
            max_val = y_predicted_tf[i][j]
            classify = j
    y_predicted.append(classify)
```

[121]:
```
len(y_predicted)
```

[121]:
```
1805
```

[122]:
```
len(y_test)
```

[122]:
```
1805
```

[123]:
```
df = pd.DataFrame({"Y_test": y_test , "Y_predicted" : y_predicted})
df.head(8)
```

[123]:
```
df = pd.DataFrame({"Y_test": y_test , "Y_predicted" : y_predicted})
df.head(8)
```

[123]:

	Y_test	Y_predicted
0	0	0
1	1	1
2	4	4
3	4	4
4	0	0
5	1	1
6	1	1
7	3	3

[124]:
```
f1_score(y_test, y_predicted, average='micro')
```

[124]:
```
0.949584487534626
```

[125]:
```
recall_score(y_test, y_predicted, average='micro')
```

[125]:
```
0.949584487534626
```

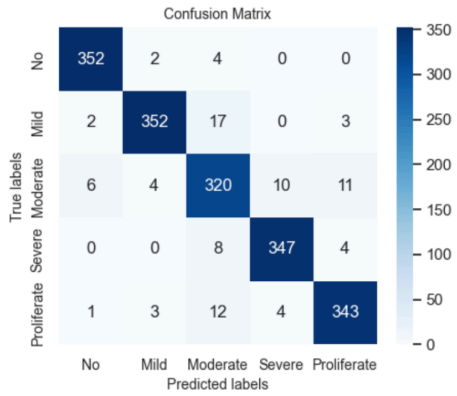
[126]:
```
precision_score(y_test, y_predicted, average='micro')
```

[126]:
```
0.949584487534626
```

[127]:
```
cm = confusion_matrix(y_test, y_predicted)
print(cm)
```

```
[[352   2   6   0   1]
 [  2 352   4   0   3]
 [  4  17 320   8  12]
 [  0   0  10 347   4]
 [  0   3  11   4 343]]
```

[128]:
```
cm = confusion_matrix(y_predicted,y_test)

plt.figure(figsize=(5, 4))
ax = plt.subplot()
sns.set(font_scale=1.0)
sns.heatmap(cm, annot=True, fmt='g', cmap="Blues", ax=ax);

# labels, title and ticks
ax.set_xlabel('Predicted labels', fontsize=10);ax.set_ylabel('True labels', fontsize=10);
ax.set_title('Confusion Matrix', fontsize=10);
ax.xaxis.set_ticklabels(['No','Mild','Moderate','Severe','Proliferate'], fontsize=10); ax.yaxis.set_ticklabels(['No','Mild','Moderate','Severe
```



2. Weighted Average

[135]:
```
import tensorflow as tf

# Define weights for weighted averaging
weights = [0.2, 0.2, 0.2, 0.2, 0.2]
model_input = tf.keras.Input(shape=(224, 224, 3))
# Perform weighted average of the model outputs
model_outputs = [model(model_input) for model in models]
ensemble_output = tf.keras.layers.Average()(model_outputs)  # Add weighted outputs
ensemble_model = tf.keras.Model(inputs=model_input, outputs=ensemble_output)

# Compile the ensemble model
# adam = tf.keras.optimizers.Adam()
```

[136]:
```
early_stop = EarlyStopping(monitor='val_loss',mode='min',verbose=1,patience=5)
#Early Stopping
es = EarlyStopping(monitor='val_accuracy', min_delta = 0.005, patience=10, verbose=1, mode='auto')
#Model Check Point
mc = ModelCheckpoint(monitor='val_accuracy', filepath = '/Users/apple/Downloads/SDP/pretrained_models/B/model_weighted_avg.keras', verbose=1,
cd = [es,mc]
adam = keras.optimizers.Adam(learning_rate=0.00001)
```

[137]:
```
class WeightedAverageLayer(tf.keras.layers.Layer):
    def __init__(self, w1, w2, w3, w4, w5, **kwargs):
        super (WeightedAverageLayer, self).__init__(**kwargs)
        self.w1 = w1
        self.w2 = w2
        self.w3 = w3
        self.w4 = w4
        self.w5 = w5

    def call(self, inputs):
        return self.w1 * inputs[0] + self.w2 * inputs[1]+ self.w3 * inputs[2] + + self.w4 * inputs[3]+ self.w5 * inputs[4]
```

[138]:
```
ensemble_output = WeightedAverageLayer(0.2, 0.2, 0.2, 0.2, 0.2)(model_outputs)  # Add weighted outputs
ensemble_model = tf.keras.Model(inputs=model_input, outputs=ensemble_output)
```

[139]:
```
ensemble_model.summary()
```

[140]:
```
ensemble_model.compile(loss=tf.losses.SparseCategoricalCrossentropy(from_logits=False), optimizer=adam, metrics=['accuracy'])
```

[141]:
```
history = ensemble_model.fit(x=X_train, y=y_train, validation_data=(X_test, y_test), epochs=5, callbacks=cd, batch_size = 32, shuffle=True)
```

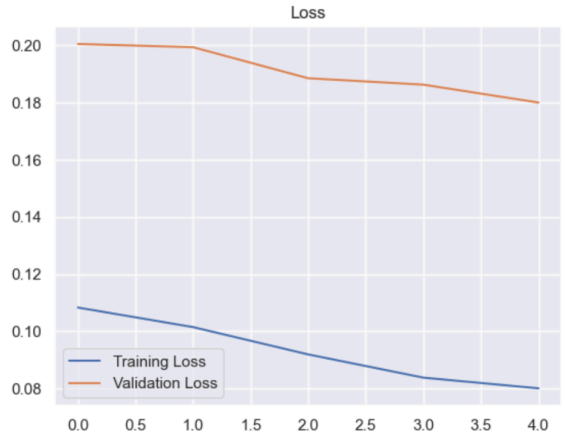
```
Epoch 1/5
226/226 ━━━━━━━━━━━━━━━━━━━━ 0s 5s/step - accuracy: 0.9881 - loss: 0.1101
Epoch 1: val_accuracy improved from -inf to 0.94848, saving model to /Users/apple/Downloads/SDP/pretrained_models/B/model_weighted_avg.keras
226/226 ━━━━━━━━━━━━━━━━━━━━ 1327s 6s/step - accuracy: 0.9881 - loss: 0.1101 - val_accuracy: 0.9485 - val_loss: 0.2005
Epoch 2/5
226/226 ━━━━━━━━━━━━━━━━━━━━ 0s 4s/step - accuracy: 0.9906 - loss: 0.0995
Epoch 2: val_accuracy did not improve from 0.94848
226/226 ━━━━━━━━━━━━━━━━━━━━ 1229s 5s/step - accuracy: 0.9906 - loss: 0.0995 - val_accuracy: 0.9457 - val_loss: 0.1994
Epoch 3/5
226/226 ━━━━━━━━━━━━━━━━━━━━ 0s 4s/step - accuracy: 0.9914 - loss: 0.0924
Epoch 3: val_accuracy improved from 0.94848 to 0.94958, saving model to /Users/apple/Downloads/SDP/pretrained_models/B/model_weighted_avg.ker
as
226/226 ━━━━━━━━━━━━━━━━━━━━ 1203s 5s/step - accuracy: 0.9914 - loss: 0.0924 - val_accuracy: 0.9496 - val_loss: 0.1885
Epoch 4/5
226/226 ━━━━━━━━━━━━━━━━━━━━ 0s 4s/step - accuracy: 0.9925 - loss: 0.0821
Epoch 4: val_accuracy did not improve from 0.94958
226/226 ━━━━━━━━━━━━━━━━━━━━ 1178s 5s/step - accuracy: 0.9925 - loss: 0.0821 - val_accuracy: 0.9468 - val_loss: 0.1863
Epoch 5/5
226/226 ━━━━━━━━━━━━━━━━━━━━ 0s 4s/step - accuracy: 0.9916 - loss: 0.0789
Epoch 5: val_accuracy improved from 0.94958 to 0.95402, saving model to /Users/apple/Downloads/SDP/pretrained_models/B/model_weighted_avg.ker
as
226/226 ━━━━━━━━━━━━━━━━━━━━ 1180s 5s/step - accuracy: 0.9916 - loss: 0.0789 - val_accuracy: 0.9540 - val_loss: 0.1800
```

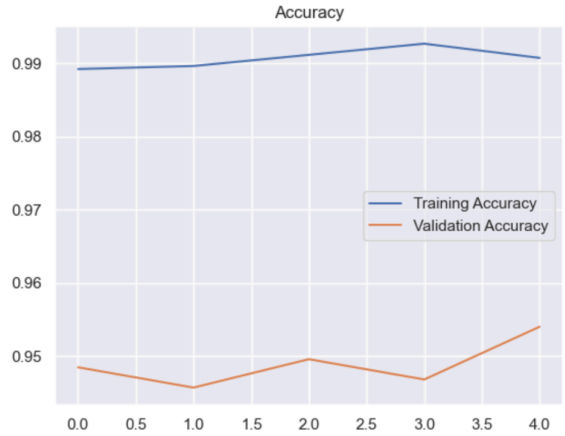
[142]:
```
h = ensemble_model.history.history
model_loss = pd.DataFrame(ensemble_model.history.history)
```

[143]:
```
def plot_metrics(history):
    train_loss = history['loss']
    val_loss = history['val_loss']
    train_acc = history['accuracy']
    val_acc = history['val_accuracy']
    # Loss
    plt.figure()
    plt.plot(train_loss, label='Training Loss')
    plt.plot(val_loss, label='Validation Loss')
    plt.title('Loss')
    plt.legend()
    plt.show()
```

```
    # Accuracy
    plt.figure()
    plt.plot(train_acc, label='Training Accuracy')
    plt.plot(val_acc, label='Validation Accuracy')
    plt.title('Accuracy')
    plt.legend()
    plt.show()
```

[144]:
```
plot_metrics(h)
```





[145]:
```
Model_evaluation = ensemble_model.evaluate(X_train, y_train)
print(f"%\nAccuracy: {Model_evaluation[1]*100:.4f} %")
```

```
226/226 ━━━━━━━━━━━━━━━━━━━━ 899s 4s/step - accuracy: 0.9935 - loss: 0.0486
%
Accuracy: 99.3906 %
```

[146]:
```
y_predicted_tf = ensemble_model.predict(X_test)
```

```
57/57 ━━━━━━━━━━━━━━━━━━━━ 231s 4s/step
```

[147]:
```
y_test
```

[147]:
```
array([0, 1, 4, ..., 1, 0, 2])
```

[152]:
```
y_predicted =[]
for i in range(1805):
    max_val = y_predicted_tf[i][0]
    classify = 0
```

```
    for j in range(5):
        if max_val<y_predicted_tf[i][j]:
            max_val = y_predicted_tf[i][j]
            classify = j
    y_predicted.append(classify)
```

[153]:
```
len(y_predicted)
```

[153]:
```
1805
```

[154]:
```
len(y_test)
```

[154]:
```
1805
```

[155]:
```
df = pd.DataFrame({"Y_test": y_test , "Y_predicted" : y_predicted})
df.head(8)
```

[155]:

	Y_test	Y_predicted
0	0	0
1	1	1
2	4	4
3	4	4
4	0	0
5	1	1
6	1	1
7	3	3

[156]:
```
f1_score(y_test, y_predicted, average='micro')
```

[156]:
```
0.9540166204986149
```

[157]:
```
recall_score(y_test, y_predicted, average='micro')
```

[157]:
```
0.9540166204986149
```

[158]:
```
precision_score(y_test, y_predicted, average='micro')
```

[158]:
```
0.9540166204986149
```

[159]:
```
cm = confusion_matrix(y_test, y_predicted)
print(cm)
```

```
[[352   3   5   0   1]
 [  2 354   2   1   2]
 [  4  20 315  11  11]
 [  0   0   7 353   1]
 [  0   3   7   3 348]]
```

[160]:
```
cm = confusion_matrix(y_predicted,y_test)

plt.figure(figsize=(5, 4))
ax = plt.subplot()
sns.set(font_scale=1.0)
sns.heatmap(cm, annot=True, fmt='g', cmap="Blues", ax=ax);

# labels, title and ticks
ax.set_xlabel('Predicted labels', fontsize=10);ax.set_ylabel('True labels', fontsize=10);
ax.set_title('Confusion Matrix', fontsize=10);
ax.xaxis.set_ticklabels(['No','Mild','Moderate','Severe','Proliferate'], fontsize=10); ax.yaxis.set_ticklabels(['No','Mild','Moderate','Severe
```

[160]:
```
cm = confusion_matrix(y_predicted,y_test)

plt.figure(figsize=(5, 4))
ax = plt.subplot()
sns.set(font_scale=1.0)
sns.heatmap(cm, annot=True, fmt='g', cmap="Blues", ax=ax);

# labels, title and ticks
ax.set_xlabel('Predicted labels', fontsize=10);ax.set_ylabel('True labels', fontsize=10);
ax.set_title('Confusion Matrix', fontsize=10);
ax.xaxis.set_ticklabels(['No','Mild','Moderate','Severe','Proliferate'], fontsize=10); ax.yaxis.set_ticklabels(['No','Mild','Moderate','Severe
```

Appendix - 5 (Web App)

main.py

```
import streamlit as st
from PIL import Image
from util import classify, set_background
import tensorflow as tf

set_background('./bgs/eye2.png')
# set title
st.title('Diabetic Retinopathy classification')
# set header
st.header('Please upload a Retinal Scan Image')
# upload file
file = st.file_uploader('', type=['jpeg', 'jpg', 'png'])

...
model = tf.keras.models.load_model('/Users/apple/Downloads/SDP/pretrained_models/B/model_mobilenet.keras')

# load class names
with open('./model/labels.txt', 'r') as f:
    class_names = [a[:-1].split(' ')[1] for a in f.readlines()]
    f.close()
print(class_names)

# display image
if file is not None:
    # image = Image.open(file).convert('RGB')
    image = Image.open(file)
    st.image(image, use_column_width=True)
    # classify image
    class_name, conf_score = classify(image, model, class_names)

    # write classification
    st.write("## {}".format(class_name))
    st.write("### score: {}%".format(int(conf_score * 1000) / 10))
```

util.py

```python
import base64
import streamlit as st
from PIL import ImageOps, Image
import numpy as np


2 usages  Eebad Reza
def set_background(image_file):...


2 usages  Eebad Reza *
def classify(image, model, class_names):
    """
    This function takes an image, a model, and a list of class names and returns the predicted class and confidence
    score of the image.

    Parameters:
        image (PIL.Image.Image): An image to be classified.
        model (tensorflow.keras.Model): A trained machine learning model for image classification.
        class_names (list): A list of class names corresponding to the classes that the model can predict.

    Returns:
        A tuple of the predicted class name and the confidence score for that prediction.
    """
    # convert image to (224, 224)
    image = ImageOps.fit(image, size: (224, 224), Image.Resampling.LANCZOS)

    # convert image to numpy array
    image_array = np.asarray(image)
    # normalize image
    normalized_image_array = (image_array.astype(np.float32) / 127.5) - 1
    # set model input
    data = np.ndarray(shape=(1, 224, 224, 3), dtype=np.float32)
    data[0] = normalized_image_array
```

```
# Predict the probabilities for each class
prediction = model.predict(data)
# Get the index of the class with the highest probability
index = np.argmax(prediction)

# Get the class name using the index
if index == 0:
    class_name = class_names[index]
else:
    class_name = 'DR, ' + class_names[index]

# Get the confidence score of the predicted class
confidence_score = prediction[0][index]
# Print the results
print(f"Predicted class: {class_name}")
print(f"Confidence score: {confidence_score}")
print(prediction, index, class_name, confidence_score)
return class_name, confidence_score
```

REFLECTION OF THE TEAM MEMBERS ON THE PROJECT

- **Syed Eebad Reza** made vital contributions to the project, including conducting a thorough literature survey, formulating the problem, leading experimentation with the models in Python. Designing and integration the model with the web app, while documenting the process. His efforts were pivotal in developing an effective system for diagnosing and classifying Diabetic Retinopathy.
- **Anurag Singh** contributed significantly to the project by conducting a thorough literature review, validating the results, designing and building the application, and carefully documenting the entire process. His involvement was pivotal in ensuring the project's success and keeping it clear for future endeavors.
- **Namrata Kumari Singh** made significant contributions to the project, including conducting the literature survey, researching the topic, designing and implementing the model, and thoroughly documenting the entire process. Her efforts were instrumental in advancing the diagnosis and classification of diabetic retinopathy, leading to impactful improvements in the field.
- **Tanmita Roy** contributed significantly throughout the project. She conducted extensive literature reviews, conducted experiments, analyzed results, designed solutions, and carefully documented the entire process. Her efforts helped advance the goals of the project and ensure the clarity and reproducibility of the research findings.