School of Computer Science and Engineering

(Computer Science & Engineering)

Faculty of Engineering & Technology

Jain Global Campus, Kanakapura Taluk - 562112

Ramanagara District, Karnataka, India

2023-2024

(VIII Semester)

A Project Report on

"Diabetic Retinopathy Detection"

Submitted in partial fulfilment for the award of the degree of

BACHELOR OF TECHNOLOGY

IN

COMPUTER SCIENCE AND ENGINEERING (ARTIFICIAL INTELLIGENCE)

Submitted by

Sreenivasan R S	**20BTRCA045**
Arvind Rathore V	**20BTRCA008**
Sai Ruthvik M	**20BTRCA029**

Under the guidance of

Prof. Narasimhayya B E

Assistant Professor

Department of Computer Science and Engineering

School of Computer Science & Engineering

Faculty of Engineering & Technology

JAIN (Deemed to-be University)

JAIN DEEMED-TO-BE UNIVERSITY | FACULTY OF ENGINEERING AND TECHNOLOGY

Department of Computer Science and Engineering

School of Computer Science & Engineering

Faculty of Engineering & Technology

Jain Global Campus, Kanakapura Taluk - 562112

Ramanagara District, Karnataka, India

CERTIFICATE

This is to certify that the project work titled **"Diabetic Retinopathy Detection"** is carried out by **Sreenivasan R S (20BTRCA045), Arvind Rathore V (20BTRCA018), Sai Ruthvik M (20BTRCA029)** as bonafide student(s) of Bachelor of Technology at the School of Engineering & Technology, Faculty of Engineering & Technology, JAIN (Deemed-to-be University), Bangalore in partial fulfillment for the award of degree in Bachelor / Master of Technology in Computer Scienceand Engineering, during the year **2023-2024**.

Prof. Narasimhayya B E
Assistant Professor
Dept. of CS&E,

Dr. Rajesh A
Program Head,
Computer Science and Engineering,
School of Computer Science & Engineering
Faculty of Engineering & Technology
JAIN (Deemed to-be University)

Dr. Geetha G
Director,
School of Computer Science & Engineering
Faculty of Engineering & Technology
JAIN (Deemed to-be University)

Name of the Examiner	Signature of Examiner
1.	
2.	

DECLARATION

We, **R S Sreenivasan (20BTRCA045), Arvind Rathore V. (20BTRCA008), Sai Ruthvik M. (20BTRCA029)** student of **VIII** semester B.Tech in **Computer Science and Engineering (Artificial Intelligence)**, at School of Engineering & Technology, Faculty of Engineering & Technology, **JAIN (Deemed to-be University)**, hereby declare that the internship work titled **“Diabetic Retinopathy Detection”** has been carried out by us and submitted in partial fulfilment for the award of degree in **Bachelor of Technology in Computer Science and Engineering (Artificial Intelligence)**, during the academic year **2023-2024**. Further, the matter presented in the work has not been submitted previously by anybody for the award of any degree or any diploma to any other University, to the bestof our knowledge and faith.

Name1: Sreenivasan R S
USN : 20BTRCA045

Signature:

Name2: Arvind Rathore V.
USN : 20BTRCA008

Signature:

Name3: Sai Ruthvik M.
USN : 20BTRCA029

Signature:

Place : Bangalore
Date :

ACKNOWLEDGEMENT

It is a great pleasure for me to acknowledge the assistance and support of a large number of individuals who have been responsible for the successful completion of this project work.

First, I take this opportunity to express my sincere gratitude to Faculty of Engineering & Technology, JAIN (Deemed to-be University) for providing me with a great opportunity to pursue my Bachelor's Degree in this institution.

I am deeply thankful to several individuals whose invaluable contributions have made this project a reality. I wish to extend my heartfelt gratitude to **Dr. Chenraj Roy Chand, Chancellor***, for his tireless commitment to fostering excellence in teaching and research at Jain (Deemed-to-be-University). I am also profoundly grateful to the honorable* **Vice Chancellor, Dr. Raj Singh, and Dr. Dinesh Nilkant, Pro Vice Chancellor***, for their unwavering support. Furthermore, I would like to express my sincere thanks to* **Dr. Jitendra Kumar Mishra, Registrar***, whose guidance has imparted invaluable qualities and skills that will serve us well in our future endeavors.*

I extend my sincere gratitude to **Dr. Hariprasad S A, Director** *of the Faculty of Engineering & Technology,* **and Dr. Geetha G, Director** *of the School of Computer Science & Engineering within the Faculty of Engineering & Technology, for their constant encouragement and expert advice. Additionally, I would like to express my appreciation to* **Dr. Krishnan Batri, Deputy Director (Course and Delivery), and Dr. V. Vivek, Deputy Director (Students & Industry Relations),** *for their invaluable contributions and support throughout this project.*

It is a matter of immense pleasure to express my sincere thanks to **Dr Dr. Rajesh A, Program Head, Computer Science and Engineering (Artificial Intelligence)***, School of Computer Science & Engineering Faculty of Engineering & Technology for providing right academic guidance that made my task possible.*

I would like to thank our **Prof. Narasimhayya B E***,* **Assistant Professor***,* **Dept. of Computer Science and Engineering (Artificial Intelligence)***, for sparing his/her valuable time to extend help in every step of my work, which paved the way for smooth progress and fruitful culmination of the project.*

I would like to thank our Project Coordinator ***Mrs. Sunena Rose M V****,* ***Assistant Professor, Dept. of Computer Science and Engineering (Artificial Intelligence)*** *and all the staff members of Computer Science and Engineering for their support.*

I am also grateful to my family and friends who provided me with every requirement throughout the course.

I would like to thank one and all who directly or indirectly helped me in completing the work successfully.

Signature of Student(s)

ABSTRACT

A crippling condition called diabetic retinopathy (DR) is brought on by persistently high blood sugar levels, which damage the retina and cause visual loss. Effective early-stage diabetic retinopathy detection techniques are essential to preventing irreversible vision loss as the prevalence of diabetes rises worldwide. In an effort to transform diagnostic processes and lessen the rising incidence of vision-related disorders, this research study introduces a novel method for the early diagnosis of diabetic retinopathy using Deep Learning and Artificial Intelligence (AI).

Our research uses state-of-the-art technology to evaluate retinal pictures and spot minute alterations that point to early-stage diabetic retinopathy. Trained on large-scale datasets, deep learning algorithms show a remarkable capacity to identify subtle patterns and anomalies, providing a quick and non-invasive way to diagnose. The present study endeavours to tackle the pressing requirement of screening techniques that are both economical and scalable, especially in areas where access to specialised medical care is restricted.

This research paper's use examples demonstrate the applicability of our method in practical settings. Early intervention and customised treatment programmes can be made possible by incorporating AI-based diabetic retinopathy detection into the current healthcare systems. By reducing the cost of advanced therapy and rehabilitation, this not only improves the quality of patient care but also adds to the overall cost-effectiveness of healthcare services.

The experiment also highlights the potential for telemedicine and remote monitoring applications, which could allow for timely intervention for people living in underserved or rural places. We envision a future where early diagnosis becomes a common reality, greatly lowering the global burden of diabetic retinopathy-related vision impairment, by deploying AI-based technologies in partnership with healthcare providers.

Keywords: Diabetic Retinopathy, Diagnosis, Computer Vision, CNN

TABLE OF CONTENTS

APPENDIX – II

INFORMATION REGARDING STUDENT

PHOTOGRAPH ALONG WITH GUIDE

LIST OF FIGURES

LIST OF TABLES

NOMENCLATURE USED

DR	Diabetic Retinopathy
CNN	Convolutional Neural Network
TFLOPS	Tensor Floating Point Operations per Second

Chapter 1

INTRODUCTION

1.1. Overview

Diabetic retinopathy (DR) poses a significant threat to vision by affecting the delicate blood vessels in the retina, and its asymptomatic early stages make it a leading cause of blindness globally. Given the rising incidence of diabetes worldwide, the demand for precise and swift diagnostic techniques for DR is paramount. This paper addresses the urgency for effective screening instruments and explores the potential of cutting-edge technologies, particularly in computer imaging and artificial intelligence (AI), to enhance the identification of diabetic retinopathy.

1.2. Motivation

The increasing global prevalence of diabetes necessitates the development of efficient screening and diagnostic tools. Early detection of diabetic retinopathy allows for effective intervention, potentially halting disease progression and minimizing vision loss. Traditional clinical procedures rely on labor-intensive manual assessments, making real-time classifications challenging. Motivated by the need for prompt and accurate interventions, the study emphasizes the transformative potential of computer-aided diagnostic systems, leveraging AI technologies like convolutional neural networks (CNNs) to revolutionize diabetic retinopathy identification

1.3. Main Objectives

The primary objectives of this study are to explore the application of Convolutional Neural Networks (CNNs) in diabetic retinopathy identification and contribute to the ongoing discourse on this critical health issue. The focus is on improving patient outcomes and enabling timely interventions by streamlining and enhancing the accuracy of diabetic retinopathy classifications through the integration of deep learning and artificial intelligence. The study aims to surpass the limitations of conventional diagnostic approaches, paving the way for more efficient and effective methods in the detection and management of diabetic retinopathy

1.4. Scope

The scope of this research encompasses an in-depth exploration of the potential applications of Convolutional Neural Networks (CNNs) in diabetic retinopathy detection. The study will delve into the intricacies of retinal images and the nuanced analysis required for a more advanced diagnostic approach. By examining the limitations of existing techniques, the research aims to broaden the scope of diagnostic capabilities and usher in a new era in ophthalmic care. The investigation into CNNs' use cases and applications in diabetic retinopathy detection is expected to provide valuable insights, with the ultimate goal of contributing to improved patient care and outcomes.

Chapter 2

LITERATURE SURVEY

Unnati V.Shukla; Koushik Tripathy [1]. The study introduces an advanced methodology for detecting diabetic retinopathy severity through fundus images, employing an ensemble of classifiers. The process commences with meticulous data pre-processing, incorporating adaptive equalization, colour normalization, and Gaussian filtering, coupled with the removal of optic disc and blood vessels to enhance image quality. Subsequent image segmentation techniques are applied to extract relevant markers and features pivotal for the classification process. Classification is accomplished through an ensemble of classifiers, utilizing a "weight voting" technique that exhibits varied performance across distinct severity levels. The system demonstrates remarkable accuracy in distinguishing different stages of diabetic retinopathy, highlighting its efficacy as an advanced diagnostic tool. The success of the robust and reliable classification system can be attributed to the effectiveness of the employed image pre-processing techniques, collectively offering clinicians a valuable and sophisticated tool for accurate assessment and timely intervention in diabetic retinopathy severity detection. This research significantly contributes to the field of medical image analysis, providing a nuanced and effective approach to diabetic retinopathy severity detection, with the potential to enhance patient outcomes and inform clinical decision-making.

Ankita Gupta , Rita Chhikara[2]. The study introduces an advanced methodology for detecting diabetic retinopathy severity through fundus images, employing an ensemble of classifiers. The process commences with meticulous data preprocessing, incorporating adaptive equalization, color normalization, and Gaussian filtering, coupled with the removal of optic disc and blood vessels to enhance image quality. Subsequent image segmentation techniques are applied to extract relevant markers and features pivotal for the classification process. Classification is accomplished through an ensemble of classifiers, utilizing a "weight voting" technique that exhibits varied performance across distinct severity levels. The system demonstrates remarkable accuracy in distinguishing different stages of diabetic retinopathy, highlighting its efficacy as an advanced diagnostic tool. The success of the robust and reliable classification system can be attributed to the effectiveness of the employed image preprocessing techniques, collectively offering clinicians a valuable and sophisticated tool for accurate assessment and timely intervention in diabetic retinopathy severity detection. This research significantly contributes to the field of medical image analysis, providing a nuanced and effective approach to diabetic retinopathy severity detection, with the potential to enhance patient outcomes and inform clinical decision-making.

Dolly Das, Saroj Kumar Biswas & Sivaji Bandyopadhyay[3]. This algorithm-based diagnostic application focuses on the detection of diabetic retinopathy by analyzing ophthalmoscopic images, specifically targeting the identification of the optic disc and associated lesions indicative of the condition. The primary objective is to enhance early detection and subsequently reduce visual impairment related to diabetes. Leveraging retinal images obtained through the Optomed Smartscope(r) PRO ophthalmoscope, the application employs morphological algorithms tailored for identifying both the optic disc and lesions characteristic of diabetic retinopathy (DR). The integration of advanced technologies significantly improves the efficiency of diabetic retinopathy diagnosis. The application employs automatic methods based on neural networks and image analysis algorithms, contributing to the precision and reliability of the diagnostic process. Notably, the user-friendly graphical interface enhances accessibility, allowing for efficient image analysis, diagnosis, and monitoring. This innovative approach to diabetic retinopathy detection holds great promise for early intervention and improved patient outcomes, underscoring the potential impact of technology-driven solutions in tackling visual impairment associated with diabetes.

M. Bader Alazzam, F. Alassery[4]. In this prospective study conducted at the Sindh Institute of Ophthalmology & Visual Sciences, the focus was on diabetic retinopathy detection utilizing a modified Convolutional Neural Network (CNN) with fundus images. The deep learning model, constructed with convolutional and pooling layers, was meticulously trained and validated on a private dataset. Subsequently, the model underwent real-time testing at the institute. Impressively, the model achieved a high accuracy of 93.72%, showcasing its proficiency in classifying test images into DR-Positive and DR-Negative categories. The sensitivity and specificity of the model were noteworthy, measuring at 97.30% and 92.90%, respectively. The dataset used in the study was thoughtfully divided into training, validation, and testing subsets, ensuring a comprehensive evaluation of the model's performance. The successful classification of images into relevant categories was confirmed through the expert evaluation of clinical professionals, validating the model's efficacy in diabetic retinopathy detection. It is crucial to highlight that the model's training and validation were conducted on a private dataset, contributing to a tailored and focused approach that aligns with the specific nuances of the target population. The real-time testing at SIOVS not only reaffirmed the model's robustness but also emphasized its potential for immediate integration into clinical settings. This research stands as a significant contribution to the field of medical image analysis, particularly in ophthalmology, offering a reliable and efficient tool for the early detection of diabetic retinopathy and, consequently, contributing to enhanced patient care and outcomes.

Naama Hammel, MD . Yun Liu, PhD [5]. This study focuses on the crucial task of early detection of diabetic retinopathy, a major contributor to visual impairment and blindness, especially in low-income countries. Recognizing the significance of timely intervention, the research combines deep learning techniques with ultra-wide-field fundus photography, presenting a promising approach to enhance detection capabilities. Acknowledging the effectiveness of the early treatment diabetic retinopathy study 7-standard field (ETDRS 7SF) images, the study emphasizes the adoption of ultra-wide-field (UWF) fundus photography, covering a remarkable 82% of the retinal surface. The methodology involves the automatic segmentation of ETDRS 7SF for diabetic retinopathy detection, utilizing a ResNet-34 deep learning architecture as a classifier. Notably, the ResNet architecture is highlighted for its ease of optimization and accuracy gain in deep networks. The study further incorporates optic disc and macula detection, strategically excluding undesirable regions and focusing on the critical areas for diagnosis. The utilization of UWF fundus photography is identified as a pivotal aspect, capturing a wide area for diabetic retinopathy detection and providing a comprehensive perspective for improved diagnosis and management. Additionally, the segmentation of ETDRS 7SF images from UWF photography is recognized as a time-saving and efficient approach, streamlining efforts in the diagnostic process. This research not only advances the capabilities of diabetic retinopathy detection through cutting-edge technologies but also underscores the potential impact on clinical efficiency and resource optimization. By combining the advantages of deep learning, UWF fundus photography, and targeted image segmentation, the study contributes to the evolving

landscape of medical image analysis, offering a robust and efficient tool for early diagnosis and management of diabetic retinopathy. The findings hold particular significance for regions with limited resources, where early detection and streamlined diagnostic processes are imperative for mitigating the impact of diabetic retinopathy on visual health.

CHAPTER 3

Proposed System

3.1 Problem Statement

Diabetic retinopathy (DR) poses a significant threat to vision by affecting the delicate blood vessels in the retina, and its asymptomatic early stages make it a leading cause of blindness globally. Given the rising incidence of diabetes worldwide, the demand for precise and swift diagnostic techniques for DR is paramount.

3.2 Algorithms Used

3.2.1 EfficientNetV2

EfficientNetV2 is an extension and refinement of the EfficientNet architecture, which was introduced by Google in 2019. The main goal of EfficientNetV2 is to further improve model efficiency without sacrificing performance. This is achieved through several key advancements, including the introduction of FusedMB convolutions. EfficientNet was groundbreaking because it introduced a scalable model architecture that achieved state-of-the-art performance with significantly fewer parameters and FLOPs (Floating Point Operations) compared to other architectures like ResNet or Inception. It achieved this through a combination of compound scaling, where the model depth, width, and resolution are scaled together, and efficient building blocks like MobileNetV2's inverted residuals.

EfficientNetV2 Improvements:

- **Compound Scaling Refinement**: While EfficientNet introduced compound scaling, EfficientNetV2 refines this approach by incorporating more precise scaling coefficients, resulting in even better performance.
- **FusedMB Convolutions**: One of the notable advancements in EfficientNetV2 is the introduction of FusedMB convolutions. FusedMB convolutions combine multiple operations typically found in convolutional layers.

This reduces the computational overhead associated with separate operations, resulting in faster inference and training speeds without sacrificing accuracy.

- **Stem Optimization:** EfficientNetV2 optimizes the initial stem of the network to reduce computational cost and improve performance.
- **Improved Regularization:** EfficientNetV2 incorporates advanced regularization techniques to enhance generalization and prevent overfitting, such as Stochastic Depth and DropPath.
- **Better Scaling Coefficients:** The scaling coefficients used in EfficientNetV2 are refined based on empirical observations and extensive experiments, resulting in improved performance across a wide range of tasks and resource constraints.

Fig 3.1 displays the convolutional architecture of the EfficientNetV2 family as discussed above. This specific image is of the EfficientNetV2S model

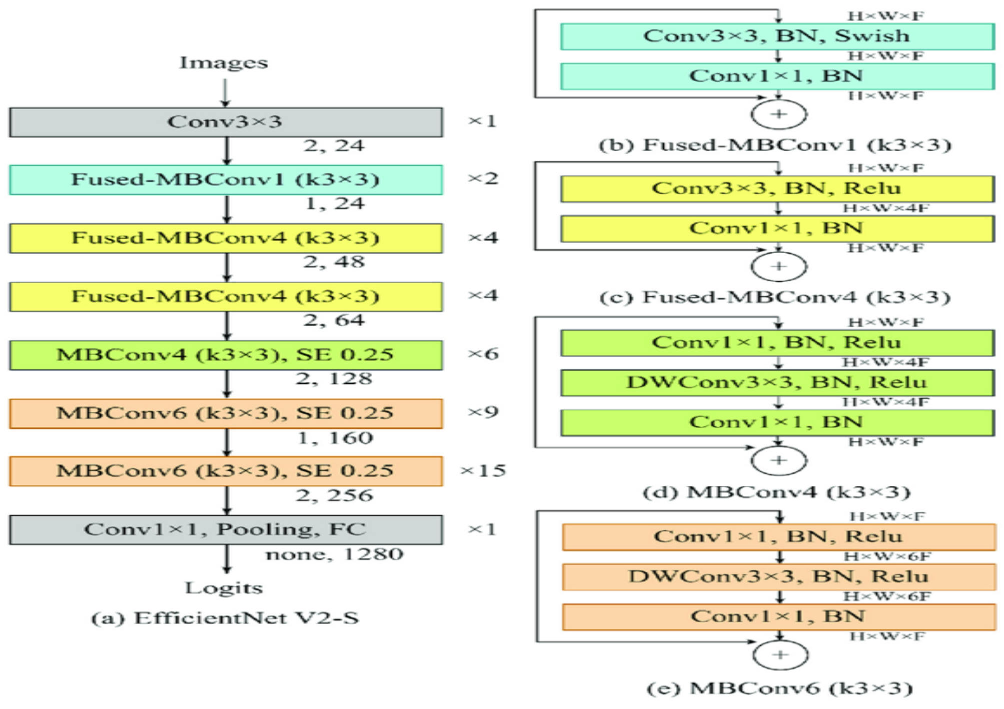

Fig 3.1. EfficientNetV2 Family Algorithm displaying the Convolutional Layers present inside

3.3 System Architecture

In the proposed system, we aim to leverage the strengths of three different variants of the EfficientNet-V2 model: effnetv2_b0, effnetv2_b1, and effnetv2_s. Each of these variants offers unique characteristics and performance trade-offs, and by combining them in a stacking ensemble, we seek to harness their collective power for improved classification accuracy.EfficientNet-V2_b0, EfficientNet-V2_b1, and EfficientNet-V2_s represent different scales or sizes within the EfficientNet-V2 model family.

These variants vary in terms of depth, width, and resolution, with b0 being the smallest and s being the largest. By incorporating models of varying scales, we can capture a broader range of features and patterns in the input data, thereby enhancing the overall robustness and generalization capability of the ensemble.

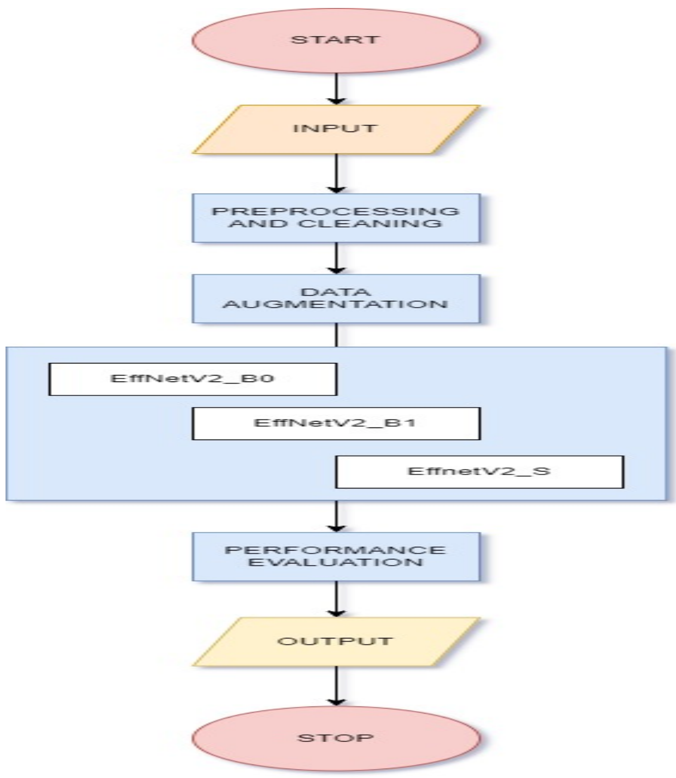

Fig 3.2. Proposed System Diagram

In the stacking ensemble approach, each individual model (effnetv2_b0, effnetv2_b1, and effnetv2_s) will independently process the input images and generate predictions for the target classes. These predictions will then be combined at a higher level, typically through a meta-learner or a simple averaging mechanism, to produce the final ensemble prediction. The stacking ensemble allows us to exploit the complementary strengths of each model variant. While effnetv2_b0 might excel at capturing fine-grained details due to its smaller size, effnetv2_s could better handle capturing broader contextual information thanks to its larger scale. By combining these models, we can create a more robust and accurate classifier that is capable of handling a wide range of input data scenarios.

Overall, the proposed system of employing effnetv2_b0, effnetv2_b1, and effnetv2_s in a stacking ensemble offers a powerful solution for image classification tasks, leveraging the strengths of multiple model variants to achieve superior performance and robustness.

3.4 Evaluation Metrics

This study employs different evaluation metrics to observe and analyze the performance of the model. Namely, they are Precision, Recall, F1 Score and Accuracy. The formula relating to the same has been given below in equations 1, 2 and 3.

$$\text{Precision} = \frac{\text{True Positives}}{\text{True Positives + False Positives}} \quad - \quad (1)$$

$$\text{Recall} = \frac{\text{True Positives}}{\text{True Positives + False Negatives}} \quad - \quad (2)$$

$$\text{F1 Score} = \frac{2 * \text{True Positives}}{\text{True Positives} + 0.5(\text{False Positives + False Negatives})} \quad - \quad (3)$$

Additionally, confusion matrices are used in order to further visualize the same. A confusion matrix is a performance measurement technique used in the field of machine learning and classification tasks to evaluate the accuracy of a classification model. It's a square matrix that tabulates the actual class labels against the predicted class labels. Each row of the matrix represents the instances in an actual class, while each column represents the instances in a predicted class. The diagonal elements of the matrix represent the instances that were correctly classified, while the off-diagonal elements represent misclassification

Chapter 4

Implementation

4.1 Software Requirements

4.1.1 Python3:

Python 3 is the latest major version of the Python programming language, emphasizing code readability and simplicity. In the context of machine learning, Python 3 is the preferred version due to its extensive ecosystem of libraries and frameworks, such as NumPy, Pandas, Scikit-learn, TensorFlow, and PyTorch. It provides a versatile and user-friendly environment for developing, training, and deploying machine learning models. Python 3's syntax and features contribute to efficient and expressive code, making it a dominant language for tasks ranging from data preprocessing to complex deep learning applications in the rapidly evolving field of machine learning

4.1.2 TensorFlow:

TensorFlow is an open-source machine learning framework developed by Google. It facilitates the creation and training of deep learning models through a flexible, high-level API. TensorFlow allows developers to build and deploy machine learning applications, leveraging computational graphs for efficient processing on CPUs or GPUs. With extensive support for neural networks and deep learning, TensorFlow has become a cornerstone in the development of various artificial intelligence applications, including image and speech recognition, natural language processing, and more.

4.2 Hardware Requirements

The following set of experiments have been conducted A100 GPU provided by the Google Colab with 40GB of virtual random access memory. Additionally it has 6912 shading units, 432 texture units and 160 ROPs. The following are the specifications.

- FP64: 9.7 TFLOPS
- FP64 Tesnsor Core: 19.5 TFLOPS
- FP32: 19.5 TFLOPS
- TF32: 156 TFLOPS
- GPU Memory: 80 GB HBM2e
- GPU Memory Bandwidth : 1,935 GB/s
- Max Thermal Design Power(TDP): 300 W

4.3 Dataset and Preprocessing

The data had been collected from the EyePacs foundation, a free platform for retinopathy screening. The data under study was a sample of 8392 images of a total of 35,321 images from the original dataset with labelled images of left and right eye fundus corresponding to 5 target classes that are No DR, Mild DR, Moderate DR, Severe DR and Proliferative DR. Preprocessing steps are taken to clean the initial data and load the images into the dataset, removing corrupt image files and null values. However on further exploration, it can be noted from Fig 4.1 that the dataset is highly imbalanced with more than 6000 images of the NO DR class.

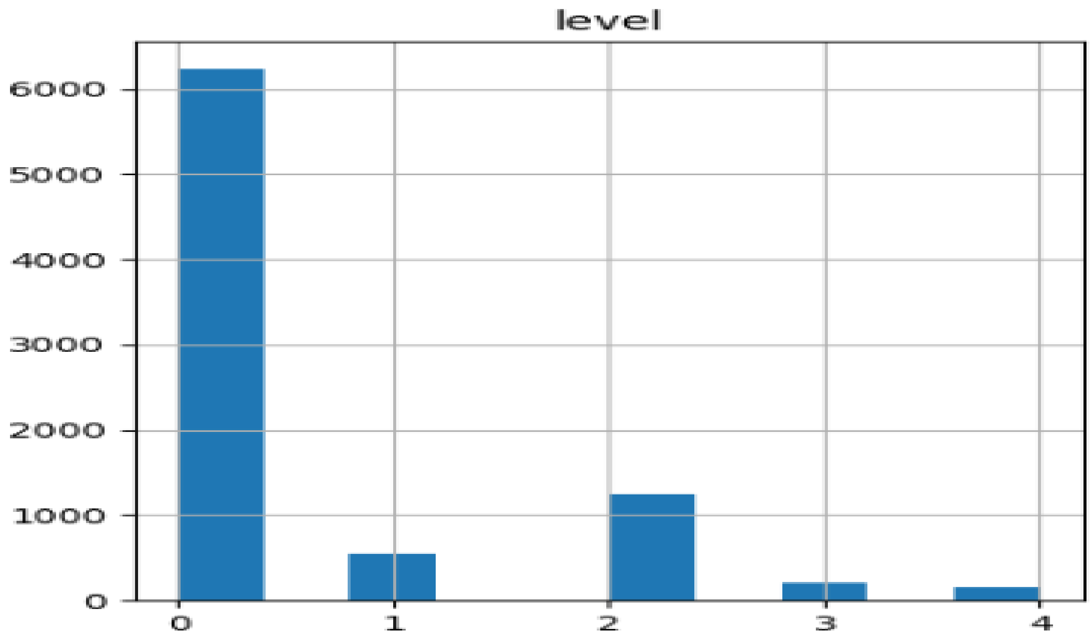

Fig 4.1: Count of Data based on Classes

Thus balancing methods are approached and applied, namely random under sampling and random oversampling in order to balance the counts of each class. Random Oversampling involves filtering through the dataset for random instances of the class and then duplicating the same, this is usually done to the minority class in order to balance out the classes. In contrast, Random Undersampling is usually done on the majority class, where the data of the majority class is reduced in order to better balance out the classes. This typically involves loss of data. The results can be clearly viewed in Fig 4.2 where all the class counts are balanced with one another.

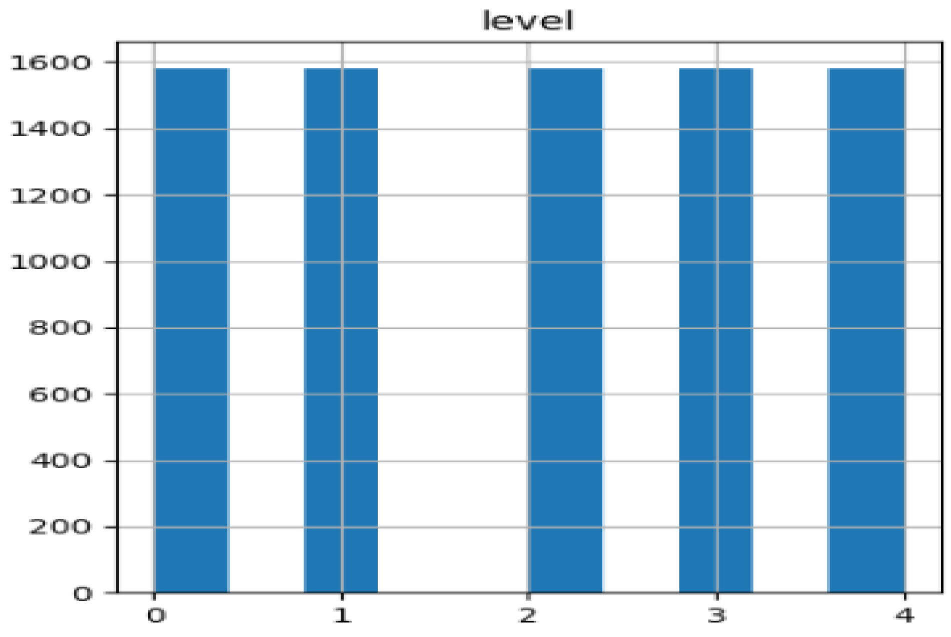

Fig 4.2 Balanced Dataset after Random Undersampling and Random

Additionally different augmentation methods have been applied to the data such as random flip, random rotation, random contrast, random hue, and random brightness in order to augment the training data for better learning representations. This can be viewed in Fig 4.3

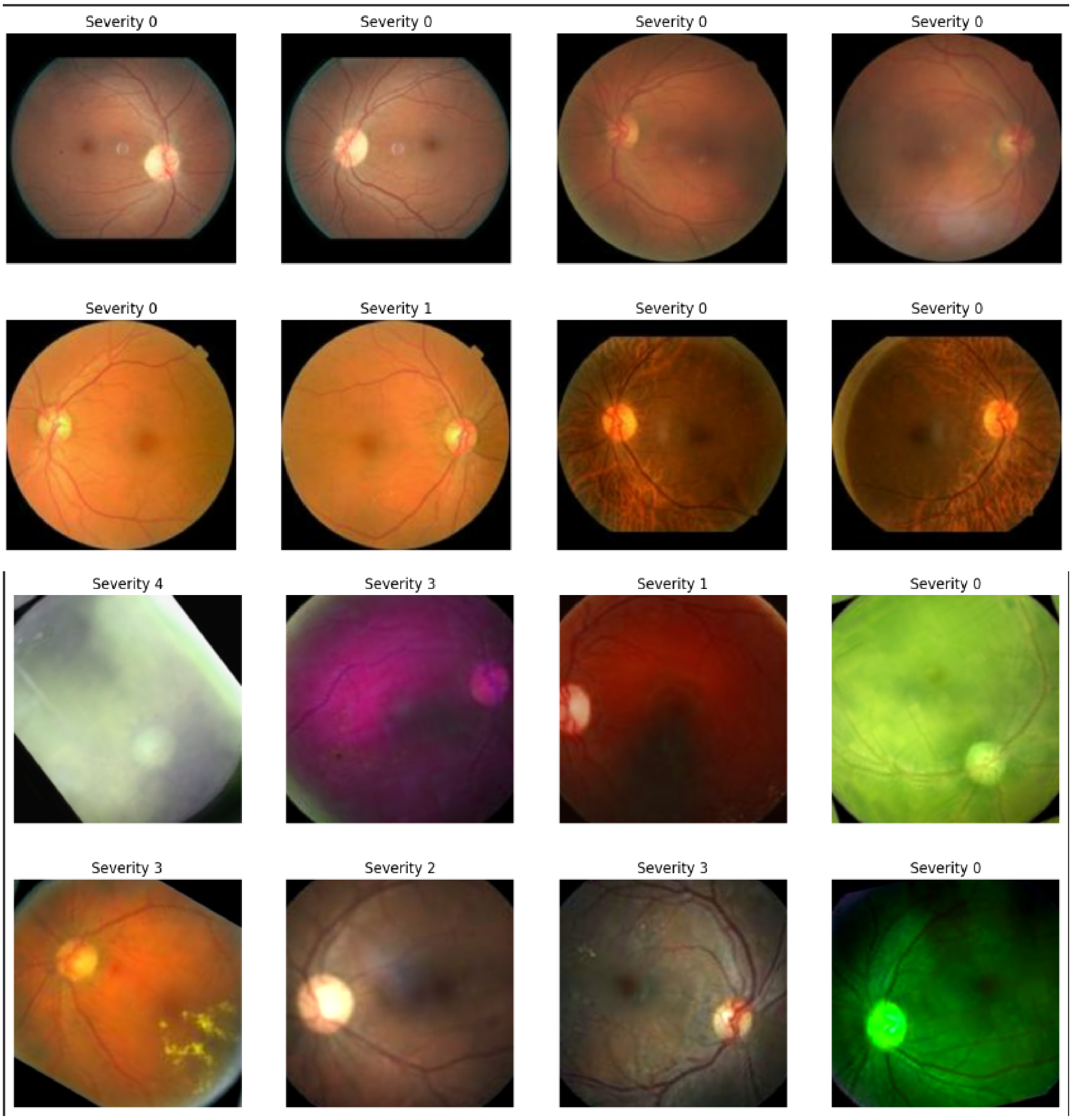

Figure 4.3 Training Images before and After Augmentation

Chapter - 5

Results and Discussion

5.1 Prediction Results of EfficientNetV2 S

From Fig 5.1, It can be noted from the confusion matrix on EfficientNetV2S that the model is unable to classify that well. We notice a great many false negatives when looking at the first class which can imply that the feature representation is not learnt well. This can be due to the fact that the data is highly imbalanced for the first class, i.e Class 0 (No DR) as compared to the rest of the other classes. Table 5.1 displays the scores and metrics of the same.

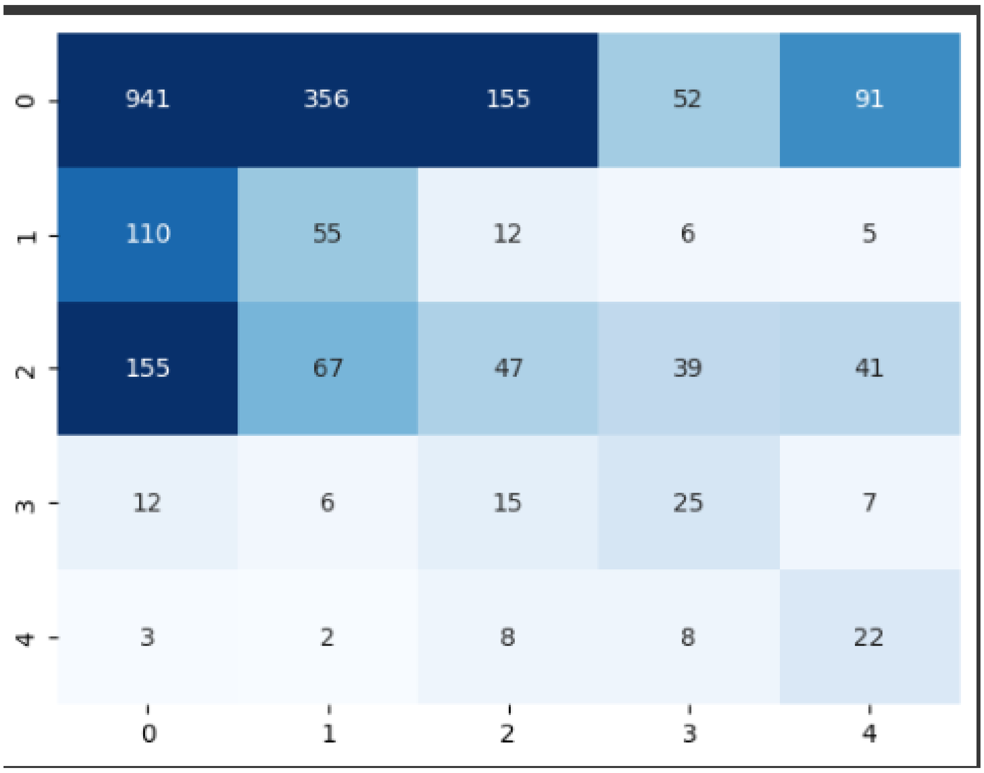

Fig 5.1 : Confusion Matrix of EfficientNetV2S

Table 5.1 Evaluation of EfficientNetV2 S

Class	Precision	Recall	F1 Score	Support
No DR	**0.77**	**0.59**	**0.6**	**1595**
Mild DR	**0.11**	**0.29**	**0.16**	**188**
Moderate DR	**0.20**	**0.13**	**0.16**	**349**
Severe DR	**0.19**	**0.38**	**0.26**	**65**
Proliferative DR	**0.13**	**0.51**	**0.21**	**43**

5.2 Prediction Results of EfficientNetV2B0

From Fig 6.2, It can be noted from the confusion matrix on EfficientNetV2B0 that the model is unable to classify properly. We notice a great many false positives when looking at the first and third class which can imply that the feature representation is not learnt well as compared to EfficientNetV2S which was able to learn somewhat better.

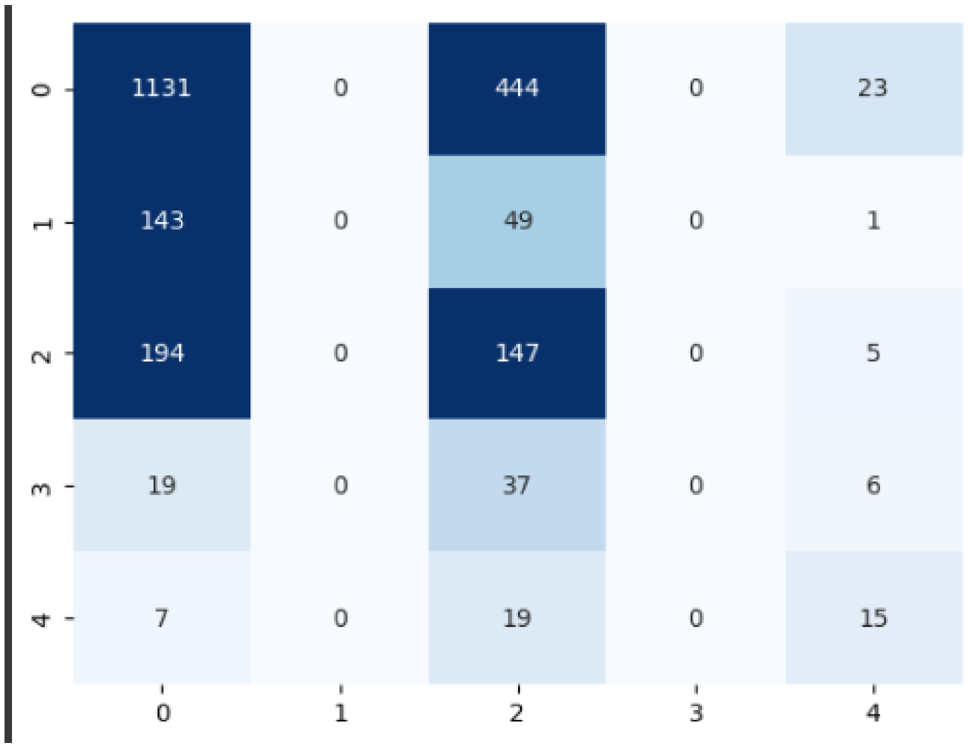

Fig 5.2 Confusion Matrix of EfficientNetV2_B0

Table 5.2 Evaluation of EfficientNetV2 B0

Class	Precision	Recall	F1 Score	Support
No DR	**0.76**	**0.71**	**0.73**	**1598**
Mild DR	**0.00**	**0.00**	**0.00**	**193**
Moderate DR	**0.21**	**0.42**	**0.28**	**346**
Severe DR	**0.00**	**0.00**	**0.00**	**62**
Proliferative DR	**0.30**	**0.37**	**0.33**	**41**

5.3 Prediction Results of EfficientNetV2B1

From Fig 5.3, It can be noted from the confusion matrix on EfficientNetV2B1 that the model is not that different from the B0 model. We notice a great many false positives when looking at the first and third class which can imply that the feature representation is not learnt well as compared to EfficientNetV2S which was able to learn somewhat better.

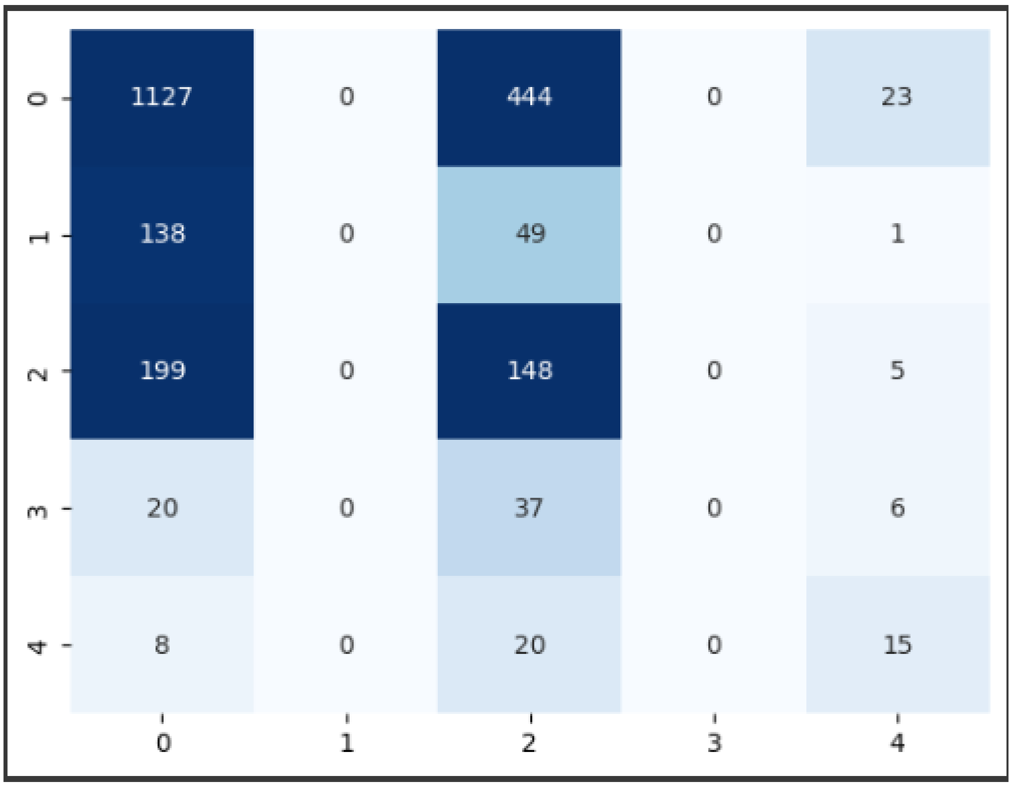

Fig 5.3. Confusion Matrix of EfficientNetV2 B1

Table 5.3. Evaluation of EfficientNetB1

Class	Precision	Recall	F1 Score	Support
No DR	**0.76**	**0.71**	**0.73**	**1594**
Mild DR	**0.00**	**0.00**	**0.00**	**188**
Moderate DR	**0.21**	**0.42**	**0.28**	**352**
Severe DR	**0.00**	**0.00**	**0.00**	**63**
Proliferative DR	**0.30**	**0.35**	**0.32**	**43**

5.1 Prediction Results of Proposed Ensemble Model

From Fig 5.4, we can see that the proposed ensemble model which was a stacking model of predictions based on that of the EfficientNetV2B0, EfficientNetV2B1, and EfficientNetV2S performs even more poorly as compared to the other models as the features were not well learnt from the previous models given from the data. This can also be indicative of the fact that the model was able to learn the features of No DR properly but unable to accurately identify the other classes.

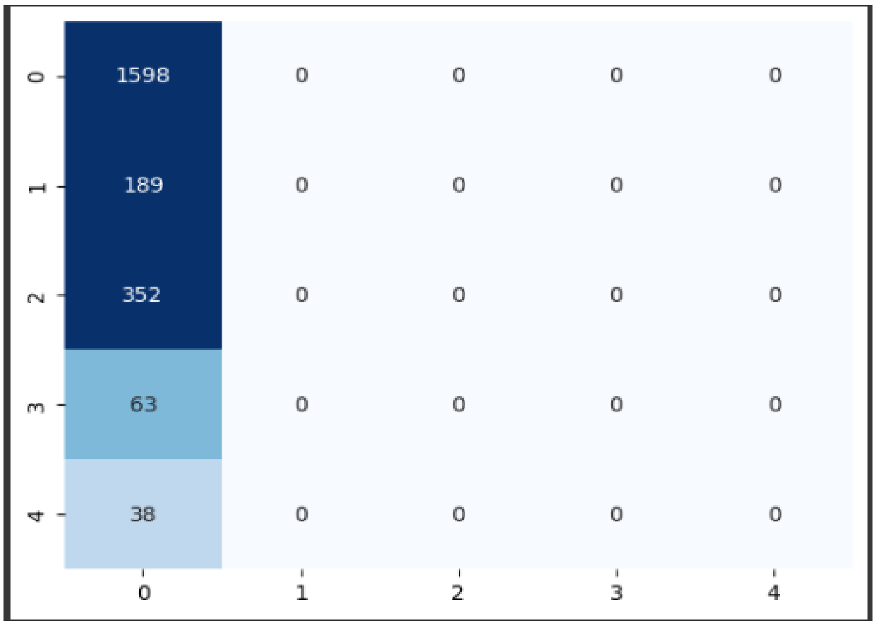

Fig 5.4 Confusion Matrix of Proposed Ensemble Stacking Model

Table 5.4. Evaluation of Proposed Ensemble Stacking Model

Class	**Precision**	**Recall**	**F1 Score**	**Support**
No DR	**0.71**	**1.00**	**0.83**	**1598**
Mild DR	**0**	**0**	**0**	**189**
Moderate DR	**0**	**0**	**0**	**352**
Severe DR	**0**	**0**	**0**	**63**
Proliferative DR	**0**	**0**	**0**	**38**

CONCLUSIONS AND FUTURE SCOPE:

From the above results, we are able to conclude that, this study has advanced the identification of diabetic retinopathy by utilizing ensemble machine learning models to their full potential. In particular, we examined the efficacy of combining a stacking ensemble technique with a state-of-the-art deep learning architecture, EfficientNetV2_S.

The study had found that the data provided from the EyePacs Foundation to be highly imbalanced with one of the classes being an absolute majority for feature representation with 6012 images from the 8400 image sample being classified as NO DR. Conversely the EfficientNetV2 family of models were not able to learn the feature representations within it as much as possible, with only being able to learn the majority feature representation.

Although, a slight improvement on the score had been noticed from different iterations where the data had been preprocessed and heavily augmented. This involves the adjustment of Saturation, Contrast, Brightness, Hue and other augmentation techniques. The model showed a significant improvement in performance when using Random Undersampling and Oversampling methods but shows indication of being overfitting as the same examples are being seen by the model again and again, which implies that the model has a tendency to over fit on test data.

The main goal of this experiment was to find a way to minimize the amount of model parameters used to classify the fundus images in order to lessen the burden on the edge device. While the goal has not been achieved, the future of this research has a plethora of interesting prospects ahead of it. Improving the methods on which the features can be learnt is one of it as we can see that the general EfficientNetV2 model is not able to able to learn the features as much as expected. An addition of a different Pooling strategy can be considered aside from the Global Average Pooling strategy that was applied in this particular study. Fractional Max Pooling is a course of interest that we look forward to working on in the future.

Additionally, different techniques for handling dataset imbalance can be explored. Combining the fundus images of both left and right eyes to better learn the feature representation before feeding the data to the actual model of predictions can also be considered.

REFERENCES

[1] Diabetic retinopathy is the leading cause of visual loss

[2] Diabetic Retinopathy: Present andPast
https://www.sciencedirect.com/science/article/pii/S1877050918308068

[3] Detection of Diabetic Retinopathy using Convolutional Neural Networks
https://link.springer.com/article/10.1007/s11042-022-14165-4

[4] Identification of Diabetic Retinopathy through Machine Learning
https://www.hindawi.com/journals/misy/2023/9832745/

[5]Predicting the risk of developing diabetic retinopathy using deep learning
https://www.thelancet.com/journals/landig/article/PIIS2589-7500(20)30250-8/fulltext

[6] Taylor R, Batey D. Handbook of retinal screening in diabetes:diagnosis and management. second ed. John Wiley & Sons, Ltd Wiley-Blackwell; 2012.

[7] Liu H, Wang L, Nan Y, et al. SDFN: Segmentation-based deep fusion net work for thoracic disease classification in chest X-ray images. Computer ized Med Imaging and Graphics 2019; 75:66–73. doi:10.1016/j. compmedimag.2019.05.005.

[8] Ananth, C. V. and Kleinbaum, D. G. (1997). Regression models for ordinal responses: a review of methods and applications. *International Journal of Epidemiology*, 26(6):1323–1333.

[9] G. G. Gardner, D. Keating, T. H. Williamson, and A. T. Elliott, 'Automatic detection of diabetic
retinopathy using an artificial neural network: a screening tool.', *British Journal of*
Ophthalmology, vol. 80, no. 11, pp. 940–944, Nov. 1996, doi: 10.1136/BJO.80.11.940.

[10] Banerjee S, Kayal D (2016) Detection of hard exudates using mean shift and normalized cut method. Biocy
bern Biomed Eng 36(4):679–685

APPENDIX
SOURCE CODE

```
retina_df = pd.read_json("/content/drive/MyDrive/Project-Work/retina_df.json")
retina_df['level_cat'] = retina_df['level'].map(lambda x: to_categorical(x,
1+retina_df['level'].max()))
retina_df

from sklearn.model_selection import train_test_split
rr_df = retina_df[['PatientId', 'level']].drop_duplicates()

#Splitting based on patient id
train_ids, valid_ids = train_test_split(rr_df['PatientId'],
                                 test_size = 0.25,
                                 random_state = 42,
                                 stratify = rr_df['level'])

# If patient id is present in retina_df
raw_train_df = retina_df[retina_df['PatientId'].isin(train_ids)]
valid_df = retina_df[retina_df['PatientId'].isin(valid_ids)]
print('train', raw_train_df.shape[0], 'validation', valid_df.shape[0])

retina_df[['level', 'eye']].hist(figsize = (10, 5))

retina_df.dtypes

from sklearn.utils import resample

maj_class = raw_train_df[raw_train_df['level'] == 0]

min_class_1 = raw_train_df[raw_train_df['level'] == 1]
min_class_2 = raw_train_df[raw_train_df['level'] == 2]
min_class_3 = raw_train_df[raw_train_df['level'] == 3]
min_class_4 = raw_train_df[raw_train_df['level'] == 4]

n_samples_per_class=len(maj_class)//3
```

```
#Downsampling by half the amount
majority_downsampled = resample(maj_class,
                                replace = False,
                                n_samples = n_samples_per_class,
                                random_state = 42)

downsampled_df = pd.concat([majority_downsampled, min_class_1, min_class_2, min_class_3,
min_class_4])
downsampled_df = downsampled_df.sample(frac = 1, random_state = 42)
print('Old Data Size:', raw_train_df.shape[0], 'New Data Size:', downsampled_df.shape[0])
downsampled_df[['level', 'eye']].hist(figsize = (10, 5))

maj_class = downsampled_df[downsampled_df['level'] == 0]

min_class_1 = downsampled_df[downsampled_df['level'] == 1]
min_class_2 = downsampled_df[downsampled_df['level'] == 2]
min_class_3 = downsampled_df[downsampled_df['level'] == 3]
min_class_4 = downsampled_df[downsampled_df['level'] == 4]

n_samples_per_class = len(maj_class)

oversampled_min_class_1 = resample(min_class_1,
                                   replace=True,
                                   n_samples=n_samples_per_class,
                                   random_state=42)

oversampled_min_class_2 = resample(min_class_2,
                                   replace=True,
                                   n_samples=n_samples_per_class,
                                   random_state=42)

oversampled_min_class_3 = resample(min_class_3,
                                   replace=True,
                                   n_samples=n_samples_per_class,
                                   random_state=42)

oversampled_min_class_4 = resample(min_class_4,
                                   replace=True,
                                   n_samples=n_samples_per_class,
                                   random_state=42)

oversampled_df = pd.concat([majority_downsampled, oversampled_min_class_1,
oversampled_min_class_2, oversampled_min_class_3, oversampled_min_class_4])
oversampled_df = oversampled_df.sample(frac = 1, random_state = 42)
print('Old Data Size:', raw_train_df.shape[0], 'New Data Size:', oversampled_df.shape[0])
oversampled_df[['level', 'eye']].hist(figsize = (10, 5))
|
import os
from PIL import Image
import numpy as np

# Define a function to load and preprocess images
def load_image(file_path, image_size):
    try:
        # Open and preprocess the image
        img = Image.open(file_path)
        img = img.resize(image_size)
        img_array = np.array(img) / 255.0  # Normalize pixel values
        return img_array, None
    except Exception as e:
        return None, file_path  # Return file path of corrupted image

# Define parent directory containing images
parent_directory = '/content/drive/MyDrive/Project-Work'

# Define image size for resizing
image_size = (224, 224)

# Iterate through files in the parent directory
corrupted_files = []
for filename in os.listdir(parent_directory):
    if filename.endswith('.jpeg'):
        file_path = os.path.join(parent_directory, filename)
        img_array, corrupted_path = load_image(file_path, image_size)
        if img_array is not None:
            # If theu image was loaded successfully, proceed with training
            # Here you would feed the image to your model and pdate weights
            pass
        else:
            # If the image could not be loaded, add its file path to the list of corrupted files
            corrupted_files.append(corrupted_path)

# Print list of corrupted files
print("Corrupted files:")
for file_path in corrupted_files:
    print(file_path)

import tensorflow as tf
from keras import backend as K
import numpy as np
from  tensorflow.keras.applications.efficientnet_v2 import preprocess_input
IMG_SIZE = (150, 150)
def tf_image_loader(out_size,
                      horizontal_flip = True,
                      vertical_flip = False,
                     random_brightness = True,
                     random_contrast = True,
                    random_saturation = True,
                    random_hue = True,
                      color_mode = 'rgb',
                       on_batch = False,
                    preprocess = preprocess_input):

    def _func(X):
      '''
      Performs Image Augmentation- Augmentation Considerations :
        Horizontal Flip, Vertical Flip, Random_Brightness, Random_Saturation, Random_Hue
      Args
      X - Train Images
      '''
      with tf.name_scope('image_augmentation'):
            with tf.name_scope('input'):
                X = tf.image.decode_jpeg(tf.io.read_file(X), channels = 3 if color_mode == 'rgb' else
0)
                X = tf.image.resize(X, out_size)
            with tf.name_scope('augmentation'):
                if horizontal_flip:
```

```
            X = tf.image.random_flip_left_right(X)
        if vertical_flip:
            X = tf.image.random_flip_up_down(X)
        if random_brightness:
            X = tf.image.random_brightness(X, max_delta = 0.5)
        if random_saturation:
            X = tf.image.random_saturation(X, lower = 0.75, upper = 1.5)
        if random_hue:
            X = tf.image.random_hue(X, max_delta = 0.25)
        if random_contrast:
            X = tf.image.random_contrast(X, lower = 0.75, upper = 1.5)
        return preprocess(X)
    if on_batch:
        # we are meant to use it on a batch
        def _batch_func(X, y):
            return tf.map_fn(_func, X, fn_output_signature=tf.float32), y
        return _batch_func
    else:
        # we apply it to everything
        def _all_func(X, y):
            return _func(X), y
        return _all_func

def tf_augmentor(out_size,
                intermediate_trans = 'crop',
                intermediate_size = (640, 640),
                batch_size = 32,
                  horizontal_flip = True,
                 vertical_flip = False,
                random_brightness = False,
                random_contrast = True,
                random_saturation = True,
                   random_hue = True,
                preprocess = preprocess_input,
                 color_mode = 'rgb',
                  crop_probability = 0.5,
                  rotation_range = 40):
```

```
load_ops = tf_image_loader(out_size = intermediate_size,
                           horizontal_flip=horizontal_flip,
                           vertical_flip=vertical_flip,
                           random_brightness = random_brightness,
                           random_contrast = random_contrast,
                           random_saturation = random_saturation,
                           random_hue = random_hue,
                           color_mode = color_mode,
                           on_batch=False)
random_rotation_layer = tf.keras.layers.RandomRotation(rotation_range/180*np.pi)
crop_height, crop_width = intermediate_size
random_crop_layer = tf.keras.layers.RandomCrop(crop_height, crop_width)

def batch_ops(X, y, training = True):
  '''
  Performs rotation, cropping and resizing operations on the batch

  Args:
  X = Train Data - Images (640 X 640)
  y = Train Labels
  '''
  with tf.name_scope('transformation'):
      if rotation_range > 0:
        X = random_rotation_layer(X)
      if crop_probability>0:
        X = random_crop_layer(X)
      #Apply intermediate transformations based on specific method
      if intermediate_trans == 'scale':
        X = tf.image.resize(X, out_size)
      elif intermediate_trans == 'crop':
        X = tf.image.resize_with_crop_or_pad(X, out_size[0], out_size[1])
      else:
        raise ValueError('Invalid Operation {}'.format(intermediate_trans))

  return X, y

def _create_pipeline(in_ds):
  '''
```

```
      Pipeline for prefetching and batching the data

      Args:
        in_ds - tf.data.Dataset : input dataset
      Returns:
        tf.data.Dataset: Processed pipeline
      '''
      if not isinstance(in_ds, tf.data.Dataset):
          raise ValueError("Input dataset must be a TensorFlow Dataset object")
      batch_ds = in_ds.map(load_ops, num_parallel_calls=tf.data.AUTOTUNE).batch(batch_size)
      return batch_ds.map(batch_ops).prefetch(tf.data.AUTOTUNE)

    return _create_pipeline

def preprocess_labels(label_list):
  return tf.convert_to_tensor(label_list, dtype=tf.float32)

def flow_from_dataframe(idg, in_df, path_col, y_col, shuffle=True, color_mode='rgb',
target_dist = None):
    paths = in_df[path_col].values
    labels = in_df[y_col].apply(preprocess_labels).tolist()

    # Convert string paths to tensors
    paths = tf.convert_to_tensor(paths, dtype=tf.string)
    labels = tf.convert_to_tensor(labels, dtype=tf.float32)

    # Create a TensorFlow Dataset from the paths and labels
    dataset = tf.data.Dataset.from_tensor_slices((paths, labels))
    # Specify the output shapes when creating the dataset
    output_shapes = (tf.TensorShape([]), tf.TensorShape([5]))

    # Shuffle the dataset if specified
    if shuffle:
        dataset = dataset.shuffle(buffer_size=len(paths))

    # Apply the image data generator pipeline
    dataset = idg(dataset)

    return dataset
```

```
batch_size = 32
core_idg = tf_augmentor(out_size = (512,512),
                        color_mode = 'rgb',
                        vertical_flip = True,
                        crop_probability=0.3,# crop doesn't work yet
                        batch_size = batch_size)

valid_idg = tf_augmentor(out_size = (512, 512), color_mode = 'rgb',
                         crop_probability=0.0,
                         intermediate_trans = 'scale',
                         horizontal_flip = False,
                         vertical_flip = False,
                         random_brightness = False,
                         random_contrast = False,
                         random_saturation = False,
                         random_hue = False,
                         rotation_range = 0,
                         batch_size = batch_size)
                        #is_training = False)

train_gen = flow_from_dataframe(core_idg, oversampled_df,
                             path_col = 'path',
                            y_col = 'level_cat')

valid_gen = flow_from_dataframe(valid_idg, valid_df,
                             path_col = 'path',
                            y_col = 'level_cat',
                                shuffle = False,) # we can use much larger batches for evaluation

valid_iter = iter(valid_gen)
v_x, v_y = next(valid_iter)
fig, m_axs = plt.subplots(2, 4, figsize = (16, 8))
for (c_x, c_y, c_ax) in zip(v_x, v_y, m_axs.flatten()):
    c_ax.imshow(np.clip(c_x, 0, 255).astype(np.uint8))
    c_ax.set_title('Severity {}'.format(np.argmax(c_y, -1)))
    c_ax.axis('off')

train_iter = iter(train_gen)
```

```
t_x, t_y = next(train_iter)
fig, m_axs = plt.subplots(2, 4, figsize = (16, 8))
for (c_x, c_y, c_ax) in zip(t_x, t_y, m_axs.flatten()):
    c_ax.imshow(np.clip(c_x, 0, 255).astype(np.uint8))
    c_ax.set_title('Severity {}'.format(np.argmax(c_y, -1)))
    c_ax.axis('off')

import tensorflow as tf
from tensorflow.keras.layers import Input, Dropout, GlobalAveragePooling2D, Dense
from tensorflow.keras.models import Model
from tensorflow.keras.applications import EfficientNetV2S  # Import EfficientNetV2S or other
variant
from tensorflow.keras.layers import BatchNormalization
# from sklearn.utils.class_weight import compute_class_weight
import numpy as np
import tensorflow_addons as tfa


tf.keras.mixed_precision.set_global_policy('mixed_float16')

in_lay = Input(t_x.shape[1:])  # Specify the input shape here

base_pretrained_model = EfficientNetV2S(weights='imagenet', include_top=False,
input_shape=t_x.shape[1:])  # Specify input shape here
base_pretrained_model.trainable = False

pt_features = base_pretrained_model(in_lay)

gap_features = GlobalAveragePooling2D()(pt_features)

gap_dr = Dropout(0.25)(gap_features)
dense_layer = Dense(128, activation='relu')(gap_dr)
dr_steps = Dropout(0.25)(dense_layer)

out_layer = Dense(5, activation='softmax', name = "output")(dr_steps)

# Create the model
retina_model = Model(inputs=in_lay, outputs=out_layer)
```

```
#Let's change to MSE Loss for Getting
loss_fn = tf.keras.losses.CategoricalCrossentropy(from_logits=False)

# def top_2_accuracy(in_gt, in_pred):
#     return top_k_categorical_accuracy(in_gt, in_pred, k=2)

# Compile the model
retina_model.compile(optimizer='adam',
                     loss= loss_fn,
                     metrics=['categorical_accuracy', tfa.metrics.CohenKappa(num_classes=5)])

# Print model summary
retina_model.summary()

from keras.callbacks import ModelCheckpoint, LearningRateScheduler, EarlyStopping,
ReduceLROnPlateau

weight_path="{}_weights.best.hdf5".format('retina')

checkpoint = ModelCheckpoint(weight_path, monitor='val_loss', verbose=1,
                             save_best_only=True, mode='min', save_weights_only = True)

# sampler = SampleBalancingCallback(train_gen, batch_size = batch_size)

reduceLROnPlat = ReduceLROnPlateau(monitor='val_loss', factor=0.8, patience=8, verbose=1,
mode='auto', cooldown=5, min_lr=0.00001)
early = EarlyStopping(monitor="val_loss",
                     mode="min",
                     patience=6) # probably needs to be more patient
callbacks_list = [checkpoint, early, reduceLROnPlat]
%%time
history_1 = retina_model.fit(train_gen,
                         steps_per_epoch = oversampled_df.shape[0]//batch_size,
                         validation_data = valid_gen,
                         validation_steps = valid_df.shape[0]//batch_size,
                           epochs = 10,
                           callbacks = callbacks_list,
                          workers = 0, # tf-generators are not thread-safe
```

```
                             use_multiprocessing=False,
                             max_queue_size = 0,
                            verbose = 1
                            ).history

import matplotlib.pyplot as plt

# Plot training & validation loss values
plt.plot(history_1['loss'])

plt.plot(history_1['val_loss'])
plt.title('Model loss')
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.legend(['Train', 'Validation'], loc='upper left')
plt.show()

from tqdm import tqdm_notebook
# fresh valid gen
valid_gen = flow_from_dataframe(valid_idg, valid_df,
                             path_col = 'path',
                            y_col = 'level_cat')
vbatch_count = (valid_df.shape[0]//batch_size-1)

out_size = vbatch_count*batch_size

test_X = np.zeros((out_size,)+t_x.shape[1:], dtype = np.float32)
test_Y = np.zeros((out_size,)+t_y.shape[1:], dtype = np.float32)

for i, (c_x, c_y) in zip(tqdm_notebook(range(vbatch_count)),
                         valid_gen):
    j = i*batch_size
    test_X[j:(j+c_x.shape[0])] = c_x
    test_Y[j:(j+c_x.shape[0])] = c_y

# Retrieve saved best
retina_model.load_weights('/content/retina_weights.best.hdf5')

from sklearn.metrics import accuracy_score, classification_report
```

```
pred_y = retina_model.predict(test_X, batch_size = 32, verbose = True)

pred_Y_cat = np.argmax(pred_y, -1)
test_Y_cat = np.argmax(test_Y, -1)
print('Accuracy on Test Data: %2.2f%%' % (accuracy_score(test_Y_cat, pred_Y_cat)))
print(classification_report(test_Y_cat, pred_Y_cat))

import seaborn as sns
from sklearn.metrics import confusion_matrix
sns.heatmap(confusion_matrix(test_Y_cat, pred_Y_cat),
            .
            annot=True, fmt="d", cbar = False, cmap = plt.cm.Blues, vmax = test_X.shape[0]//16)

from sklearn.metrics import roc_curve, roc_auc_score
sick_vec = test_Y_cat>0
sick_score = np.sum(pred_y[:,1:],1)
fpr, tpr, _ = roc_curve(sick_vec, sick_score)
fig, ax1 = plt.subplots(1,1, figsize = (6, 6), dpi = 150)
ax1.plot(fpr, tpr, 'b.-', label = 'Model Prediction (AUC: %2.2f)' % roc_auc_score(sick_vec,
sick_score))
ax1.plot(fpr, fpr, 'g-', label = 'Random Guessing')
ax1.legend()
ax1.set_xlabel('False Positive Rate')
ax1.set_ylabel('True Positive Rate');

import tensorflow as tf
from tensorflow.keras.layers import Input, Dropout, GlobalAveragePooling2D, Dense
from tensorflow.keras.models import Model
from tensorflow.keras.applications import EfficientNetV2B0  # Import EfficientNetV2S or other
variant
from tensorflow.keras.layers import BatchNormalization

#Turn on Mixed Precision Training
tf.keras.mixed_precision.set_global_policy('mixed_float16')

# Define input layer
in_lay = Input(t_x.shape[1:])  # Specify the input shape here

# Load pre-trained model
```

```
base_pretrained_model = EfficientNetV2B0(weights='imagenet', include_top=False,
input_shape=t_x.shape[1:])  # Specify input shape here

# Unfreeze layers that are not BatchNormalization layers
base_pretrained_model.trainable = False

# Get the output features from the pre-trained model
pt_features = base_pretrained_model(in_lay)

# Global average pooling
gap_features = GlobalAveragePooling2D()(pt_features)

# Dropout and dense layers
gap_dr = Dropout(0.25)(gap_features)
dense_layer = Dense(128, activation='relu')(gap_dr)
dr_steps = Dropout(0.25)(dense_layer)

# Output layer
out_layer = Dense(5, activation='softmax', name = "output")(dr_steps)

# Create the model
retina_model_finetune_first10 = Model(inputs=in_lay, outputs=out_layer)

#Define topK Accuracy
from keras.metrics import top_k_categorical_accuracy

def top_2_accuracy(in_gt, in_pred):
    return top_k_categorical_accuracy(in_gt, in_pred, k=2)

#Use the focal loss function
import tensorflow as tf

# Define focal loss function
loss_fn = tf.keras.losses.CategoricalCrossentropy()

# Compile the model
retina_model_finetune_first10.compile(optimizer='adam',
                      loss= loss_fn,
                      metrics=['categorical_accuracy', top_2_accuracy])
```

```
# Print model summary
retina_model_finetune_first10.summary()

for layers in base_pretrained_model.layers:
    print(layers, layers.trainable)

from keras.callbacks import ModelCheckpoint, LearningRateScheduler, EarlyStopping,
ReduceLROnPlateau

weight_path="{}_weights.best.hdf5".format('retina')

# sampler = SampleBalancingCallback(train_gen, batch_size = batch_size)
checkpoint = ModelCheckpoint(weight_path, monitor='val_loss', verbose=1,
                             save_best_only=True, mode='min', save_weights_only = True)
reduceLROnPlat = ReduceLROnPlateau(monitor='val_loss', factor=0.8, patience=3, verbose=1,
mode='auto', cooldown=5, min_lr=0.00001)
early = EarlyStopping(monitor="val_loss",
                      mode="min",
                      patience=3) # probably needs to be more patient
callbacks_list = [early, reduceLROnPlat, checkpoint]

%%time
history_2 = retina_model_finetune_first10.fit(train_gen,
                           steps_per_epoch = downsampled_df.shape[0]//batch_size,
                           validation_data = valid_gen,
                           validation_steps = valid_df.shape[0]//batch_size,
                              epochs = 10,
                              callbacks = callbacks_list,
                             workers = 0, # tf-generators are not thread-safe
                             use_multiprocessing=False,
                             max_queue_size = 0,
                            verbose = 1
                            ).history

best_lr = float(retina_model_finetune_first10.optimizer.learning_rate)
print("Best learning rate:", best_lr)

import matplotlib.pyplot as plt
```

```
plt.plot(history_2['loss'], label = 'Training Loss')
plt.plot(history_2['val_loss'], label = 'Validation Loss')

plt.xlabel("Epoch")
plt.ylabel("Loss")
plt.title("Training and Validation Loss")

plt.legend()
plt.show()

from tqdm import tqdm_notebook
# fresh valid gen
valid_gen = flow_from_dataframe(valid_idg, valid_df,
                             path_col = 'path',
                            y_col = 'level_cat')
vbatch_count = (valid_df.shape[0]//batch_size-1)

out_size = vbatch_count*batch_size

test_X = np.zeros((out_size,)+t_x.shape[1:], dtype = np.float32)
test_Y = np.zeros((out_size,)+t_y.shape[1:], dtype = np.float32)

for i, (c_x, c_y) in zip(tqdm_notebook(range(vbatch_count)),
                         valid_gen):
    j = i*batch_size
    test_X[j:(j+c_x.shape[0])] = c_x
    test_Y[j:(j+c_x.shape[0])] = c_y

from sklearn.metrics import accuracy_score, classification_report
pred_y = retina_model_finetune_first10.predict(test_X, batch_size = 32, verbose = True)

pred_Y_cat = np.argmax(pred_y, -1)
test_Y_cat = np.argmax(test_Y, -1)
print('Accuracy on Test Data: %2.2f%%' % (accuracy_score(test_Y_cat, pred_Y_cat)))
print(classification_report(test_Y_cat, pred_Y_cat))

import seaborn as sns
from sklearn.metrics import confusion_matrix
sns.heatmap(confusion_matrix(test_Y_cat, pred_Y_cat),
        annot=True, fmt="d", cbar = False, cmap = plt.cm.Blues, vmax = test_X.shape[0]//16)

import tensorflow as tf
from tensorflow.keras.layers import Input, Dropout, GlobalAveragePooling2D, Dense
from tensorflow.keras.models import Model
from tensorflow.keras.applications import EfficientNetV2B1  # Import EfficientNetV2S or other
variant
from tensorflow.keras.layers import BatchNormalization

#Turn on Mixed Precision Training
tf.keras.mixed_precision.set_global_policy('mixed_float16')

# Define input layer
in_lay = Input(t_x.shape[1:])  # Specify the input shape here

# Load pre-trained model
base_pretrained_model = EfficientNetV2B1(weights='imagenet', include_top=False,
input_shape=t_x.shape[1:])  # Specify input shape here

# Unfreeze layers that are not BatchNormalization layers
base_pretrained_model.trainable = False

# Get the output features from the pre-trained model
pt_features = base_pretrained_model(in_lay)

# Global average pooling
gap_features = GlobalAveragePooling2D()(pt_features)

# Dropout and dense layers
gap_dr = Dropout(0.25)(gap_features)
dense_layer = Dense(128, activation='relu')(gap_dr)
dr_steps = Dropout(0.25)(dense_layer)

# Output layer
out_layer = Dense(5, activation='softmax', name = "output")(dr_steps)

# Create the model
retinav2_b1 = Model(inputs=in_lay, outputs=out_layer)
```

```
#Define topK Accuracy
from keras.metrics import top_k_categorical_accuracy

def top_2_accuracy(in_gt, in_pred):
    return top_k_categorical_accuracy(in_gt, in_pred, k=2)

#Use the focal loss function
import tensorflow as tf

# Define focal loss function
loss_fn = tf.keras.losses.CategoricalCrossentropy()

# Compile the model
retinav2_b1.compile(optimizer='adam',
                    loss= loss_fn,
                    metrics=['categorical_accuracy', top_2_accuracy])

# Print model summary
retinav2_b1.summary()

from keras.callbacks import ModelCheckpoint, LearningRateScheduler, EarlyStopping,
ReduceLROnPlateau

weight_path="{}_weights.best.hdf5".format('retina')

# sampler = SampleBalancingCallback(train_gen, batch_size = batch_size)
checkpoint = ModelCheckpoint(weight_path, monitor='val_loss', verbose=1,
                          save_best_only=True, mode='min', save_weights_only = True)
reduceLROnPlat = ReduceLROnPlateau(monitor='val_loss', factor=0.8, patience=3, verbose=1,
mode='auto', cooldown=5, min_lr=0.00001)
early = EarlyStopping(monitor="val_loss",
                    mode="min",
                    patience=3) # probably needs to be more patient
callbacks_list = [early, reduceLROnPlat, checkpoint]

%%time
history_2 = retinav2_b1.fit(train_gen,
                        steps_per_epoch = downsampled_df.shape[0]//batch_size,
                        validation_data = valid_gen,
                           validation_steps = valid_df.shape[0]//batch_size,
                              epochs = 10,
                              callbacks = callbacks_list,
                             workers = 0, # tf-generators are not thread-safe
                             use_multiprocessing=False,
                             max_queue_size = 0,
                            verbose = 1
                            ).history

import matplotlib.pyplot as plt
plt.plot(history_2['loss'], label = 'Training Loss')
plt.plot(history_2['val_loss'], label = 'Validation Loss')

plt.xlabel("Epoch")
plt.ylabel("Loss")
plt.title("Training and Validation Loss")

plt.legend()
plt.show()

from tqdm import tqdm_notebook
# fresh valid gen
valid_gen = flow_from_dataframe(valid_idg, valid_df,
                             path_col = 'path',
                            y_col = 'level_cat')
vbatch_count = (valid_df.shape[0]//batch_size-1)

out_size = vbatch_count*batch_size

test_X = np.zeros((out_size,)+t_x.shape[1:], dtype = np.float32)
test_Y = np.zeros((out_size,)+t_y.shape[1:], dtype = np.float32)

for i, (c_x, c_y) in zip(tqdm_notebook(range(vbatch_count)),
                         valid_gen):
    j = i*batch_size
    test_X[j:(j+c_x.shape[0])] = c_x
    test_Y[j:(j+c_x.shape[0])] = c_y

from sklearn.metrics import accuracy_score, classification_report
```

```
pred_y = retina_model_finetune_first10.predict(test_X, batch_size = 32, verbose = True)

pred_Y_cat = np.argmax(pred_y, -1)
test_Y_cat = np.argmax(test_Y, -1)
print('Accuracy on Test Data: %2.2f%%' % (accuracy_score(test_Y_cat, pred_Y_cat)))
print(classification_report(test_Y_cat, pred_Y_cat))

import seaborn as sns
from sklearn.metrics import confusion_matrix
sns.heatmap(confusion_matrix(test_Y_cat, pred_Y_cat),
        annot=True, fmt="d", cbar = False, cmap = plt.cm.Blues, vmax = test_X.shape[0]//16)
```

INFORMATION REGARDING STUDENT(S)

STUDENT NAME	EMAIL ID	PERMANENT ADDRESS	PHONE NUMBER	LANDLINE NUMBER	PLACEMENT DETAILS	PHOTOGRAPH
R S SREENIVASAN	SREENIVASANSUDHARSAN49@GMAIL.COM	53-1/53A, West Street, R K Naidu Layout, Venkitapuram, Coimbatore-641013	+91 9655067775	-	OPTER FOR PLACEMENTS	
ARVIND RATHORE V	VASISTHAARVIND@GMAIL.COM	38-38-80, FCI colony, Marripalem, Visakhapatnam-18	+91 8367025373	-	HIGHER STUDIES	
SAI RUTHVIK M	MSAIRUTHVIK@GMAIL.COM	16-9-534 4/1 Vikas Nagar Guntur Guntur Andhra Pradesh - 522007	+91 9059391729	-	HIGHER STUDIES	

s

Photograph Along With Guide