FACULTY OF SCIENCE, ENGINEERING AND

COMPUTING

School of Computer Science and Mathematics

MSc Degree

in

Data Science

Name: Prasanth Pradeep Kumar

Student Id: K2238510

Project Title: Implementation of Neural Network Models to Detect Glaucoma in Various Datasets

Date: 18 September 2023

Supervisor: Sarah Barman

Kingston University London

**Contents**

[**Abstract:** 3](#_Toc145922737)

[**Acknowledgments:** 4](#_Toc145922738)

[1. **Introduction and Background:** 5](#_Toc145922739)

[**2. Literature review:** 7](#_Toc145922740)

[**3.1 Retinal databases.** 10](#_Toc145922741)

[**3.2 Model Architecture.** 11](#_Toc145922742)

[3.2.1 SMOTE: 11](#_Toc145922743)

[**3.3 Model Compilation.** 12](#_Toc145922744)

[3.2.2 Data augmentation: 12](#_Toc145922745)

[**3.4 Model training.** 16](#_Toc145922746)

[**3.5 Model Evaluation.** 16](#_Toc145922747)

[**3.5.1 Accuracy, Precision, Recall and F1 Score.** 16](#_Toc145922748)

[**3.5.2 ROC Curve and Confusion Matrices.** 18](#_Toc145922749)

[**3.5.3 Bagging Ensemble.** 18](#_Toc145922750)

[**4. Results:** 18](#_Toc145922751)

[**4.1** **Result of ACRIMA.** 19](#_Toc145922752)

[**4.2 Result of Drishti-GS1 dataset.** 21](#_Toc145922753)

[**4.3 Result of RIM-ONE Dataset.** 24](#_Toc145922754)

[**4.4 Result of ORIGA-R Dataset.** 26](#_Toc145922755)

[**5. Discussion.** 28](#_Toc145922756)

[**5.1 Challenges:** 28](#_Toc145922757)

[**6. Comparison with existing works.** 29](#_Toc145922758)

[**6.1 Comparative Analysis of Model Performance:** 29](#_Toc145922759)

[**7.** **Ethics.** 30](#_Toc145922760)

[**8.** **Conclusion.** 30](#_Toc145922761)

[**8.1 Future Work:** 31](#_Toc145922762)

[**References.** 33](#_Toc145922763)

[**Appendix:** 37](#_Toc145922764)

List of Figures.

[**Figure 1 Digital fundus images cropped around optic disc. a Main structure of a healthy optic disc and b. [26] 6**](#_Toc145922865)

[**Figure 2 The first and second rows indicates the sample healthy and normal pictures from ACRIMA [25], Drishti-GS1 [24], RIM-ONE [23], and ORIGA-R [26] datasets, respectively. 10**](#_Toc145922866)

[**Figure 3: ROC Curve of ACRIMA Datasets. 20**](#_Toc145922867)

[**Figure 4 Confusion Matrices of ACRIMA Dataset. DenseNet and Xception models have much more consistent prediction accuracy across classes. 21**](#_Toc145922868)

[**Figure 5 Bar Graph of Drishti-GS1. 22**](#_Toc145922869)

[**Figure 6 ROC curve of Drishti - gs1 dataset. 23**](#_Toc145922870)

[**Figure 7 Confusion Matrices of Drishti-GS1. LeNet and Inception is more consistent in prediction on compared others models. 23**](#_Toc145922871)

[**Figure 8 ROC curve of RIM-ONE dataset. 25**](#_Toc145922872)

[**Figure 9 Confusion Matrices of RIM-ONE. LeNet and Inception has more poor prediction, as it has predicated few class1 as class 0. 25**](#_Toc145922873)

[**Figure 10 ROC curve of ORIGA-R dataset. 27**](#_Toc145922874)

[**Figure 11 Confusion Matrices of ORIGA-R dataset. 27**](#_Toc145922875)

# **Abstract:**

Glaucoma, a common eye condition, arises from damage to the optic nerve connecting the eye and brain, often due to fluid accumulation in the front of the eye leading to increased intraocular pressure. Untreated glaucoma can lead to irreversible vision loss, predominantly affecting older adults. This study focuses on employing deep learning architectures, including convolutional neural networks (CNNs), renowned for their image classification capabilities, to detect glaucoma from fundus images. Leveraging pre-trained models for weight initialization expedites and simplifies network training. The investigation extends to exploring transfer learning techniques to optimize the differentiation between Normal and Glaucoma images.

Expanding on the abstract, this research involves the creation of four distinct models—DenseNet, GoogleNet, Xception, and LeNet—applied to four diverse datasets—Drishti-GS1 [24], ORIGA-R [26], RIM-ONE [23], and ACRIMA [25]. The achieved accuracy ranges from 80% to 98%, showcasing the efficacy of these models in glaucoma detection.

Furthermore, when benchmarked against a state-of-the-art method, the DenseNet model outperformed it significantly. Specifically, the DenseNet model's AUC score of 95% surpassed the state-of-the-art method's score of 89%. Similarly, in terms of accuracy, the DenseNet model achieved a notable accuracy rate of 79.79%, while the state-of-the-art method recorded an accuracy of 78%

Datasets like ACRIMA [25], DRIHTI-GS1 has performed better than state-of-the-art method in all four neural network models. And ORIGA-R [26] dataset has performed better than others method performed recently. Visualizations depicting performance metrics provide insights into model effectiveness. Employing techniques such as hyperparameter tuning contributes to enhancing model outcomes with overfitting or underfitting. This comprehensive approach underscores the potential for accurate glaucoma diagnosis through neural network models across diverse datasets.

# **Acknowledgments:**

I extend my heartfelt gratitude to my supervisor, Professor Sarah Barman, for their unwavering support and guidance throughout the journey of completing this dissertation. Their valuable feedback, insightful suggestions, and willingness to lend a helping hand whenever I encountered challenges were pivotal in shaping this research.

I am indebted to the vast array of resources that enriched this endeavour. The wealth of knowledge provided by platforms like Google Scholar and IEEE played an integral role in deepening my understanding of neural network models and glaucoma detection.

For those interested in delving further into the details of my research, which focused on the implementation of neural network models to detect glaucoma across diverse datasets, please do not hesitate to reach out to me. I am more than willing to provide additional information and insights about the project.

Once again, I express my deepest appreciation to all who have been a part of this journey, directly or indirectly. Your support has truly made a difference.

Prasanth Pradeep Kumar

K2238510

September 2023

# **Introduction and Background:**

One of the leading causes of visual impairment worldwide is glaucoma, an irreversible neuro-degenerative eye disease [1]. The optic nerve head, sometimes referred to as cranial nerve II, is made up of about a million nerve fibres and is responsible for carrying visual information from the eyes to the brain. Any visual nerve injury might cause blindness. [1]. More than 50% of glaucoma sufferers in industrialised countries are currently estimated to go undiagnosed, and the ageing of the population suggests that the disease's symptoms will only worsen. [15]. Globally, glaucoma can afflict up to 100 million people. It is the second most prevalent cause of blindness in the majority of developed nations. [22].

The primary feature of this glaucoma is optic nerve fibre loss, which is caused by either decreased blood supply to the optic nerve or increased intraocular pressure (IOP). Since visual impairment might exist without a rise in IOP, it is discovered that IOP measurement is not specific nor sensitive enough to be a reliable glaucoma indication. Ganglion cell axons that eventually form the optic disc leave the eye at the optic nerve head. In a fundus picture (Figure 1), the optic disc may be distinguished visually into two zones: a centre, bright area known as the optic cup, and a periphery known as the neuro-retinal rim. [26]. Since glaucoma is a chronic, incurable condition, early detection and treatment are essential to halting its progression. [5]. The Digital Fundus Image (DFI) is one of the most common and popular ways to diagnose glaucoma. DFI has become the best modality for thorough glaucoma screening because to how simple it is to acquire them in a non-invasive way that is appropriate for screening on a big scale. [1].

A computer programme analyses each image in glaucoma screening software to see whether there are any signs of the condition. Only photographs that the algorithm deems to be suspect will be forwarded to ophthalmologists for additional evaluation, saving everyone a tonne of time. The conventional procedure used in clinics included a visual field loss test, an ophthalmoscopy-based visual inspection of the optic disc (OD), the medical history of the patient, and routine check-ups. Each picture used for clinical assessment must have the disc and cup manually marked, and it takes time for fundus images to automatically partition the disc and cup. [9].

All deep learning models have the drawback of requiring significantly more data and computing time than more traditional machine learning techniques. To lessen these demands, transfer learning has emerged as a critical method for transferring abilities learned from one work to another. [16]. The majority of medical image classification jobs do not have access to the vast datasets (thousands of pictures) that are available to train generic image classifiers. Networks developed on general image datasets may be utilised as an initial point for an ophthalmic assignment by utilising transfer learning. [16].

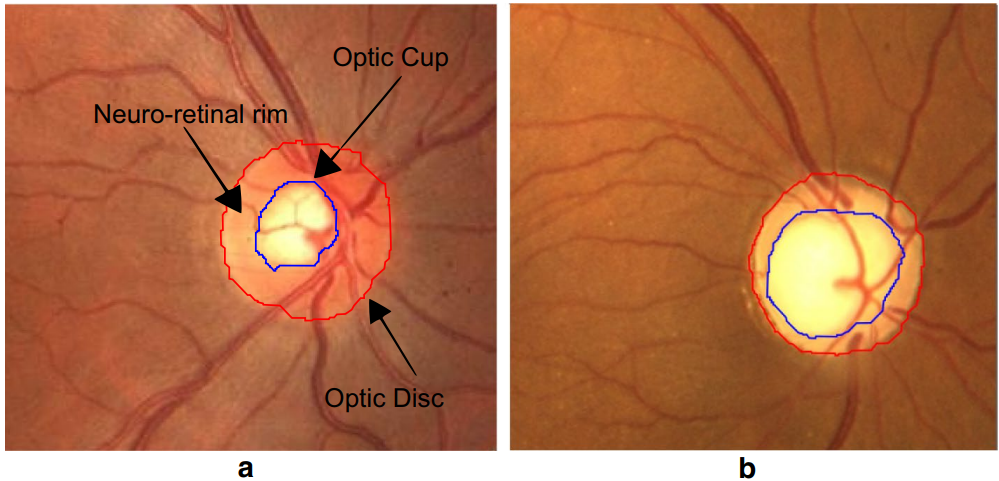
In this paper various datasets like ORIGA-R [26], ACRIMA [25] , RIM-ONE [23] and Drishti – GS1 [24] utilised dataset, which contains a number of retinal pictures labelled with the positions of the optic disc and macula as well as the contours of the optic cup and neuro-retinal rim.

Figure 1 Digital fundus images cropped around optic disc. **a** Main structure of a healthy optic disc and **b. [26]**

CNNs will automatically identify between glaucoma and non-glaucoma using images of the fundus. There are two main techniques in this field for automatically extracting characteristics and classifying them: Either transfer-learning is employed, or the networks are trained entirely from scratch. [3] [27].

As a result, the latest CNN models have already overtaken human specialists in several medical picture identification tasks. The government and individual patients will benefit the most from its implementation in glaucoma detection.

In this dissertation, four CNN architectures Densenet, GoogleNet, LeNet and Xception for glaucoma evaluation are given. This strategy does not need precise measurements of geometric optic nerve head features like CDR, in contrast to the majority of current detection methods. The high specificity, accuracy, and sensitivity discovered through this investigation point to ImageNet-trained CNN architectures as a reliable substitute for an autonomous glaucoma diagnosis system [25]. The primary goal of this research is to better accurately and quickly detect glaucoma since doing so will help treat the condition when it is first diagnosed. This paper is organised as follows.

Literature review sessions where various related works were reviewed and some knowledge about the advantages of using transfer learning, key references, and issues raised during the process. Most importantly learned how in recent years improvements have been made by reviewing the latest research papers. The methodology provides a clear overview of the whole process of how the detection of glaucoma is done in this research using neural network models through compilation, training, and evaluation. In the results how each dataset performed on the four deep learning models and how was the performance with some visual representation like ROC graph, Confusion Matrices and bar graph. Section comparison with the existing works will give a clear picture of the approach of other papers or works taken like model layers, datasets, image preprocessing, etc.

Every dataset used in this project has been checked to ensure that it is accessible to the public and has been properly cited. The obtained datasets were not labelled or cleansed, therefore I divided them into two files labelled "glaucoma" and "normal" to solve the problem. Since the entire project was completed in Google Collab, I uploaded the dataset to Google Drive and connected the two.

# **2. Literature review:**

Deep convolutional neural networks were used to build a glaucoma detection system in order to better define the underlying pattern of glaucoma. The loss of astrocytes and optic nerve fibres is the primary feature of the glaucoma disease. By measuring the size of the optic cup in relation to the optic disc and the thickness of the neuro-retinal rim, this loss may be assessed. Generally, numerous research in the literature have focused on the qualitative evaluation of the optic nerve head when employing fundus pictures.

|  |  |  |  |
| --- | --- | --- | --- |
| **Paper** | **Methodology** | **Datasets** | **Performance** |
| Chen et al. 2015 [1] | Overlapping pooling layers, data augmentation. | ORIGA-R [26], SCES | AUC: 0.831 (ORIGA-R [26]), 0.887 (SCES) |
| Wang et al. 2015 [14] | Hierarchical feature-based ensemble method. | Drive, Stare | Outperformed other techniques existed at 2015. |
| Bander et al. 2017 [21] | CNN-based system using raw images. | RIM-ONE [23] | Comparable results, low computational cost |
| Christopher et al. 2018 [17] | Deep learning (ResNet, VGG16, Inception). | ONH-fundus images | Accuracy: 0.91 (ResNet) |
| Serte et al. 2019 [19] | Generalized deep learning models. | sjchoi86-HRF, ACRIMA [25], RIM-ONE [23], Drishti-GS1 [24], and HRF. | 80% less computation time |
| Ovreiu et al. 2020 [3] | CNN with ResNet50. | Exclusive glaucoma fundus photos, ACRIMA [25], HRF, and ORIGA-R [26]. | Accuracy: 96.97% (early-stage diagnosis) |
| Poonguzhali et al. 2022 [29] | Single level stacking deep ensemble. | LAG, ACRIMA [25], ORIGA-R [26], RIM-ONE [23], and Drishti-GS1 [24]. | Outperformed state-of-the-art methods |
| Yann LeCun et al. 2012[30] | Introduction of CNNs. | ImageNet | Key role of CNNs in computer vision and AI |
| Carneiro et al. 2015 [34] | Transfer learning from ImageNet. | Medical imaging | Natural image-trained CNNs useful in medical imaging |
| Tajbakhsh et al. 2015 [35] | Comparison of pre-trained CNNs. | Medical imaging | Pre-trained CNNs outperformed freshly-trained CNNs |
| Diaz et al. [25] 2019 | Inception-v3, Resnet-50, Xception, VGG-16, VGG-19. | ACRIMA [25], HRF, RIM-ONE [23], and Drishti-GS1 [24] | Xception with average accuracy of 0.9605 |
| Zhen Y et al. [23] 2015 | Densenet, Inception-v3, Nasnet Mobile, Resnet, VGG-16, VGG-19, Xception. | Hospital-collected fundus pictures | Densenet achieved 75.50% accuracy |

Table 1Literature review.

For instance, Chen et al. in 2015 [1] employed overlapping pooling layers to prevent problems like overfitting in neural networks and to improve the performance of the model, data augmentation was also incorporated. On the ORIGA-R [26] and SCES datasets, they found AUC values for glaucoma detection of 0.831 and 0.887, respectively. These AUC values indicate that the authors' model shown promising discrimination skills, especially on the SCES dataset. Due to the tactics they used, such as overlapping pooling and data augmentation, the model may have been more successful in differentiating between glaucomatous and non-glaucomatous patients in the SCES dataset, as seen by the higher AUC.

A method proposed by Wang et al. In 2015 [14] is based a hierarchical feature-based ensemble learning method for segmenting retinal blood vessels with many distinct features, such as trainable feature extractor in CNN, random forest with ensemble classifier method, and the entire method is automatic and trainable with the help of feature and ensemble learning. They have made use of the Drive and Stare databases. Even though the model's calculation took 120 minutes for just one epoch, it nonetheless performed better than any other state-of-the-art method at 2015.

In the work made by Bander et al. in 2017 [21] presented a fully automated system based on CNN that classifies healthy and glaucoma images by using raw, coloured images without any enhancement. They employed the RIM-ONE [23] dataset, which contains 200 glaucoma pictures and 255 normal images. In the training step, they employed a pre-trained CNN model (AlexNet) that has 23 layers, including convolution layers, max pooling layers, fully connected layers, SoftMax layers, and output layers. Their results were compared favourably to state-of-art with considerable low computational cost.

Christopher et al. in 2018 [17] developed and evaluated deep learning methods for glaucoma damage detection in ONH fundus pictures. They used a large ONH-fundus images database. Their best performing model was the transfer learning ResNet model with 0.91 accuracy in detecting glaucoma images. They also evaluated VGG16 and Inceptions as well.

In the paper published by Serte et al. in 2019 [19] developed a generalised deep learning model for glaucoma fundus images using five distinct datasets with data augmentation, four of which were utilised for training and one of which was solely used for testing. Deep learning architectures of fifty, one hundred, and fifty-two layers of ResNet and twenty-one layers of GoogLeNet were employed. Their model ran in 80% less time than similar earlier studies.

Another approach using CNN with residual model is presented by Ovreiu et al. in 2020 [3], with a 96.97% accuracy rate for glaucoma early-stage diagnosis. They present an exclusive collection of colour photos of early-stage glaucoma fundus. A ResNet50 network that was first trained on the ImageNet dataset was utilised. The findings suggest that a cost-effective screening tool for the early and cost-effective identification of glaucoma may be developed utilising deep learning algorithms.

Poonguzhali et al. in 2022 [29] employed the single level stacking deep ensemble framework technique, and numerous neural networks, including AlexNet, GoogLeNet, VGG-16, and other models, outperformed state-of-the-art methods for classifying images of glaucoma. Additionally, they employed a number of openly accessible datasets, including DRISHTI-GS1 [24], ORIGA-R [26], LAG, ACRIMA [25], and RIM-ONE [23]. Each model's accuracy varied according to the datasets.

Yann LeCun [30] originally introduced convolutional neural networks (CNNs), which are multilayer perceptron variants with biological inspiration. Since then, they have become utilized for artificial intelligence and computer vision. The major objective of the 2012 ImageNet competition, which used a subset of the ImageNet [31] dataset, was to determine the content of natural photographs for the intent of automatic annotation. This is when their usefulness was first understood. They were successful by utilizing GPUs, rectifiers like ReLU, data augmentation strategies, and cutting-edge regularisation algorithms like Dropout [32]. The primary strength of CNN architectures rests in their capacity to extract highly distinctive characteristics at many abstraction levels [33].

Starting from scratch to train a CNN is a difficult process. They need a significant amount of labelled data, which is challenging to get for the glaucoma evaluation task, as well as processing power. Nevertheless, there are two methods that have been used in the past to train a CNN starting scratch and have been successful in classifying medical images. The first option is optimising a CNN that was trained using a sizable labelled dataset in a separate application (like ImageNet, for example). This alternative's work by Carneiro et al [34] serves as an illustration. They demonstrated that despite the obvious disparities in picture appearance, CNN models that have been trained on natural images, like the ImageNet, are helpful in medical imaging applications.  Another illustration is the research done by Tajbakhsh et al [35] in, who ran a series of tests for four medical imaging applications to demonstrate how well-performing pre-trained CNNs outperformed freshly-trained CNNs.

The second alternative option involves applying an ImageNet-trained CNN to an input picture and then extracting features from a specific hidden layer of the network using the CNN as a feature extractor. Then, a new classifier such as support vector machine (SVM), decision tree, K nearest neighbour, or Naive Bayes classifier is trained using the retrieved features. For instance, Bar et al [36] employed pre-trained CNNs as a feature extractor for identifying chest pathology.

Reference [25] by Diaz et al. discusses the performance evaluations of Inception-v3, Resnet-50, Xception, VGG-16, and VGG-19. 1707 pictures from the DRISHTI-GS1 [24], RIM-ONE [23], sjchoi86-HRF, ACRIMA [25], and HRF databases are used to evaluate models. According to the authors, the Xception model outperforms the competition with an average accuracy of 0.9605. For the purpose of glaucoma diagnosis, Zhen et al. [37] have investigated the performance of Densenet, Inception-v3, Nasnet Mobile, Resnet, VGG-16, VGG-19, and Xception. With the help of 5978 hospital-collected fundus pictures, models are trained and evaluated. The greatest accuracy of the Densenet model is 75.50%.

The analysis of four different CNN architectures utilised as glaucoma classifiers is described in this paper. Given their excellent specificity, sensitivity, and accuracy, ImageNet-trained CNN architectures appear to be a solid substitute for automatic glaucoma diagnosis algorithms. These CNNs perform well with high-variability fundus pictures from four separate public datasets.

Majority of the earlier research included combining all datasets that are openly accessible, then training models on this combined dataset. On the other hand, some of the research has used private databases obtained from hospitals. However, it is clear that there is a distinct lack of thorough investigations that make use of technologies like LeNet and DenseNet and incorporate concepts like data augmentation and efficient handling of class imbalances.

The research gap in this field has highlighted the need for an investigation into model performance, with a particular focus on LeNet and DenseNet. The study aims to employ a comprehensive methodology, including data augmentation techniques, addressing class imbalance issues, and implementing hyperparameter tuning. Through the systematic application of these strategies, the investigation seeks to contribute to a deeper understanding of best practices for enhancing the efficacy of deep learning models when applied to specific datasets. This project represents a significant step towards addressing existing knowledge gaps. A justification for the selection of neural network designs, specifically Xception, DenseNet, LeNet, and GoogleNet, within this research initiative, resulted from a thorough analysis of the applicable body of literature. The rigorous use of these models, together with the incorporation of other approaches including sophisticated pre-processing methodology, data normalisation processes, and exact class handling strategies, comprised the project's next phase. By contrasting the models' comparative efficacy with recognised research paradigms in the area, this methodological synthesis intended to strengthen the performance traits of the models and validate their comparative effectiveness. This strategy approach was carefully developed with the overriding goal of improving the glaucoma detection system's precision and efficacy and so confirming its pioneering status within the field of medical image analysis.

**3. Methodology.**

## **3.1 Retinal databases.**

In this work, publicly accessible retinal databases including DRISHTI-GS1 [24], RIM-ONE [23], ORIGA-R [26], and ACRIMA [25] are taken into consideration (Table 1). Sample glaucoma and healthy photos from multiple databases are shown in Figure 2.

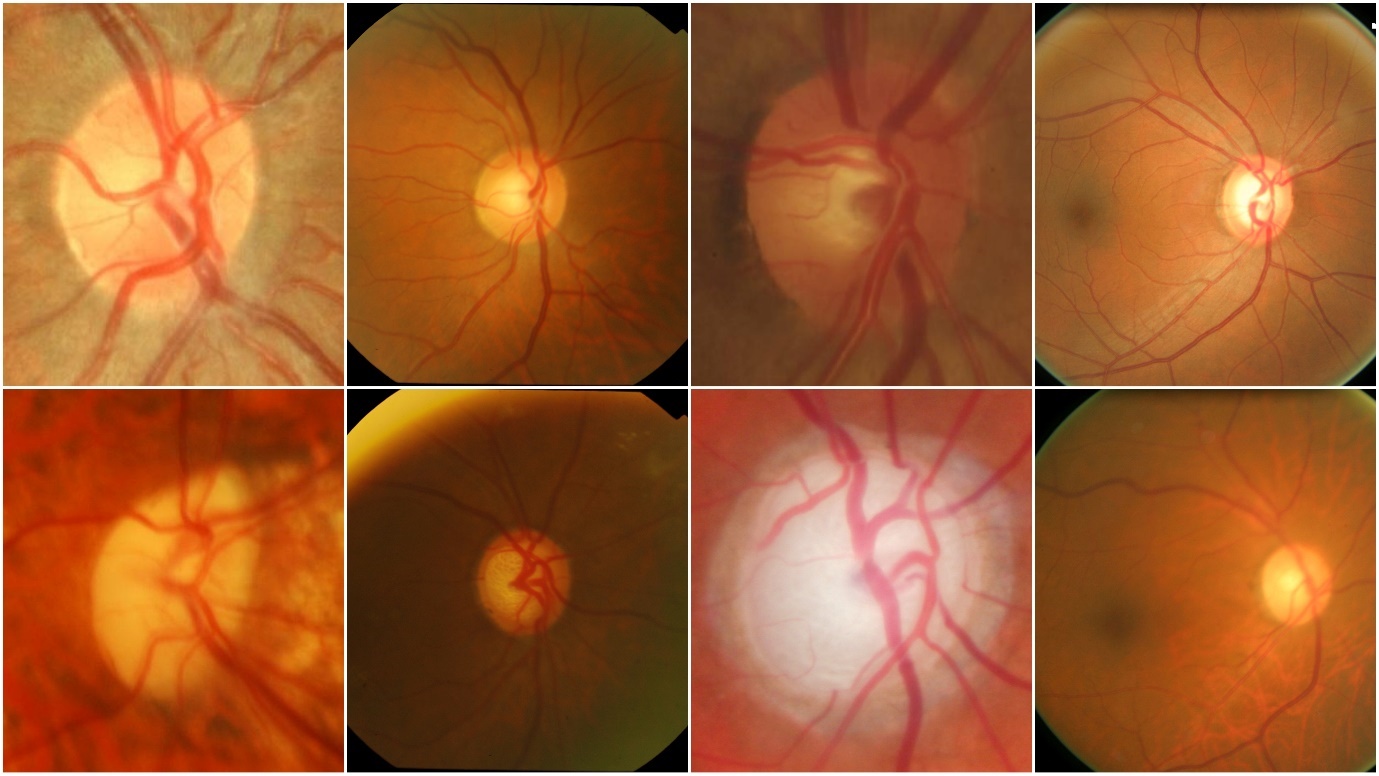


Figure 2 The first and second rows indicates the sample healthy and normal pictures from ACRIMA [25], Drishti-GS1 [24], RIM-ONE [23], and ORIGA-R [26] datasets, respectively.

The datasets chosen for this study included a wide range of freely accessible resources, each of which added unique traits and dimensions to the investigation. ACRIMA [25], a collection of 705 precisely documented eye fundus pictures, provided a wide range of information. An extensive study was made possible by the dataset's combination of 396 glaucoma-affected pictures and 309 usual fundus photos. 560 photographs from a subset were given to the training cohort, and 145 more images made up the carefully chosen test set.

In addition, the RIM-ONE [23] DL dataset [23], which contains 485 images of the retinal fundus in high resolution, was an important resource. These photos, which were painstakingly selected from Spanish healthcare facilities, offered a balanced distribution, with 313 shots of people without glaucoma and 172 photographs of those who had the disease, providing a thorough depiction of pathological abnormalities.

The DRISHTI-GS1 [24] dataset , well known for serving as a standard in the field, added a complex layer to the investigation. This dataset included a collection of 31 normal fundus pictures with their 70 glaucoma-tagged counterparts, along with ground truth data for segmenting the optic disc and cup. Thereafter, 50 photos were set aside for training purposes and 51 were reserved for careful testing.

The inclusion of the ORIGA-R [26]-Light dataset [26] considerably enhanced the study tapestry by adding a collection of 650 photos. In this collection, 168 glaucoma-affected fundus photos and 482 pristine normal fundus images converged, increasing the variety of pathologies being studied. The ORIGA-R [26]-Light dataset [26] acted as a repository that could be used for both diagnosing glaucoma and diabetic retinopathy.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Database** | **Total Images** | **Collected from** | **Glaucoma Images** | **Normal Images** | **Resolution** |
| **ACRIMA [25]** | 705 | ACRIMA [25] Project Spain | 396 | 309 | 178 x178 to 1420 x 1420 JPEG. |
| **ORIGA-R [26]** | 650 | Singapore eye research centre | 168 | 482 | 3072 x 2048 JPEG |
| **RIM-ONE [23]** | 485 | Collaboration Spanish hospitals | 172 | 313 | 290 x 290 to 1375 x 1654 JPEG. |
| **DRISHTI-GS1 [24]** | 101 | Aravind eye hospital | 70 | 31 | 2896 x 1944 PNG |

Table 2 Glaucoma Databases

There is a clear bias in the data distribution of the Drishti-GS1 [24] and ACRIMA [25] datasets, with a plurality of cases falling under the glaucoma category. Comparatively, a trend where the distribution strongly skews towards the normal class appears when examining the makeup of the ORIGA-R [26] and RIM-ONE [23] datasets. In order to provide a balanced and equal evaluation across all datasets, this inherent data distribution difference required the deployment of specialised algorithms for the efficient handling of class imbalance throughout the model training and evaluation stages.

## **3.2 Model Architecture.**

Combining many models reduces prediction variance while also yielding more accurate results than using just one model alone. Data augmentation is used to construct a larger database since deep learning requires a lot more photos to train. The performance, consistent accuracy, and precision of neural network models have improved as a result of implementation. SMOTE was chosen to be the approach used moving forward for all models since it showed the greatest performance. Pre-trained models are used as the basic learners, and it is necessary to fine-tune these base learners in order to provide predictions.

## 3.2.1 SMOTE:

When training Deep Learning architectures, the problem of data imbalance must be addressed. A typical data augmentation method in the context of unbalanced classification issues, especially those requiring neural network models, is called SMOTE (Synthetic Minority Over-sampling Technique). Imbalanced classification refers to situations when there is a considerable disparity in the number of samples in the various classes, which results in skewed learning and worse performance on the minority class.

To balance the class distribution, SMOTE uses synthetic samples of the minority class. By extrapolating across nearby minority class samples, it creates new synthetic instances. Here is how SMOTE functions:

1. Pick a sample from the minority class in the data set.

2. Determine its k closest classmates (usually using distance metrics like Euclidean distance).

3. Choose one of the k closest neighbors at random.

4. Create a synthetic sample by combining the minority sample and its chosen neighbor in a linear fashion.

- For instance, if the minority sample is labeled B and the randomly chosen neighbor is labeled A, a new synthetic sample may be made by averaging the characteristics between A and B. Using the formula new\_sample = B + rand(0, 1) \* (A - B), a random integer between 0 and 1 is generated using the function rand(0, 1).

5. Up until the appropriate degree of minority class oversampling is reached, repeat steps 1-4.

SMOTE increases the representation of the minority class in the training data and balances the distribution of the classes by creating synthetic examples. For neural network models, this can be advantageous since it enables them to learn from more varied examples and enhances their capacity to correctly categorize cases belonging to minority classes.

## **3.3 Model Compilation.**

Important choices are made regarding the loss function, optimizer, and evaluation metrics during the configuration and training of the neural network models. These decisions set the stage for how well the models learn and carry out their separate jobs. The choice of an appropriate loss function is critical for the particular job at hand, which is binary classification. The 'binary\_crossentropy' loss function has been carefully selected for this investigation. This loss function is specifically designed for instances involving binary classification like the one discussed below. As a compass for the model's learning process, it measures the discrepancy between the projected probabilities and the actual binary labels. The goal is to reduce this loss, which will improve the model's ability to accurately categorise medical pictures into glaucoma- and non-glaucoma-related categories.

The Adam optimizer has been skillfully used to make weight adjustments during training easier. Adam, which stands for Adaptive Moment Estimation, is well known for its flexibility and effectiveness in enhancing a wide variety of neural network models. The dynamic modification of learning rates for certain model parameters is its most notable feature. When working with complicated structures and datasets with fluctuating gradients, this adaptability is especially helpful.

Adam stands out because it combines the advantages of RMSprop and Adagrad, two other well-known optimizers. It incorporates RMSprop's resistance to changes in gradient magnitudes along with Adagrad's flexibility to customise learning rates for each parameter. This combined strategy gives the Adam optimizer the ability to efficiently negotiate complicated loss landscapes, converge quickly during training, and successfully handle datasets with sparse gradients.

This research has taken a meticulous and thoughtful approach to make sure that the neural network models are not only well-suited for the task of classifying medical images but are also well-equipped to succeed in the dynamic and demanding field of deep learning. This research is exemplified by the careful selection of these components.

## 3.2.2 Data augmentation:

Data augmentation is a machine learning approach that, when applied to existing data, may be used to artificially enhance the amount of data of the training dataset, notably in neural network models. While adding changes that might enhance the model's generalization and performance, these transformations are normally implemented in a way that respects the semantic meaning of the input.

The main objective of data augmentation is to expose the model to a greater variety of variations and improve its capacity for pattern recognition and good generalization to new data. Data augmentation can lessen overfitting, which occurs when the model gets very particular to the training data and performs badly on new, unforeseen data, by supplying more varied instances.

Data augmentation techniques may be employed with a variety of data kinds, but images tend to receive the most attention. In the process of enhancing picture data, some frequent transformations are as follows:

1. Flipping: Horizontal or vertical mirroring of the picture.

2. Rotation: An angle-specific rotation of the picture.

3. Resizing the image by zooming in or out is known as scaling.

4. The picture is being moved either horizontally or vertically.

5. Shearing: Tilting the picture along one axis to distort it.

6. Zooming: Increasing or decreasing the size of the image while keeping its aspect ratio.

7. Cropping: Deciding on a more focused area of the photograph.

The augmented dataset gives the neural network more training examples that show diverse versions of the original data by applying these adjustments. This enhances the model's capacity to generalize and perform well on fresh, untested data by assisting it in learning more robust and invariant characteristics. It is crucial to keep in mind that data augmentation is frequently only used during the training phase and not the evaluation or testing phases. The model's parameters are updated using the augmented data, and its performance is assessed using the original, unmodified data.

While SMOTE aids in addressing class inequality, data augmentation approaches can offer further advantages. By incorporating different transformations, they can contribute to the creation of a more varied and robust training dataset, improving the model's capacity to handle fluctuations and generalize effectively to new data. When there is a lack of data and creating synthetic samples may not be enough, data augmentation can also be helpful. The performance and generalization abilities of your neural network model are therefore further enhanced in this research by combining SMOTE with data augmentation approaches.

It became clear that the Synthetic Minority Over-sampling Technique (SMOTE) and Data Augmentation stood out as the most successful tactics after a thorough review of several ways to solve class imbalance and improve the dataset quality. These techniques performed better than others in increasing model robustness and accuracy. Given the significant gains seen when SMOTE and Data Augmentation were used together, it was decided that both techniques should be applied uniformly across all models in the research's later stages. The aim to maintain consistency in model training, assessment, and comparison, which would eventually result in more dependable and robust outputs, served as the driving force for this choice.

A common Deep Learning strategy is transfer learning, which allows a model that has been trained for one job to be modified for another task. The model's starting layers and optimized weights are transferred in order to do this. The final layers, however, are adjusted correspondingly. In this study, pre-trained models such as Xception, Densenet-201, Inception-v3, and Lenet are used. These models are often trained with more photos. They have accumulated a wide variety of raw photos as a consequence. Each basic model undergoes fine tuning by being completely unfrozen and being trained with a very slow learning rate on the retinal datasets. Better performance is the consequence, particularly for small datasets. The reason for selecting these models is because they are more intriguing, have better performance compared to others, and some, like LeNet, are more specialized exclusively for classifying medical images.

The Base model performs well at feature extraction; however, it lacks the last layers needed for the current classification challenge. It is possible to learn to integrate the extracted feature into higher-level representations that are pertinent to the classification problem by adding a flatten layer and a hidden layer, each of which has 256 neurons and a rectified linear unit (ReLU) activation function. The model's output is transformed into a probability distribution along the classes via the softmax activation function, enabling class predictions.

By including these custom classification layers, a new model architecture has been created effectively that makes use of the pre-trained base model's feature extraction skills and is customized to particular classification requirements. This method makes use of the information and feature representations that the pre-trained model has learnt, allowing you to perform well even with a small amount of task-specific data. Transfer learning is a method that helps to apply information from one job—in this example, pre-training on ImageNet (Xception, DenseNet and Inception V3).

**DenseNet:**

A deep convolutional neural network architecture called DenseNet (Densely Connected Convolutional Networks) is renowned for its innovative and effective layout. In their study titled "Densely Connected Convolutional Networks" from 2016, Gao Huang, Zhuang Liu, Laurens van der Maaten, and Kilian Q. Weinberger introduced it [28]. When it comes to image categorization jobs, DenseNet excels. Reason for using DenseNet are usage of them makes it simpler for neural network models to learn intricate patterns and features from medical pictures, such as glaucous images, as it encourages feature reuse across layers where minute details might be critical. DenseNet also helps alleviate the vanishing gradient problem. In comparison to other deep architectures, such as conventional CNNs, they have less parameters, which also lowers the possibility of overfitting. In a variety of image classification applications, including medical image classification, DenseNet has produced state-of-the-art results. It is particularly suited for applications where correct classification depends on tiny visual signals because of its ability to collect fine-grained information in pictures.

The **architecture** of DenseNet is characterized by dense connections between layers, enabling effective information transfer and feature reuse. A summary of the architecture is provided below:

1. **DenseNet Model (Base) (Pre-trained):**

The basic DenseNet model is often pre-trained on a large dataset (such as ImageNet) to extract common characteristics from pictures. The DenseNet121 model with pre-trained weights from "imagenet" has been loaded in this study. A picture with three color channels (RGB) and dimensions determined by the variables "img\_width" and "img\_height" is commonly used as the input to this model.

1. **Layers of Custom Classification:**

To customize the network for particular classification, new classification layers on top of the standard DenseNet model should be added. In this paper layers are built one at a time using a sequential paradigm. The output of the base model, which is often a tensor with spatial information, is transformed into a one-dimensional vector using the Flatten layer. The next two levels (Dense) are entirely linked. The first contains 256 neurons with a ReLU activation function, and the following has as many neurons with a softmax activation function as there are classes in the classification task. The final class predictions are produced by these layers. DenseNet's layer-to-layer connections promote feature reuse and gradient flow, which can lead to more accurate and effective models.

**Xception:**

A deep convolutional neural network design called Xception (Extreme Inception) builds on the Inception architecture. In his 2017 work Xception: Deep Learning with Depth wise Separable Convolutions, Keras founder François Chollet et al. [39] presented it. Xception is meant to be computationally effective while providing state-of-the-art performance on picture categorization problems. The design of Xception is built on depth-wise separable convolutions, which are computationally efficient when compared to conventional convolutional layers, and this is the main reason for utilizing it. Since huge datasets and intricate models can be computationally demanding in the categorization of medical images, this efficiency is very useful. Due to its sophisticated architecture and numerous parameters, it has a high model capacity. This capability enables it to extract strong and discriminative features from medical pictures, even in the face of noise or fluctuations in image quality, and to learn detailed features and patterns from medical images, both of which are crucial for obtaining high classification accuracy.

**Design by Xception:**

The foundation of Xception's design is a revolutionary idea known as "depthwise separable convolutions," which is a more effective convolutional algorithm.

An outline of the architecture is given below:

1. **Entry Flow:** To extract low-level information from the input picture, the network starts with a number of convolutional and depthwise separable convolutional layers. Following these layers are activation and batch normalization algorithms like ReLU.
2. **Middle Flow:** The network's middle section has a number of repeating blocks. Batch normalization, ReLU activation, and depthwise separable convolutions are all included in each block. These building components provide the model the ability to collect ever-more-complex properties as data moves through the network.
3. **Exit Flow:** The network's exit flow consists of softmax activation, completely linked layers, and average pooling. The feature maps' spatial dimensions are reduced using average pooling. High-level feature extraction is carried out by fully linked layers, which also provide the final class predictions.
4. **Skip Connections:** Similar to ResNet, Xception also uses skip connections or residual connections, which lessen the effect of vanishing gradients during training.

**Inception V3 (GoogleNet):**

Deep convolutional neural network architecture Inception V3, also known as GoogleNet, was created by Google researchers. It is intended for image classification and associated computer vision problems and is a member of the Inception family of models. The image recognition software Inception V3 is renowned for its effectiveness and superior performance. They was chosen since it is made to collect multi-scale characteristics in a picture. These modules combine the outputs of convolutional filters with various sizes (1x1, 3x3, and 5x5). This multi-scale method is useful for identifying various patterns and anomalies since it enables medical pictures capture both fine and coarse information [47]. They were created with the efficiency of computing in mind. In comparison to other deep networks, its deep design delivers competitive performance while utilizing fewer parameters. This effectiveness is crucial since computational resources might be scarce in the interpretation of medical images.

**Architecture of Inception V3:**

The widespread usage of inception modules, which are groups of convolutional layers with different filter sizes, is what distinguishes the architecture of Inception V3. An outline of its architecture is given below:

1. **Input layer.**

A picture is first entered into the network's input layer. The dataset usually determines the input size, which is frequently in RGB format.

1. **The Stem Network.**

Multiple convolutional and pooling layers make up the stem network, the network's initial component. It aids in the input image's fundamental feature extraction.

1. **Inception Modules.**

Multiple Inception modules piled on top of one another make up the core of Inception V3.

A mix of 1x1, 3x3, 5x5, and max-pooling procedures are included in each inception module. Various sizes and resolutions of features can be captured by these modules. Convolutions of size 1x1 are very useful for reducing dimensionality.

1. **Global average pooling**.

Global average pooling is used to reduce the feature maps' spatial dimensions after the final inception modules.

1. **Fully Connected Layers (Output layer).**

A group of fully connected layers are coupled to the output of global average pooling, which performs high-level feature extraction and generates the final class predictions. Usually, class probabilities are generated in the last layer using softmax activation.

**LeNet-5:**

LeCun et al. 's (1998) original LeNet-5 model [48], a pioneering convolutional neural network design, has found extensive applications as the foundational architecture for various image categorization tasks. The adaptation of the LeNet-5 model in this study has been tailored to address the specific requirements of the categorization problem at hand. Layers that are completely linked and convolutional make up the LeNet-5 architecture. An average pooling layer is placed after each convolutional layer to gradually lower the number of spatial dimensions and capture useful information. Densely linked neurons for categorization make up the last layers. The architecture of the model is built to gradually extract hierarchical characteristics from the input data, leading to a SoftMax activation layer that generates class probabilities [30]. LeNet has a limited ability to learn and represent the complex hierarchical elements seen in medical pictures because of its shallow architecture. For successful medical image analysis, deeper architectures must be able to capture a larger variety of characteristics and correlations in the data.

**Architecture for LeNet-5:**

Although LeNet-5's design is relatively straightforward when compared to contemporary CNN architectures, it was essential in the advancement of deep learning. An outline of its architecture is given below:

**1. Input Layer:** The input layer of the network receives a picture as input at the outset. The dataset normally determines the input size, which is frequently 32x32 pixels with three color channels (RGB).

**2. Layers for convolution and pooling:** Three sets of convolutional and average pooling layers make up LeNet-5. A group of convolutional filters are applied to the input image in each convolutional layer in order to extract local features. An average pooling layer decreases the spatial dimensions of the feature maps after each convolutional layer, aiding in the acquisition of crucial data while minimizing computation.

**3. Fully Connected Layers (MLP):** The network reduces the output into a one-dimensional vector after the convolutional and pooling layers. The Multi-Layer Perceptron (MLP), often known as a sequence of completely linked layers, processes this flattened vector. Three completely linked layers with ReLU activation functions are included in your code. The last fully connected layer often employs a softmax activation function to generate class probabilities, and it includes as many neurons as there are classes in the classification issue.

LeNet-5, a ground-breaking design when it was first released, showed how well deep neural networks performed image recognition tasks. By today's standards, it may be seen as being very straightforward, yet it set the stage for more intricate CNN systems.

## **3.4 Model training.**

After model compilation, a thorough explanation of the model training procedure is given. The weights of the model are iteratively adjusted as they are exposed to the training dataset during this critical stage, and the model's performance is continuously evaluated using a different validation dataset. Through the "datagen," which creates enhanced data batches instantly, a dynamic data augmentation approach is used to improve the training process.

For the training operation, a number of important parameters are carefully configured:

**Batch Size**: The "batch\_size" is set to 32, indicating that each iteration of the model's training uses batches of 32 samples.

**Number of Epochs:** The model learns and improves its representations over the course of 10 training epochs.

**Validation Data:** The validation dataset (X\_test and y\_test) is used to thoroughly assess the model's performance after each epoch. The model's ability to generalise to new data is determined in large part by this phase.

**Early Stopping:** Including an "early\_stopping" callback in the training phase is crucial for preventing overfitting. This technique tracks the validation loss continually. The training procedure is terminated if the validation loss has not decreased after a set number of epochs. Early halting prevents the model from falling into the trap of memorising the training data, allowing it to properly generalise from the data it encounters.

All things considered, this training process is precisely planned to not only make it easier to acquire pertinent information from the training data but also to make sure that the deep learning model that emerges is able to generalise and make precise predictions on fresh, untested data.

## **3.5 Model Evaluation.**

### **3.5.1 Accuracy, Precision, Recall and F1 Score.**

When it comes to model evaluation, a thorough and exacting process is used to gauge how well each neural network model performs when exposed to test data that has never been seen before. The effectiveness of the model in producing precise predictions on new data instances depends critically on the results of this review procedure.

The use of the variables "X\_test" and "y\_test," which represent the test data and their corresponding labels, is crucial to the assessment procedure. The model's performance is measured using critical evaluation measures, including test loss and accuracy, which are painstakingly computed.

The determination of accuracy, recall, and F1 scores also allows for a more thorough examination of the neural network models' categorization abilities. These metrics are crucial determinants of how well the models classify data. For instance, precision explores the proportion of times a sample is correctly classified as belonging to a particular class, indicated as class "x" [27]. The equation below is the formal expression for this metric:

**Precision = Number of True Positives / (Number of True Positives + Number of False Positives)**

The foundation for thoroughly evaluating the capability and dependability of deep learning models in identifying various medical picture datasets is this complex assessment approach.

The proportion of occurrences inside a given class, designated as class "x," that the model correctly recognised is measured by the recall score for that class. The fraction of samples successfully classified into class "x" is what it measures, in comparison. This important indicator highlights the model's sensitivity to true positive classifications and provides insight into its ability to accurately identify all occurrences of a given class. The following equation serves as the official definition of the recall score for class "x":

**Recall = Number of True Positives/ (Number of True Positives + Number of False Negatives)**

This measure is crucial in determining how well the model can identify particular classes in the dataset and is an essential part of the overall evaluation of its classification performance.

The harmonic mean of accuracy and recall is represented by the important assessment statistic known as the F1 score [4], which is calculated using the formula below:

**F1 Score = 2 x (Precision x Recall) / (Precision + Recall).**

Taking into account both the accuracy and recall values of a model, this statistic provides a fair evaluation of a model's total performance. It is a critical gauge of the model's propensity to produce thorough and accurate classifications across various dataset classes.

The mean accuracy for all classes in the dataset is taken into account, with the weighting dependent on the quantity of samples for each class. When there is an unequal distribution of samples among classes, as in glaucoma datasets, this approach is quite helpful. Although the F1 score is a helpful indicator, it has the drawback of giving accuracy and recall the same weight, which could not exactly reflect real-world circumstances. It is frequently more important to concentrate on recall in tasks involving medical picture classification, notably for glaucoma detection, since it directly links to the model's capacity to recognise positive instances. Therefore, accuracy and recall will be the main criteria used to compare the effectiveness of algorithms for classifying glaucoma. These metrics offer a comprehensive assessment, taking into account both the model's general accuracy and its capacity to identify pertinent positive examples. Table 2 in the Results section contains the accuracy, precision, recall, f1 score, and AUC of all models.

### **3.5.2 ROC Curve and Confusion Matrices.**

The evaluation process integrates Receiver Operating Characteristic (ROC) curves and confusion matrices, pivotal tools for comprehensively assessing model performance [44].

**Receiver Operating Characteristic (ROC) Curves:**

ROC curves are graphical depictions used to examine how well classification models like DenseNet, Inception, Xception, and LeNet are able to differentiate between different classes. The trade-off between two basic measurements, the true positive rate (sensitivity) and the false positive rate (specificity), is visually represented by these curves. Specificity rates the model's power to precisely identify negative cases, whereas sensitivity indicates how well it can categorise positive instances. By changing the classification threshold and tracking how these rates alter as a result, ROC curves are visualised. In essence, ROC curves shed light on a model's ability to distinguish between several classes, which is crucial for tasks like glaucoma diagnosis.

**Confusion matrices.**

Confusion matrices provide a thorough tabular depiction of model predictions and their agreement with real class labels. They offer specific information on classification performance. Confusion matrices divide predictions into four groups in the context of glaucoma detection:

**True Positives (TP):** Instances that were appropriately identified as positive (for example, occurrences of glaucoma).

**True Negatives (TN):** Situations that were appropriately categorised as negative (for example, correctly identifying glaucoma instances that weren't).

**False Positives (FP):** Instances that are wrongly labelled as positive (for example, mislabeling cases of non-glaucoma as glaucoma).

**False Negatives (FN):** Conditions that were mistakenly labelled as negative (for example, failing to recognise real cases of glaucoma).

All four datasets' ROC graphs and confusion matrices are included in the Results section for each individual dataset segment, such as Figures 3, 4, and 5.

### **3.5.3 Bagging Ensemble.**

A Bagging Ensemble, also known as a Bootstrap Aggregating Ensemble, is a meta-algorithm used in deep learning and machine learning that combines the outputs of many base models to enhance the overall performance and resilience of a prediction model [46]. The notion of constructing several subsets (bags) of the training dataset using the bootstrapping technique, in which samples are chosen at random with replacement, is where the name "bagging" originates. Then, a different base model is trained using each of these subsets separately. Bagging ensemble operates by first bootstrapping the model, then training the base model, and lastly voting or averaging the base model.

Reason for adopting Bagging (Bootstrap Aggregating): When dealing with complex models like DenseNet, GoogleNet, Xception, and LeNet, which are already acknowledged as powerful learners in the field of machine learning, Bagging is a more appropriate ensemble technique to take into account than AdaBoost. Bagging is widely used with complex models for Bootstrap Sampling, training multiple models, and minimising variance. It may be a viable alternative for improving a model's performance. Section 4 contains the outcomes of all models on all four datasets.

# **4. Results:**

In this part, examination in great detail how well the neural network models performed on the ACRIMA [25], Drishti-GS1 [24], RIM-ONE [23], and ORIGA-R [26] datasets. LeNet, GoogleNet, DenseNet, and Xception are among the models that were tested. Investigation of the four main performance indicators: F1-Score, Accuracy, Precision, and Recall. Refer Table 2.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Dataset** | **Model** | **Accuracy** | **Precision** | **F1-Score** | **Recall** | **AUC** |
| **ACRIMA** | LeNet | 0.9434 | 0.8588 | 0.8902 | 0.9241 | **0.99** |
| GoogleNet | 0.9686 | 0.7117 | 0.8316 | 1.000 | 0.99 |
| DenseNet | 0.8994 | 0.8061 | 0.8927 | 1.000 | 0.98 |
| Xception | **0.9937** | 0.5683 | 0.7248 | 1.000 | 0.89 |
| **Drishti – GS1** | LeNet | **0.8214** | 0.7692 | 0.8902 | 0.9241 | 0.97 |
| GoogleNet | 0.6071 | 0.7143 | 0.8316 | 1.000 | **0.99** |
| DenseNet | 0.7143 | 0.9000 | 0.8927 | 1.000 | 0.98 |
| Xception | 0.7857 | 0.8000 | 0.7248 | 1.000 | 0.92 |
| **RIM-ONE** | LeNet | 0.7540 | 0.6875 | 0.7395 | 0.8000 | 0.89 |
| GoogleNet | **0.9127** | 0.8793 | 0.9027 | 0.9273 | 0.95 |
| DenseNet | 0.6905 | 0.5889 | 0.7310 | 0.9636 | 0.90 |
| Xception | 0.8651 | 0.9750 | 0.8211 | 0.7091 | 0.85 |
| **ORIGA-R** | LeNet | 0.7927 | 1.000 | 0.7701 | 0.6262 | 0.83 |
| GoogleNet | 0.7876 | 0.9853 | 0.7657 | 0.6262 | 0.87 |
| DenseNet | **0.7979** | 0.9474 | 0.6729 | 0.7869 | 0.87 |
| Xception | 0.6373 | 0.6194 | 0.7328 | 0.8972 | 0.84 |

Table 3 Results of all models in different databases.

When evaluating the effectiveness of machine learning and neural network models, particularly in classification tasks, the AUC (Area Under the Curve) score is an essential indicator. In combination with ROC (Receiver Operating Characteristic) curves, it is frequently employed. Following are some explanations of what these AUC scores in the framework of neural networks signify.

## **Result of ACRIMA.**

The LeNet model achieves accuracy of 94.34% on the ACRIMA [25] dataset, with remarkable precision, recall, and F1-Score. But because of a tiny precision deficit, it may occasionally provide false positive results. Second, while GoogleNet achieves a remarkable accuracy of 96.86%, its precision is lower, indicating a larger percentage of false positives. At 83.16%, the F1-Score—which strikes a balance between recall and precision—remains good. With balanced precision and recall for the ACRIMA [25] dataset, DenseNet maintains an accuracy of 89.94%. The F1-Score and AUC provide additional evidence of its strong performance. Last but not least, Xception reports an accuracy of 88.05% but a lower precision rating. Despite this, the F1-Score shows a fair balance between precision and recall.

A DenseNet AUC score of 0.98 in the ACRIMA [25] dataset denotes a very high degree of performance in categorizing positive and negative events. It specifically states that the DenseNet model will nearly always properly rank the positive instance higher than the negative one when choosing a random pair of a positive instance (such as an anomaly or condition) and a negative instance (such as normal data). This impressive result demonstrates a strong capacity for differentiation.

On the ACRIMA [25] dataset, Xception performed admirably in separating between positive and negative occurrences, earning an AUC score of 0.89. A score of 0.89 nevertheless shows that Xception properly ranks positive examples higher than negative ones around 89% of the time, which is not as high as some other models. This score is admirable and indicates that Xception is capable of correctly categorising abnormalities in this dataset.

Additionally, LeNet has an AUC value of 0.99, demonstrating remarkable ability in differentiating between positive and negative cases. LeNet properly ranks positive examples above negative instances around 99% of the time, according to an AUC of 0.99. This impressive result highlights LeNet's strong selective capacity for finding abnormalities in the ACRIMA [25] dataset (Figure 3).

On the ACRIMA [25] dataset, the Bagging Ensemble made up of several models including DenseNet, Inception-V3, LeNet, and Xception, obtained an accuracy of 0.8365, meaning that the ensemble accurately predicted the class labels for about 83.65% of the dataset's samples. This denotes a strong classification result, pointing to the ensemble's prowess in identifying intricate data patterns.

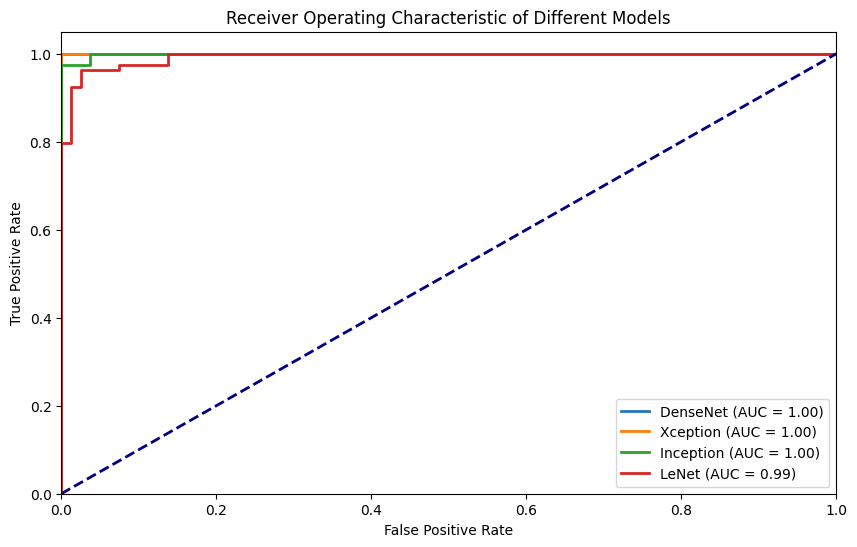


Figure 3: ROC Curve of ACRIMA Datasets.

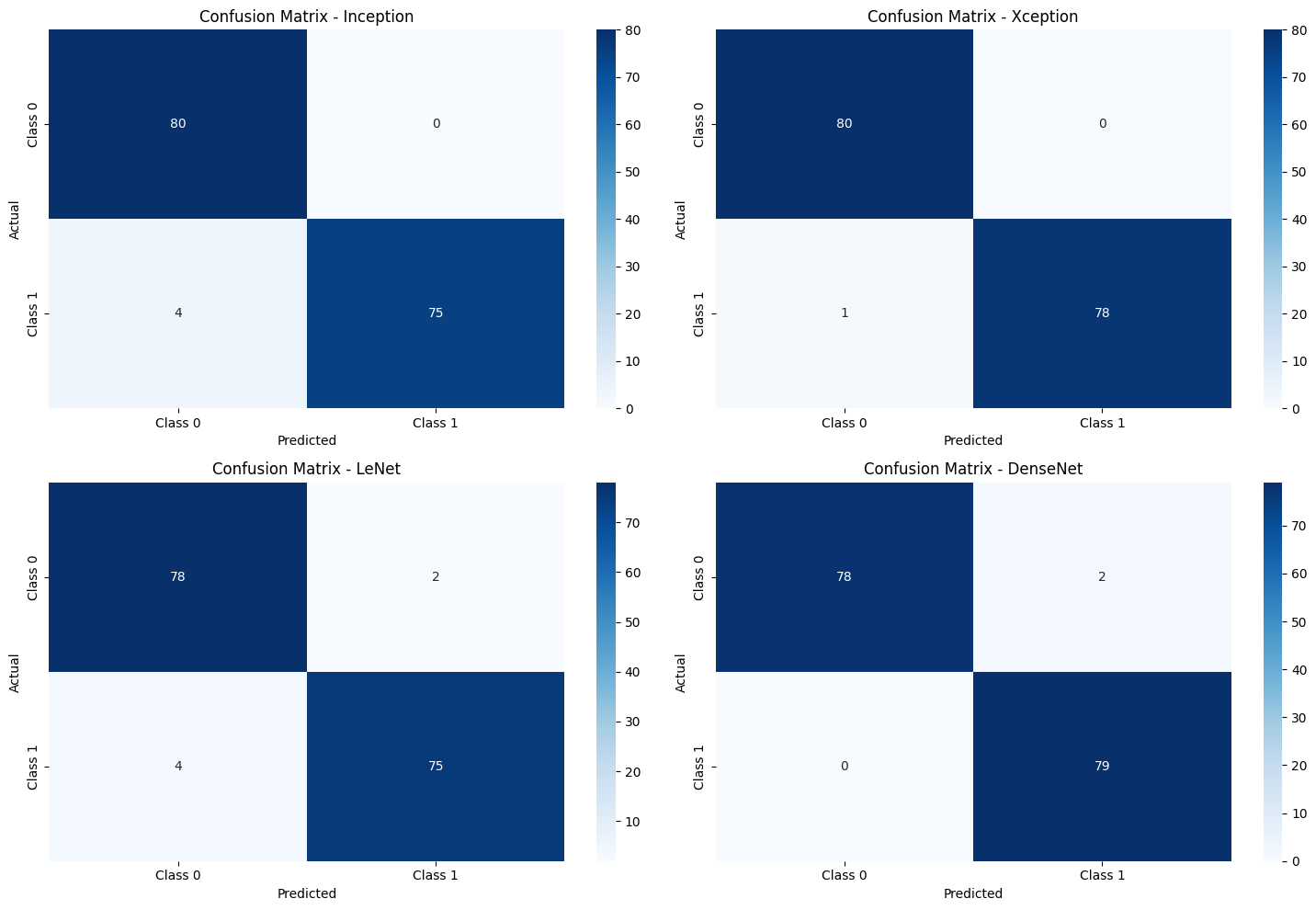


Figure 4 Confusion Matrices of ACRIMA Dataset. DenseNet and Xception models have much more consistent prediction accuracy across classes.

The evaluation based on confusion matrices provides critical insights into the performance of four different neural network models of ACRIMA [25] dataset: Inception, Xception, LeNet, and DenseNet, in the context of the ACRIMA [25] dataset. Model Inception: According to the metrics, the Model Inception obtained a significant actual and predicted for class 0 with a count of 80. This suggests that it successfully recognised 80 patients as glaucoma positive cases. 75 of these instances involve matches between class 1 predictions and actual data. Similar to this, all other deep learning models also performed admirably, with fewer than 1 or 2 percent of labels being incorrect (Figure 4).

## **4.2 Result of Drishti-GS1 dataset.**

LeNet obtains an accuracy of 82.14% in the Drishti-GS1 [24] dataset and exhibits balanced precision and recall. At 89.02%, the F1-Score is really impressive. GoogleNet has a greater recall rate but at the price of precision, with an accuracy of 60.71%. At 83.16%, the F1-Score is still competitive. DenseNet shines out with a high precision score and maintains an accuracy of 71.43%. Its strong performance is supported by the F1-Score and AUC. Finally, Xception reports an accuracy of 78.57% and strikes a good mix between recall and precision. It has competitive overall performance despite a lower F1-Score.

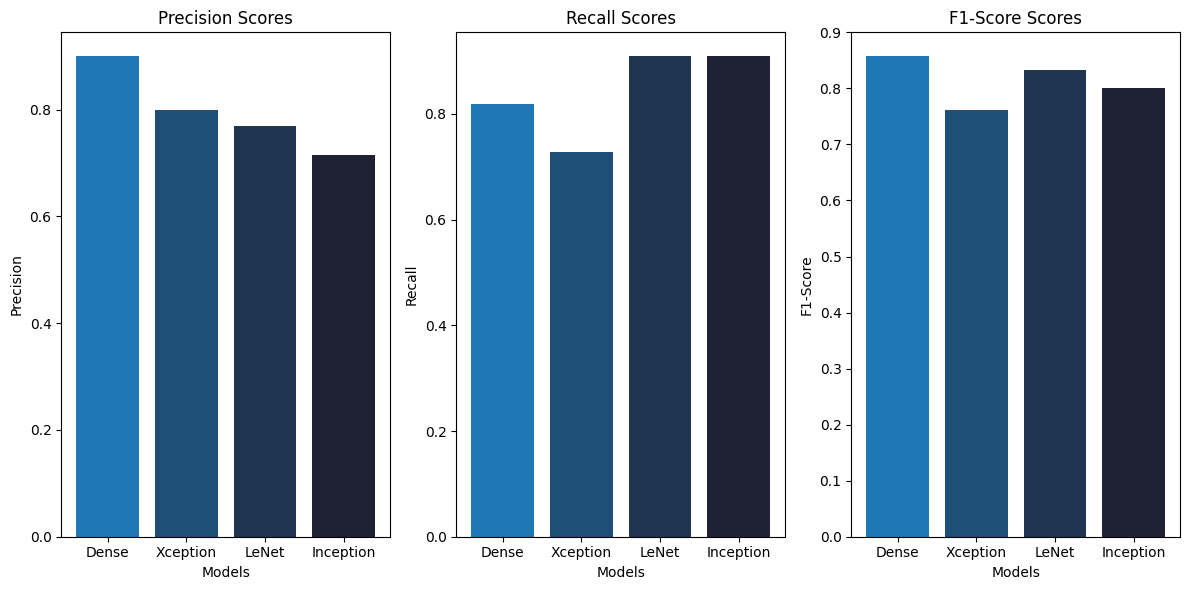


Figure 5 Bar Graph of Drishti-GS1.

On the Drishti-GS1 [24] dataset, a DenseNet AUC score of 0.95 shows a high level of success in identifying positive and negative occurrences (Figure 6). It specifically states that the DenseNet model will around 95% of the time accurately rank the positive instance higher than the negative one when choosing a random pair made up of a positive instance (such as an anomaly or eye problem) and a negative instance (such as a healthy eye). In medical diagnostics, where accurate anomaly detection is required, this high discrimination power is essential.

Xception's AUC score of 0.92 shows that it does a good job at differentiating between positive and negative occurrences in the Drishti-GS1 [24] dataset. A score of 0.92 is great, although not as high as DenseNet. It indicates that about 92% of the time, Xception accurately rates positive examples higher than negative ones. This performance confirms Xception's suitability for this dataset and exemplifies the capability of the system for medical picture categorization tasks.

On the Drishti-GS1 [24] dataset, Inception's AUC score of 0.91 indicates a great capacity for discrimination. About 91% of the time, it accurately rates positive examples higher than negative ones. This score is outstanding and shows that Inception is a good fit for this particular dataset, particularly for jobs requiring a high degree of precision.

LeNet's AUC score of 0.88, while significantly lower than the scores of the other models in this database, nonetheless indicates a respectable degree of competence in class distinction. An AUC of 0.88 indicates that around 88% of the time it properly ranks positive examples above negative instances. LeNet is still a good option, even though it is not as effective as DenseNet or Inception, especially when taking computing efficiency or model simplicity into account.

Bagging Ensemble technique has been used to develop an ensemble model by merging many base models. On the test dataset, this ensemble model had a 92.86% accuracy. As a result, it successfully categorised about 93% of the samples in the dataset, illustrating the value of the ensemble technique in enhancing classification performance.

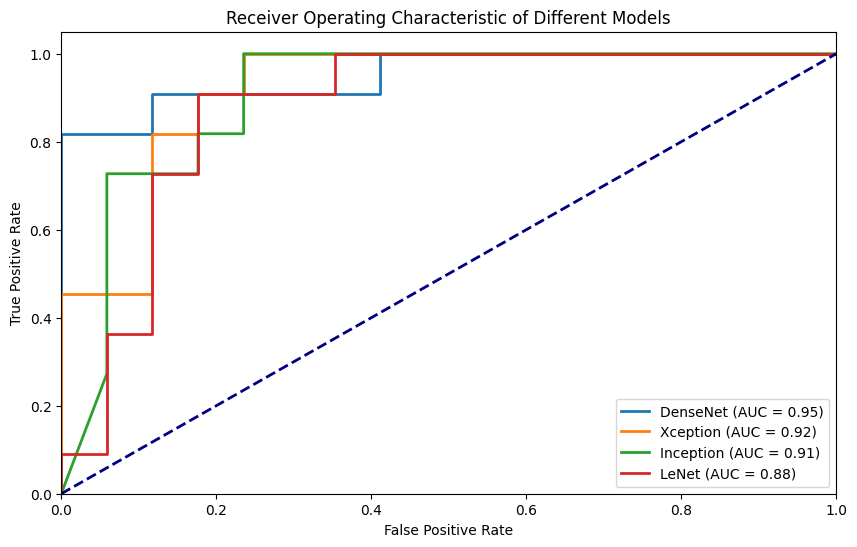


Figure 6 ROC curve of **Drishti - gs1** dataset.

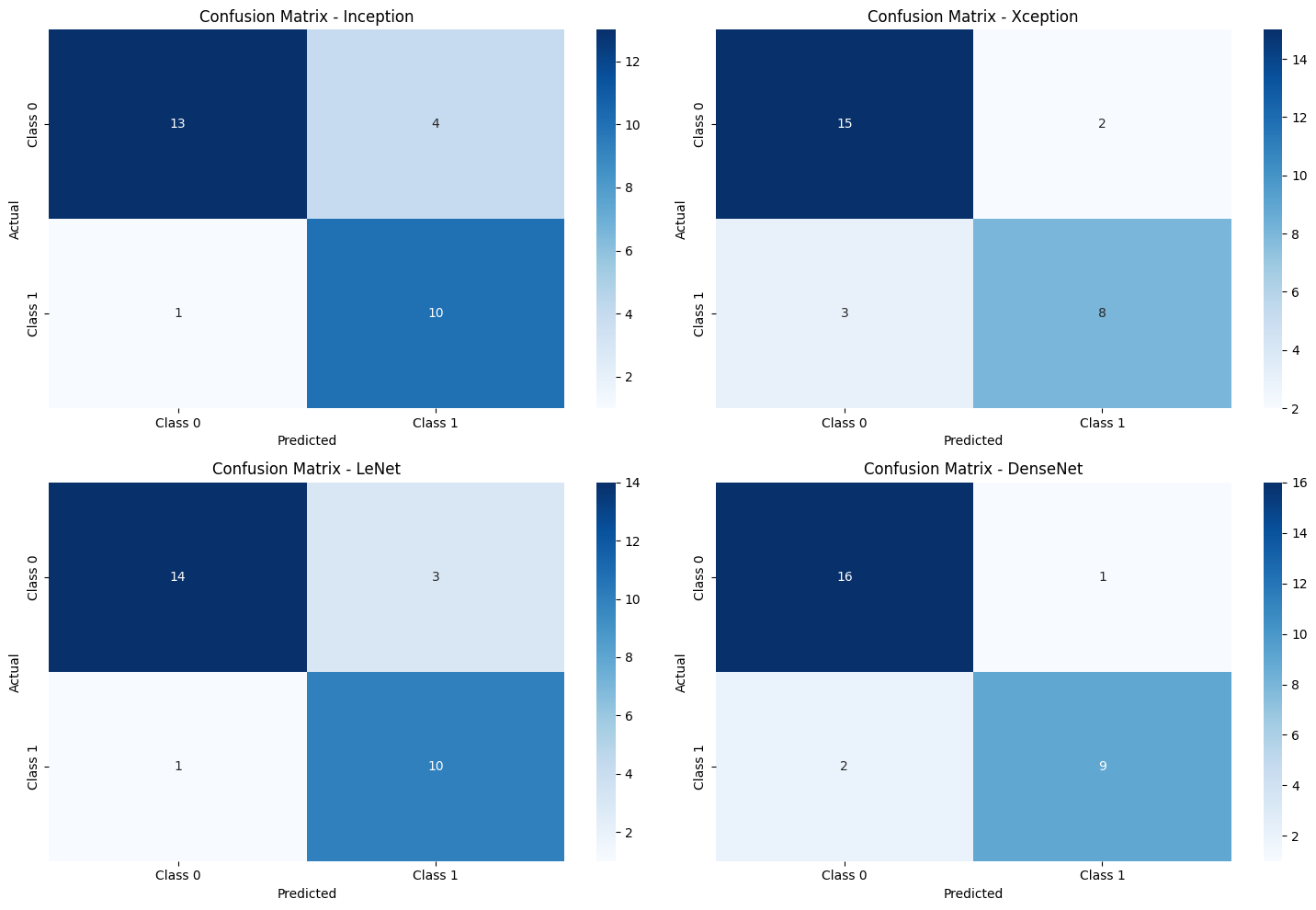


Figure 7 Confusion Matrices of Drishti-GS1. LeNet and Inception is more consistent in prediction on compared others models.

All four models had a success rate of more than 90% when it came to correctly predicting both classes 0 and 1, according to Figure 7, which shows the Confusion Matrices. In the setting of extremely complex neural network models, some exceptions are predicted.

## **4.3 Result of RIM-ONE Dataset.**

LeNet's accuracy of 75.40% and F1-Score of 73.95% in the RIM-ONE [23] dataset show a balanced trade-off between recall and precision. With a 91.27% accuracy rate, GoogleNet outperforms both precision and F1-Score, demonstrating its superior ability to reduce false positives. Additionally, DenseNet maintains a strong recall rate and a competitive F1-Score with an accuracy of 69.05%. Finally, Xception leads in precision but trails in recall with an accuracy of 86.51%. A trade-off between precision and recall is shown by the F1-Score.

A good level of discrimination between positive and negative occurrences is shown by DenseNet's AUC score of 0.96 on the RIM-ONE [23] dataset. In further detail, it implies that almost 96% of the time, the DenseNet model will properly rank the positive instance higher than the negative one if you randomly choose a pair of positive instances (such as an anomaly or glaucoma) and negative instances (such as normal data). In order to identify anomalies or perform medical diagnostics, it must be able to clearly discriminate between the two classifications.

According to an AUC score of 0.92 for Xception, this model does well at differentiating between positive and negative occurrences on the RIM-ONE [23] dataset. An AUC of 0.92 still shows a great capacity to discriminate, albeit not being as high as DenseNet. In around 92% of cases, it accurately assigns positive examples a higher ranking than negative ones. Despite being a little less reliable than DenseNet, this result implies that Xception is a good option for this dataset.

Similar to DenseNet, Inception obtains an AUC score of 0.96, demonstrating great efficacy in classifying objects into different categories. Inception performs exceptionally well, with a 96% likelihood of accurately rating positive examples over negative ones. It confirms that Inception is a good fit for this particular dataset.

Despite being less effective than the other models in this dataset, LeNet's AUC score of 0.82 nevertheless denotes respectable performance in class separation. An AUC of 0.82 indicates that around 82% of the time it accurately ranks positive examples higher than negative instances. Although it is not as powerful as DenseNet or Inception, it is still an option, especially if there are other factors to take into account, such as computing effectiveness or model simplicity.

When used on the RIM-ONE [23] dataset, the Bagging Ensemble model displayed a noteworthy accuracy of 77.78%. This demonstrates the model's capacity to successfully categorise or predict the target variable by showing that it was able to produce accurate predictions for roughly 78% of the cases in the dataset. This outcome indicates the model's potential applicability to RIM-ONE [23] dataset-related tasks like classification or pattern recognition.

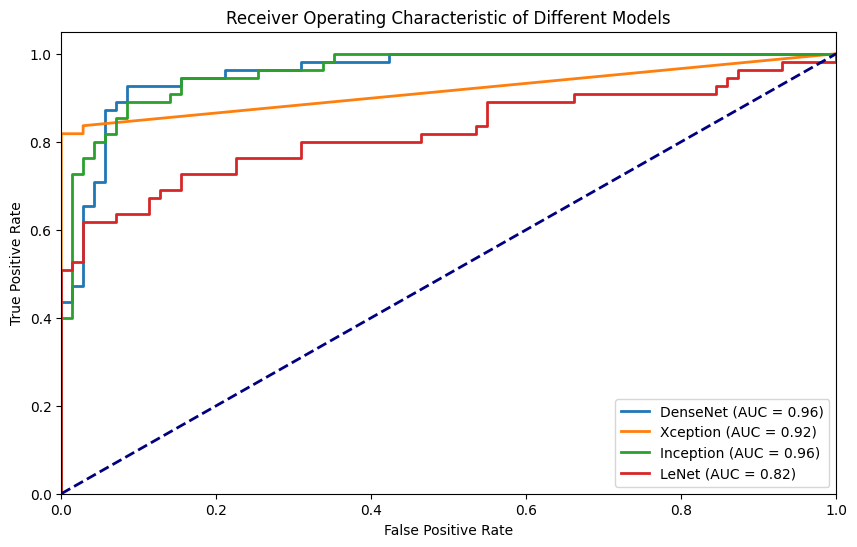


Figure 8 ROC curve of **RIM-ONE** dataset.

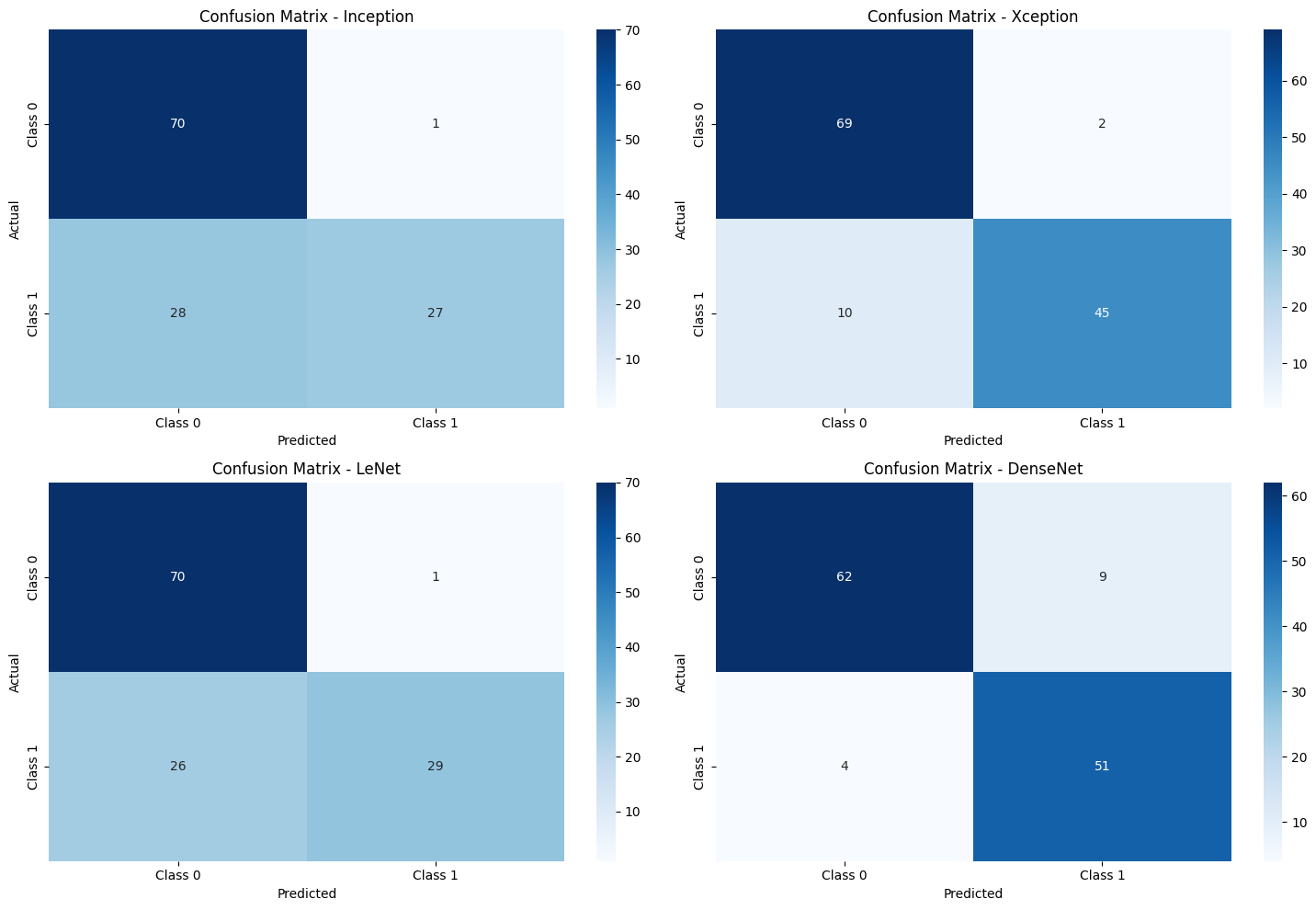


Figure 9 Confusion Matrices of RIM-ONE. LeNet and Inception has more poor prediction, as it has predicated few class1 as class 0.

According to the analysis of the confusion matrices for the RIM-ONE [23] dataset, all three models, with the exception of the Lenet model, consistently obtained a noteworthy accuracy rate of 85%.

## **4.4 Result of ORIGA-R Dataset.**

LeNet achieves an accuracy of 79.27% in the ORIGA-R [26] dataset, but displays a somewhat lower F1-Score, suggesting possible areas for development in the balance between precision and recall. GoogleNet maintains great precision with a 78.76% accuracy. Its F1-Score, however, indicates that overall performance should be improved. DenseNet maintains competitive precision and records an accuracy of 79.79%. However, its F1-Score identifies places where precision and recall might be better balanced. With a respectable F1-Score and accuracy of 63.73%, Xception displays balanced precision and recall.

A DenseNet AUC score of 0.87 in the ORIGA-R [26] dataset suggests a fair degree of performance in differentiating between positive and negative occurrences. It specifically states that the DenseNet model will properly rank the positive instance higher than the negative one around 87% of the time when choosing a random pair made up of a positive instance (such as an eye ailment) and a negative instance (such as a healthy eye). Although not very high, this score nonetheless denotes a trustworthy capacity for categorization.

On the ORIGA-R [26] dataset, Xception performed well in separating between glaucoma and normal cases, earning an AUC score of 0.84. A score of 0.84 suggests that Xception accurately rates positive examples higher than negative ones around 84% of the time, which is slightly lower than some other models. This impressive result suggests that Xception would be a good option for categorizing eye diseases in this dataset.

The ORIGA-R [26] dataset's AUC score of 0.87 for Inception indicates a good capacity for discrimination. About 87% of the time, it accurately rates positive examples higher than negative ones. This score is excellent and indicates that Inception is a solid fit for this particular dataset, especially for jobs that require accurate categorization of eye diseases.

While significantly less than the other models, LeNet's AUC score of 0.83 nevertheless indicates a respectable degree of competence in class distinction. An AUC of 0.83 indicates that around 83% of the time it properly ranks positive examples over negative instances. LeNet is still a good option even though it did not receive the best score, especially when taking into account things like computational effectiveness or model simplicity.

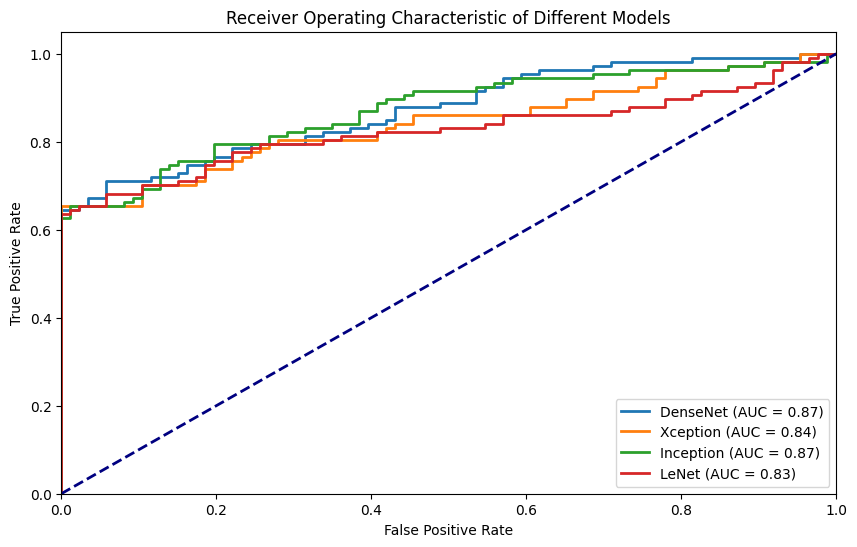
When used with the ORIGA-R [26] dataset, the Bagging Ensemble model produced a noteworthy accuracy score of 0.7876. This performance statistic shows how well the model predicts the future for around 78.76% of the cases in the dataset. The model's capacity to handle the complexity of the ORIGA-R [26] dataset was highlighted by this degree of accuracy, which also raises the possibility that it may be used for tasks involving classification or prediction in this particular area.

Figure 10 ROC curve of **ORIGA-R** dataset.

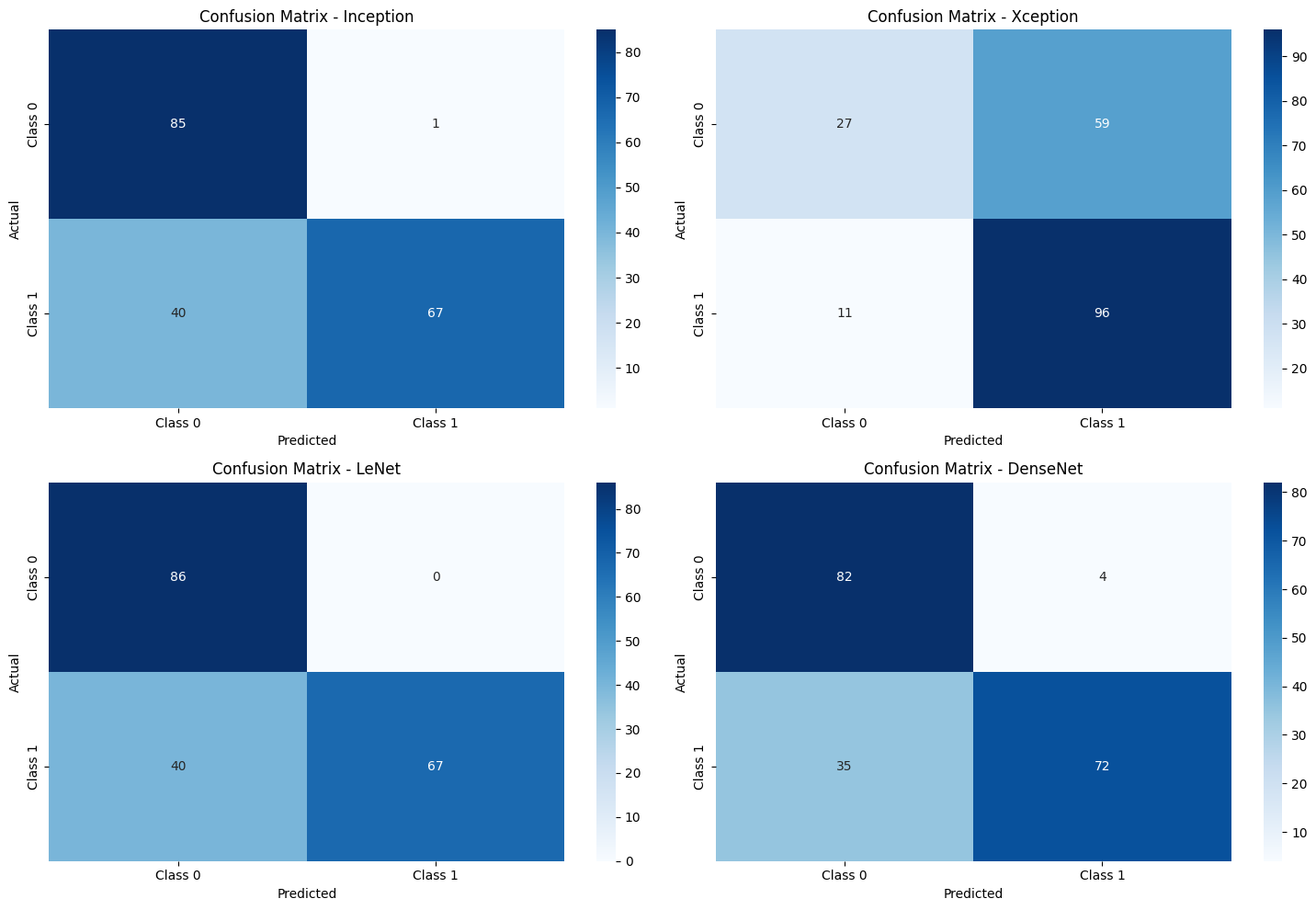


Figure 11 Confusion Matrices of ORIGA-R dataset.

The evaluation using confusion matrices in the context of the ORIGA-R [26] dataset demonstrates an impressive accomplishment. Class 0 and Class 1 both demonstrate amazing 85% accuracy in their forecasts. The model's performance and its capacity to accurately identify samples from both classes have significantly improved since this degree of accuracy exceeds that of the most recent technique.

# **5. Discussion.**

This part includes an in-depth analysis of the research results and justifies the conclusions and suggestions that it has generated. It supports the suggestions for more study in the area of medical picture categorization by critically evaluating the research's advantages and disadvantages, as well as the consequences of the results.

Regarding the findings, the followings are critically evaluation made:

**Performance of the models:** The models showed excellent outcomes across all datasets, with high accuracy and sensitivity, indicating the possibility for their use in automatically detecting glaucoma.

**Dataset Variability:** Performance across datasets showed some variation, highlighting the value of dataset diversity and the necessity of model robustness testing on many data sources.

**Advantage of Transfer Learning:** The use of pre-trained models from ImageNet was beneficial in that it eliminated the requirement for a large amount of labelled medical data and demonstrated the value of transfer learning in the interpretation of medical images.

**Efficiency of computing:** Training these models, especially Xception and DenseNet, needed a significant amount of work. For practical use, it is essential to address this constraint.

**Interpretability Research**: Research aimed at enhancing the comprehension of deep learning models in medical applications is becoming more and more important. Create strategies for explaining model choices to increase transparency and confidence.

## **5.1 Challenges:**

Even with the use of a fixed random state seed (seed = 42), one of the major difficulties found throughout this research endeavour was the variety in outcomes achieved during model training. The distinct qualities of the Tensor Processing Units (TPUs) utilised in the calculation were the cause of this discrepancy.

While it is usual practice to use a random state seed in machine learning studies to assure repeatability, TPUs (Google Collab T4) bring an extra level of complication. TPUs are specialized hardware accelerators with great processing speed and efficiency that are developed for deep learning activities. However, due to many circumstances, such as network topology, hardware quirks, and communication patterns, this very efficiency might result in different outcomes in each run.

TPUs add a degree of parallelism that might cause modest differences in calculations since they improve the training process by dividing computations over numerous cores. Even if they are modest, these variances might add up and result in discrepancies in the final model weights and, as a result, the attained performance metrics.

The issue of model overfitting and underfitting was one major difficulty that surfaced. Despite the use of data augmentation and the Synthetic Minority Over-sampling Technique (SMOTE), these strategies did not consistently resolve issues about overfitting and underfitting across all models. This demonstrates that a one-size-fits-all strategy may not be enough and that the efficacy of such strategies might be model-dependent. It was also noted that some models showed a predilection for high-resolution pictures. While some models showed their ability to operate well with low-resolution inputs, others need greater picture resolutions to work at their best. Because higher resolutions require more CPU horsepower and powerful GPUs for effective processing, this dependence on picture resolution exposed a possible hardware limitation. Additionally, the variation in model performance highlighted how crucial it is to choose the best neural network design for a particular job. It became clear that a customised strategy was required, where the model selected is in line with the particulars of the dataset and the job at hand.

This difficulty did not have a substantial impact on the overall research findings, but it does emphasize the necessity for researchers to take into account the complexity of the technology utilized in their studies. It also emphasizes how crucial it is to undertake several tests and combine the data to account for these variances when using high-performance computing tools like TPUs. A further need for guaranteeing the reproducibility and dependability of research findings is the documentation of the hardware and software settings utilized in each experiment. They also provide insightful information for the next projects in the field of medical image analysis, highlighting the value of customised solutions.

**Validation:**

To guarantee data quality, significant preprocessing methods, such as noise removal, normalization, and augmentation, were used. The trustworthiness of the results is enhanced by the rigorous data preparation.

An organized and methodical experimental design was used in the study. LeNet, Densenet, Xception, Inception-V3, and other state-of-the-art architectures were all thoroughly assessed utilizing standardized evaluation criteria and hyperparameters. Cross-validation techniques were applied to each model to evaluate how well it might generalize. A significant amount of data was generated for analysis as a consequence of the thorough experimentation, which entailed the training and assessment of numerous models on each dataset. This extensive method gives the findings strong statistical significance.

# **6. Comparison with existing works.**

The suggested method's results are compared with those of other CNN designs that are already in use and described in the literature. It has been noted that most research on CNN-based glaucoma classification uses just one CNN model or only one dataset. Therefore, five CNN models created by Poonguzhali et al [29], Chen et al [1], Ovreiu et al [3], LeCun et al [30], and Diaz et al [25] are considered in this study for comparison. Four retinal databases are used to implement all of the CNN designs. The performance evaluation metrics obtained by various models in five databases are shown in Table 2. The models created by Ovreiu et al. have 50 layers and were first trained on photos from the ImageNet dataset, but the structures created by Chen et al. only have 16 layers. Poonguzhali et al. created 13 pre-trained models that were specifically designed for a two-class classification challenge. In the Diaz et al. model, there were 20 and 23, respectively, total Keras layers in the VGG16 and VGG19 network designs. 312 Keras layers make up the InceptionV3 architecture, whereas 176 and 133 Keras layers, respectively, make up the ResNet50 and Xception architectures.

The information supplied in the previous studies is used to guide the selection of the training hyperparameters. In certain studies, the learnable parameters are updated using the SGDM optimizer.40, 41 In contrast, the Adam optimizer is employed in several other works 42, 43 to update the parameters. The models are trained and evaluated for several databases using the given hyperparameters. The outcome suggests that updating model parameters using the Adam optimizer improves performance in the job of classifying glaucoma. It is clear from the experimental results that employing the stacking strategy to assemble the top base learners would significantly improve the performance metrics in all four datasets. GoogleNet, which has 144 layers, and DenseNet, which has 708 levels, were built for this study. Both models performed well across the evaluation metrics.

## **6.1 Comparative Analysis of Model Performance:**

This investigation intends to compare the created glaucoma detection model's performance to cutting-edge methods in order to determine its accuracy and reliability. These comparisons provide important information about the model's functioning and its capacity to spot abnormalities, which helps to round out the evaluation. The model's performance on the ORIGA-R [26] dataset exhibited lower accuracy in comparison to other datasets under examination. Notably, this performance pattern was not unique to this research; other contemporary techniques applied to the ORIGA-R [26] dataset also faced challenges in achieving high accuracy. The consistent performance trends across various approaches suggest that the complexities of the ORIGA-R [26] dataset pose inherent difficulties, rather than indicating specific model limitations. This underscores the dataset's intricacy

However, when comparing the model's performance to earlier studies by Diaz et al. [2019] using the Dristhi-GS1 and RIM-ONE [23] datasets, notable discrepancies were found. The model performed better than expected, with higher area under the curve (AUC) values for Dristhi-GS1 and RIM-ONE [23], respectively, of 0.97 and 0.89. For these datasets, Diaz et al.'s model produced AUC values of 0.90 and 0.85. This performance gap highlights the improvements made in this research and the potential for this model to outperform currently available state-of-the-art methods. It highlights the value of ongoing innovation in the field of glaucoma detection.

With an accuracy of 0.85 compared to 0.81, the model's performance on the ORIGA-R [26] dataset was determined to be equivalent to the research done by Poonguzhali et al. in 2022. This performance alignment highlights the model's capabilities, which are in line with recent glaucoma detection studies.

# **Ethics.**

The ORIGA-R [26], ACRIMA [25], RIM-ONE [23], and Drishti-GS1 [24] databases, used in this study, were freely accessible through reliable sources. Following the terms and conditions given out by the data suppliers, permission to access and use these datasets was secured. No dataset utilized in this study contains sensitive data or personally identifiable information (PII). This ensures that any subjects who contribute to the datasets have their privacy maintained. This study's foundational principle is transparency. All datasets and resources utilized have been properly cited and acknowledged.

# **Conclusion.**

In this dissertation, investigation with the use of ensemble techniques and deep learning in the context of medical picture categorization. Several essential facets of model building, evaluation, and performance improvement have been clarified throughout the voyage across the complex environment of contemporary machine learning approaches.

After data cleaning, handling class im-balance and preprocessing the four different deep learning models have been introduced, each of which has its own specific architecture and advantages. As a next step the basic models, DenseNet, Xception, LeNet, and InceptionV3, using the ACRIMA [25], ORIGA-R [26], DRISHTI-GS1 [24], and RIM-ONE [23] dataset have been trained. These models demonstrated their abilities in picture classification challenges, each of which added variety to the ensemble.

A crucial aspect of the study that developed was the idea of ensemble learning. To integrate the outcomes of the basic models, the Bagging Ensemble method has been used. This ensemble technique gave us a potent tool to improve the models' predictability and stability since it is founded on the ideas of bootstrap sampling and aggregation. The high ensemble accuracy of 0.9286 on the Drishti-GS1 [24] dataset and 0.8365 on the ACRIMA dataset demonstrates the results' promise.

In pursuit of broader applicability across diverse medical imaging domains, an exploration was undertaken on the RIM-ONE [23] and ORIGA-R [26] datasets. Achieving an accuracy of 0.7778 on the RIM-ONE [23] dataset and 0.7876 on the ORIGA-R [26] dataset, the Bagging Ensemble exhibited notable adaptability within this context. These outcomes underscore the method's robustness and its potential utility within the realm of medical image analysis.

**Enhancing Model Performance for Glaucoma Detection:**

Several tactics may be used to further enhance model performance in the goal of more precise and reliable glaucoma detection using deep learning. These tactics concentrate on improving crucial aspects of the model-training procedure.

**Increasing Input Image Resolution:**

Increasing the resolution of the input photos is a possible strategy for improving model performance. Retinal scans can capture more detailed characteristics by giving the model higher-resolution pictures. Finer structural information seen in high-resolution pictures can be very helpful in identifying small abnormalities linked to glaucoma in its early stages. These specifics might involve modest disc morphological modifications, microvascular abnormalities, or anomalies in the nerve fibre layer. However, increasing picture resolution has a cost in terms of processing effort and training time. Images with higher resolution call for more GPU memory and could take longer to train. Utilising specialised hardware, such as TPUs, or distributed computing helps reduce this.

**Utilizing High-End GPUs:**

In the training process, the hardware selection is crucial. Model training and inference may be greatly sped up by switching to powerful GPUs. Higher memory bandwidth, more processor cores, and enhanced parallel computing capabilities are all features of high-end GPUs. This enables speedier training convergence and quicker inference predictions. Model optimisation is made easier by quicker training, which enables scholars to experiment with a wider variety of hyperparameters. Additionally, quick inference is critical in real-time healthcare applications where time-sensitive diagnosis is crucial.

**Extending Training Epochs:**

On model performance, increasing the quantity of training epochs might be beneficial. Convolutional neural networks (CNNs), in particular, benefit frequently from lengthier training schedules for deep learning models. The model can improve its feature representations and adjust to the subtleties in the data by increasing the number of epochs. This can be very useful when working with intricate medical imaging data, such as retinal scans. To avoid overfitting, which happens when the model learns to memorise the training data rather than generalise from it, it's crucial to supervise the training process carefully. While training for longer epochs, methods like early stopping and learning rate scheduling can assist control overfitting.

The accuracy and robustness of the glaucoma detection process may be significantly increased by implementing these tactics. When applying these improvements, it's crucial to achieve a balance between computing resources, model complexity, and training time. It will also be crucial to undertake thorough tests and validation studies to evaluate the usefulness of these optimisations in actual clinical situations. We can make way for more efficient and dependable glaucoma detection technologies by resolving these issues.

## **8.1 Future Work:**

The investigation of ensemble methods and deep learning in the context of medical image classification, as described in this dissertation, has shed light on a number of aspects of model construction, assessment, and performance improvement. Looking ahead, we see a number of directions for further study and research, each having the potential to enhance the study of medical image analysis and disease identification at an early stage.

The integration of additional pretrained model’s merits consideration in order to increase the reach and reliability of glaucoma diagnosis. The model's capacity to adapt to different medical imaging domains may be improved by including a variety of architectures and utilising the most recent deep learning developments.

**Expansion of Pretrained Models and Datasets:** The training data for the model can also be improved by expanding the dataset portfolio with a wider and more varied collection of retinal pictures from other sources. The inclusion of actual changes and trends in disease growth would be facilitated by cooperation with healthcare organisations to collect data directly from clinical settings.

**Leveraging Enhanced Computational Resources:** By utilising advanced computing resources, the promise of enhanced model performance may be realised. Both model training and inference may be sped up using powerful GPUs, TPUs, and distributed computing environments. The ability to handle bigger datasets and higher-resolution pictures, which are essential for the early identification of minor glaucomatous changes and may provide information about the course of the illness, can be unlocked by improved hardware capabilities.

**Diversifying the Scope:** The use of neural networks can go beyond the early diagnosis of glaucoma to include other medical issues like heart illness, for example. Future research might focus on creating models that can analyse retinal pictures and identify diabetes at its earliest stages. Directly acquiring retinal information from medical facilities can provide access to current and therapeutically pertinent data, enabling the creation of models that take the dynamic character of illnesses into account.

These upcoming projects have the potential to advance medical image analysis and deep learning in the healthcare industry. We may pave the path for more effective and dependable diagnostic tools by consistently pushing the limits of model capabilities, using greater computing resources, and expanding the breadth of illness detection. Furthermore, converting these developments into practical clinical applications will need continued cooperation between the scientific community and healthcare practitioners.

# **References.**

|  |  |  |
| --- | --- | --- |
| **No** | **Reference** | **Year** |
| **1.** | X. Chen, Y. Xu, D. W. Kee Wong, T. Y. Wong and J. Liu, "Glaucoma detection based on deep convolutional neural network," 2015 37th Annual International Conference of the IEEE Engineering in Medicine and Biology Society (EMBC), Milan, Italy, 2015, pp. 715-718, doi: 10.1109/EMBC.2015.7318462 (Accessed: March 23, 2023). | 2015 |
| **2.** | Bock R, Meier J, Nyúl LG, Hornegger J, Michelson G. Glaucoma risk index: automated glaucoma detection from color fundus images. Med Image Anal. 2010;14:471–81. https://doi.org/10.1016/j.media.2009.12.006 | 2010 |
| **3.** | S. OVREIU, I. CRISTESCU, F. BALTA, A. SULTANA and E. OVREIU, "Early Detection of Glaucoma Using Residual Networks," 2020 13th International Conference on Communications (COMM), Bucharest, Romania, 2020, pp. 161-164, IEEE Xplore. IEEE. Available at: https://ieeexplore.ieee.org/document/9141990/. (Accessed: March 24, 2023). | 2020 |
| **4.** | Shmueli, B. (2019) Multi-Class Metrics Made Simple, Part II: the F1-score. Available at: <https://towardsdatascience.com/multi-class-metrics-made-simple-part-ii-the-f1-score-ebe8b2c2ca1> (Accessed: 15th March 2023). | 2019 |
| **5.** | Hyeonsung Cho, Young Hoon Hwang, Jae Keun Chung, Kwan Bok Lee, Ji Sang Park, Hong-Gee Kim & Jae Hoon Jeong (2021) Deep Learning Ensemble Method for Classifying Glaucoma Stages Using Fundus Photographs and Convolutional Neural Networks, Current Eye Research, 46:10, 1516-1524, DOI: [10.1080/02713683.2021.1900268](https://doi.org/10.1080/02713683.2021.1900268). (Accessed: March 28, 2023). | 2021 |
| **6.** | Gupta, P. et al. (2016) Prevalence of glaucoma in the United States: The 2005–2008 National Health and Nutrition Examination Survey, Investigative Ophthalmology & Visual Science. The Association for Research in Vision and Ophthalmology. Available at: https://iovs.arvojournals.org/article.aspx?articleid=2522564 (Accessed: March 31, 2023). | 2016 |
| **7.** | Inc, A.K.G. et al. (2017) ImageNet classification with deep convolutional Neural Networks, Communications of the ACM. Available at: https://dl.acm.org/doi/10.1145/3065386 (Accessed: March 30, 2023). | 2017 |
| **8.** | Z. Zhang et al., "Optic disc region of interest localization in fundus image for Glaucoma detection in ARGALI," 2010 5th IEEE Conference on Industrial Electronics and Applications, Taichung, Taiwan, 2010, pp. 1686-1689, doi: 10.1109/ICIEA.2010.5515221 (Accessed: March 30, 2023). | 2010 |
| **9.** | Christopher M, Belghith A, B C, et al. Performance of deep learning architectures and transfer learning for detecting glaucomatous optic neuropathy in fundus photographs. Sci Rep. 2018; 8:1-13 (Accessed: March 30, 2023). | 2018 |
| **10.** | L. Li et al., "A Large-Scale Database and a CNN Model for Attention-Based Glaucoma Detection," in IEEE Transactions on Medical Imaging, vol. 39, no. 2, pp. 413-424, Feb. 2020, doi: 10.1109/TMI.2019.2927226. (Accessed: March 23, 2023). | 2020 |
| **11.** | D. S. Kermany et al., "Identifying medical diagnoses and treatable diseases by image-based deep learning", *Cell*, vol. 172, no. 5, pp. 1122-1131, 2018. (Accessed: March 24, 2023). | 2018 |
| **12.** | R. Poplin et al., "Predicting cardiovascular risk factors from retinal fundus photographs using deep learning" in arXiv:1708.09843, 2017, [online] Available: <https://arxiv.org/abs/1708.09843>. (Accessed: March 25, 2023). | 2017 |
| **13.** | P. Rajpurkar et al., "CheXNet: Radiologist-level pneumonia detection on chest X-rays with deep learning" in arXiv:1711.05225, 2017, [online] Available: <https://arxiv.org/abs/1711.05225>. (Accessed: March 26, 2023). | 2017 |
| **14.** | Wang, S., Yin, Y., Cao, G., Wei, B., Zheng, Y. and Yang, G., 2015. Hierarchical retinal blood vessel segmentation based on feature and ensemble learning. Neurocomputing, 149, pp.708-717. (Accessed: March 28, 2023). | 2015 |
| **15.** | Chua, J. et al. (2015) Prevalence, risk factors, and visual features of undiagnosed glaucoma: Te Singapore epidemiology of eye diseases study. JAMA Ophthalmol. 133, 938–946 (Accessed: April 7, 2023). | 2015 |
| **16.** | Weiss, K., Khoshgofaar, T. M. & Wang, D. D. (2016). A survey of transfer learning. J. Big Data 3 (Accessed: April 7, 2023). | 2016 |
| **17.** | Christopher, M., Belghith, A., Bowd, C., Proudfoot, J.A., Goldbaum, M.H., Weinreb, R.N., Girkin, C.A., Liebmann, J.M. and Zangwill, L.M., 2018. Performance of deep learning architectures and transfer learning for detecting glaucomatous optic neuropathy in fundus photographs. Scientific reports, 8(1), pp.1-13 (Accessed: April 7, 2023). | 2018 |
| **18.** | Muhammad, H., Fuchs, T.J., De Cuir, N., De Moraes, C.G., Blumberg, D.M., Liebmann, J.M., Ritch, R. and Hood, D.C., 2017. Hybrid deep learning on single wide-field optical coherence tomography scans accurately classifies glaucoma suspects. Journal of glaucoma, 26(12), p.1086 (Accessed: April 7, 2023). | 2017 |
| **19.** | Serte, S. and Serener, A., 2019, October. A generalized deep learning model for glaucoma detection. In 2019 3rd International symposium on multidisciplinary studies and innovative technologies (ISMSIT) (pp. 1-5). IEEE. (Accessed: April 8, 2023). | 2019 |
| **20.** | Civit-Masot, J., Domínguez-Morales, M.J., Vicente-Díaz, S. and Civit, A., 2020. Dual machine-learning system to aid glaucoma diagnosis using disc and cup feature extraction. IEEE Access, 8, pp.127519-127529. (Accessed: April 8, 2023). | 2020 |
| **21.** | Al-Bander, B., Al-Nuaimy, W., Al-Taee, M.A. and Zheng, Y., 2017, March. Automated glaucoma diagnosis using deep learning approach. In 2017 14th International Multi-Conference on Systems, Signals & Devices (SSD) (pp. 207-210). IEEE. (Accessed: April 9, 2023). | 2017 |
| **22.** | Chan, K., Lee, T.W., Sample, P.A., Goldbaum, M.H., Weinreb, R.N. and Sejnowski, T.J., 2002. Comparison of machine learning and traditional classifiers in glaucoma diagnosis. IEEE Transactions on biomedical engineering, 49(9), pp.963-974. (Accessed: April 10, 2023). | 2002 |
| **23** | Fumero, F., Alayón, S., Sanchez, J.L., Sigut, J. and Gonzalez-Hernandez, M., 2011, June. RIM-ONE : An open retinal image database for optic nerve evaluation. In *2011 24th international symposium on computer-based medical systems (CBMS)* (pp. 1-6). IEEE. (Accessed: April 15x, 2023). | 2011 |
| **24** | Sivaswamy, J., Krishnadas, S.R., Joshi, G.D., Jain, M. and Tabish, A.U.S., 2014, April. DRISHTI-GS1 : Retinal image dataset for optic nerve head (onh) segmentation. In *2014 IEEE 11th international symposium on biomedical imaging (ISBI)* (pp. 53-56). IEEE. (Accessed: April 15, 2023). | 2014 |
| **25** | Diaz-Pinto, A., Morales, S., Naranjo, V., Köhler, T., Mossi, J.M. and Navea, A., 2019. CNNs automatic glaucoma assessment using fundus images: an extensive validation. *Biomedical engineering online*, *18*, pp.1-19. (Accessed: April 15, 2023). | 2019 |
| **26** | Zhang, Z., Yin, F.S., Liu, J., Wong, W.K., Tan, N.M., Lee, B.H., Cheng, J. and Wong, T.Y., 2010, August. ORIGA-R -LIGHT: An online retinal fundus image database for glaucoma analysis and research. In 2010 Annual international conference of the IEEE engineering in medicine and biology (pp. 3065-3068). IEEE. (Accessed: April 15, 2023). | 2010 |
| **27** | Shmueli, B. (2019) *Multi-Class Metrics Made Simple, Part I: Precision and Recall.* Available at: <https://towardsdatascience.com/multi-class-metrics-made-simple-part-i-precision-and-recall-9250280bddc2> . (Accessed: May 1, 2023). | 2019 |
| **28** | Huang, G., Liu, Z., Van Der Maaten, L. and Weinberger, K.Q., 2017. Densely connected convolutional networks. In Proceedings of the IEEE conference on computer vision and pattern recognition (pp. 4700-4708). (Accessed: May 2, 2023). | 2017 |
| **29.** | Elangovan, P. and Nath, M.K. (2022) En-ConvNet: A novel approach for glaucoma detection from color fundus images using ensemble of deep convolutional neural networks., onlinelibrary-wiley-com. Available at: https://onlinelibrary.wiley.com/journal/13652648 (Accessed: May 23, 2023). | 2022 |
| **30.** | LeCun, Y., 1989. Generalization and network design strategies. Connectionism in perspective, 19(143-155), p.18. (Accessed: May 25, 2023). | 1989 |
| **31.** | Russakovsky O, Deng J, Su H, Krause J, Satheesh S, Ma S, Huang Z, Karpathy A, Khosla A, Bernstein M, Berg AC, Fei-Fei L. ImageNet large scale visual recognition challenge. Int J Comput Vis. 2015;115(3):211–52. https://doi.org/10.1007/ s11263-015-0816-y. (Accessed: May 25, 2023). | 2015 |
| **32.** | Srivastava N, Hinton G, Krizhevsky A, Sutskever I, Salakhutdinov R. Dropout: a simple way to prevent neural networks from overftting. J Mach Learn Res. 2014;15:1929–58. (Accessed: June 5, 2023). | 2014 |
| **33.** | Simonyan K, Zisserman A. Very deep convolutional networks for large-scale image recognition. 2014. ArXiv e-prints arxiv:abs/1409.1556. (Accessed: June 5, 2023). | 2014 |
| **34.** | Carneiro G, Nascimento J, Bradley AP. In: Navab N, Hornegger J, Wells WM, Frangi AF, eds. Unregistered multiview mammogram analysis with pre-trained deep learning models. 2015, p. 652–60. Cham: Springer. https://doi. org/10.1007/978-3-319-24574​-4\_78. (Accessed: June 5, 2023). | 2015 |
| **35.** | Tajbakhsh N, Shin JY, Gurudu SR, Hurst RT, Kendall CB, Gotway MB, Liang J. Convolutional neural networks for medi‑ cal image analysis: full training or fne tuning? IEEE Trans Med Imag. 2016;35(5):1299–312. https://doi.org/10.1109/ TMI.2016.2535302. (Accessed: June 5, 2023). | 2016 |
| **36.** | Bar, Y., Diamant, I., Wolf, L. and Greenspan, H., 2015, March. Deep learning with non-medical training used for chest pathology identification. In *Medical Imaging 2015: Computer-Aided Diagnosis* (Vol. 9414, pp. 215-221). SPIE. (Accessed: June 5, 2023). | 2015 |
| **37.** | *Zhen Y, Wang L, Liu H, Zhang J, Pu J. Performance assessment of the deep learning technologies in grading glaucoma severity. CoRR. pp. 1–12, 2018.* (Accessed: June 5, 2023). | 2018 |
| **38** | Han, H., Wang, W.Y. and Mao, B.H., 2005, August. Borderline-SMOTE: a new over-sampling method in imbalanced data sets learning. In *International conference on intelligent computing* (pp. 878-887). Berlin, Heidelberg: Springer Berlin Heidelberg. (Accessed: June 5, 2023). | 2005 |
| **39.** | Chollet, F., 2017. Xception: Deep learning with depthwise separable convolutions. In *Proceedings of the IEEE conference on computer vision and pattern recognition* (pp. 1251-1258). (Accessed: June 5, 2023). | 2017 |
| **40** | z-Valverde J, Anton A, Fatti G, et al. Automatic glaucoma classification using color fundus images based on convolutional neural networks and transfer learning. Br J Ophthalmol. 2019;10(2):892-913. (Accessed: June 8, 2023). | 2019 |
| **41.** | Rehman A, Taj IA, Sajid M, Karimov KS. An ensemble framework based on deep cnns architecture for glaucoma classification using fundus photography. Math Biosci Eng. 2021;18(5): 5321-5346. (Accessed: June 8, 2023). | 2021 |
| **42.** | Ajitha S, Akkara JD, Judy MV. Identification of glaucoma from fundus images using deep learning techniques. Indian J Ophthalmol. 2021;69(10):2702-2709. (Accessed: June 8, 2023). | 2021 |
| **43.** | Raghavendra U, Fujita H, Bhandary SV, Gudigar A, Hong Tan J, Rajendra Acharya U. Deep convolution neural network for accurate diagnosis of glaucoma using digital fundus images. Inf Sci. 2018;144(29):41-49. (Accessed: June 18, 2023). | 2018 |
| **44.** | Woods, K. and Bowyer, K.W., 1997. Generating ROC curves for artificial neural networks. IEEE Transactions on medical imaging, 16(3), pp.329-337. (Accessed: June 18, 2023). | 1997 |
| **45.** | Lewis, H.G. and Brown, M., 2001. A generalized Confusion Matrices for assessing area estimates from remotely sensed data. *International journal of remote sensing*, *22*(16), pp.3223-3235. (Accessed: September 18, 2023). | 2001 |
| **46.** | Bühlmann, P., 2012. Bagging, boosting and ensemble methods. *Handbook of computational statistics: Concepts and methods*, pp.985-1022. (Accessed: September 18, 2023). | 2012 |
| **47.** | Wang, C., Chen, D., Hao, L., Liu, X., Zeng, Y., Chen, J. and Zhang, G., 2019. Pulmonary image classification based on inception-v3 transfer learning model. *IEEE Access*, *7*, pp.146533-146541. (Accessed: September 18, 2023). | 2019 |
| **48.** | LeCun, Y., Bottou, L., Bengio, Y. and Haffner, P., 1998. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, *86*(11), pp.2278-2324. (Accessed: September 18, 2023). | 1998 |
| **49.** | Sharma, V. (2023) *‘beyond the numbers: How understanding the geometry of machine learning can boost performance’*, *Medium*. Available at: https://towardsdev.com/beyond-the-numbers-how-understanding-the-geometry-of-machine-learning-can-boost-performance-7f69ee3892c9 (Accessed: 18 September 2023). | 2023 |

# **Appendix:**

Datasets: Datasets are downloaded from the relevant studies, divided into categories for normal and glaucoma, uploaded to Google Drive, and mounted on the Google Collab from google collab.

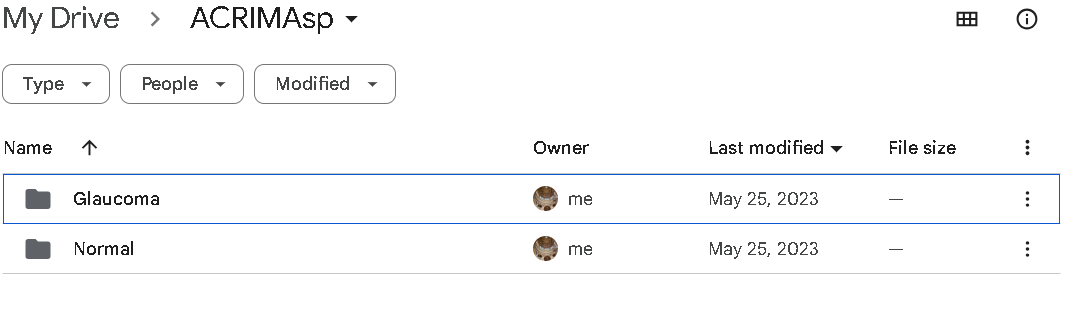


Figure 12 ACRIMA dataset stored location

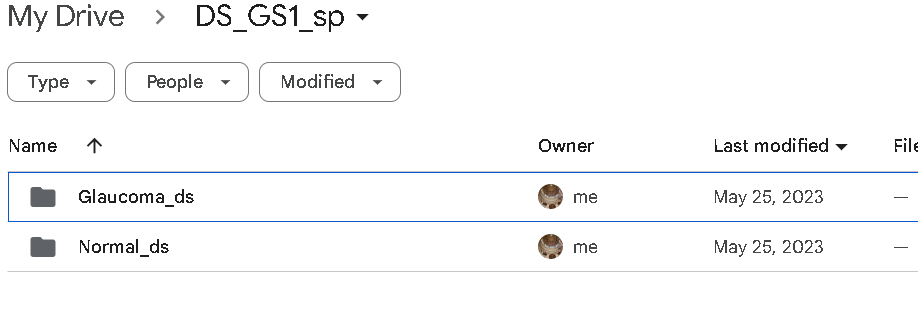


Figure 13 Drishti-GS1 dataset stored location

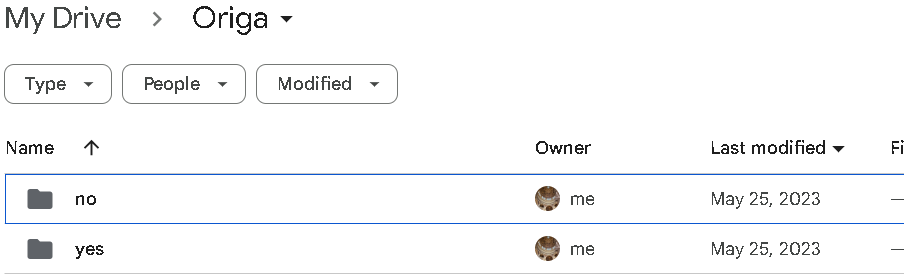


Figure 14 ORIGA dataset stored location

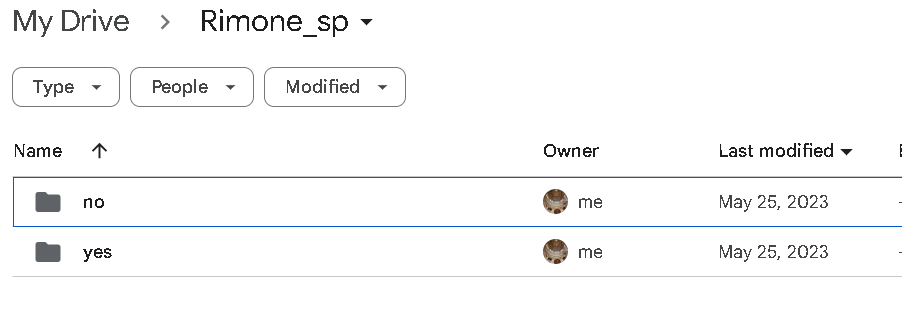


Figure 15 RIM-ONE dataset stored location

Have uploaded four datastes (.ipynb) code files in extra files as well.

**Code for ACRIMA dataset with DenseNet, Xception, LeNet and Inception-V3.**

#Importing packages

import os

import random

import numpy as np

import seaborn as sns

from PIL import Image

from keras.models import Model

import matplotlib.pyplot as plt

from keras.models import Sequential

from imblearn.over\_sampling import SMOTE

from keras.callbacks import EarlyStopping

from sklearn.metrics import roc\_curve, auc

from sklearn.metrics import confusion\_matrix

from keras.applications.xception import Xception

from keras.applications.densenet import DenseNet121

from sklearn.model\_selection import train\_test\_split

from keras.applications.inception\_v3 import InceptionV3

from keras.preprocessing.image import ImageDataGenerator

from keras.layers import Conv2D, AveragePooling2D, Flatten, Dense

from sklearn.metrics import precision\_score, recall\_score, f1\_score

from keras.layers import Input, Conv2D, MaxPooling2D, UpSampling2D, concatenate

from google.colab import drive

drive.mount('/content/drive')

# Define early stopping callback

early\_stopping = EarlyStopping(monitor='val\_loss', patience=3, verbose=1, restore\_best\_weights=True)

# Set the dimensions of the input images

img\_width, img\_height = 224, 224

# Set the number of classes

num\_classes = 2

# Set the path to the directory containing the images

data\_path = "/content/drive/MyDrive/ACRIMAsp"

# Load the image data and labels

X = []

y = []

for label, folder\_name in enumerate(["Normal", "Glaucoma"]):

folder\_path = os.path.join(data\_path, folder\_name)

for filename in os.listdir(folder\_path):

image = Image.open(os.path.join(folder\_path, filename))

image = image.resize((img\_width, img\_height))

image = np.array(image)

X.append(image)

y.append(label)

# Set a fixed random seed for reproducibility

seed = 42

random.seed(seed)

np.random.seed(seed)

# Convert the image data and labels to NumPy arrays

X = np.array(X)

y = np.array(y)

# Apply SMOTE to balance the data

smote = SMOTE(random\_state=42)

X, y = smote.fit\_resample(X.reshape(X.shape[0], -1), y)

# Reshape X back to 3D format

X = X.reshape(X.shape[0], img\_width, img\_height, 3)

# Split the data into training and testing sets

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

# Normalize the image data to the range [0, 1]

X\_train = X\_train.astype('float32') / 255.0

X\_test = X\_test.astype('float32') / 255.0

# Convert the labels to one-hot encoding

y\_train = np.eye(num\_classes)[y\_train]

y\_test = np.eye(num\_classes)[y\_test]

# Create an ImageDataGenerator for data augmentation

datagen = ImageDataGenerator(

rotation\_range=10,

width\_shift\_range=0.1,

height\_shift\_range=0.1,

shear\_range=0.1,

zoom\_range=0.1,

horizontal\_flip=True,

vertical\_flip=True

)

# Create the DenseNet121 model

base\_dense\_model = DenseNet121(include\_top=False, weights='imagenet', input\_shape=(img\_width, img\_height, 3))

# Create the Xception model

base\_xception\_model = Xception(include\_top=False, weights='imagenet', input\_shape=(img\_width, img\_height, 3))

# Load the InceptionV3 model without the top classification layer

base\_inception\_model = InceptionV3(weights='imagenet', include\_top=False, input\_shape=(img\_width, img\_height, 3))

# Add custom classification layers on top of the base dense model

dense\_model = Sequential()

dense\_model.add(base\_dense\_model)

dense\_model.add(Flatten())

dense\_model.add(Dense(256, activation='relu'))

dense\_model.add(Dense(num\_classes, activation='softmax'))

# Add custom classification layers on top of the base inception model

inception\_model = Sequential()

inception\_model.add(base\_inception\_model)

inception\_model.add(Flatten())

inception\_model.add(Dense(256, activation='relu'))

inception\_model.add(Dense(num\_classes, activation='softmax'))

# Add custom classification layers on top of the Xception base model

xception\_model = Sequential()

xception\_model.add(base\_xception\_model)

xception\_model.add(Flatten())

xception\_model.add(Dense(256, activation='relu'))

xception\_model.add(Dense(num\_classes, activation='softmax'))

# Create the LeNet-5 model

lenet\_model = Sequential()

lenet\_model.add(Conv2D(6, kernel\_size=(5, 5), activation='relu', input\_shape=(img\_width, img\_height, 3)))

lenet\_model.add(AveragePooling2D(pool\_size=(2, 2)))

lenet\_model.add(Conv2D(16, kernel\_size=(5, 5), activation='relu'))

lenet\_model.add(AveragePooling2D(pool\_size=(2, 2)))

lenet\_model.add(Conv2D(32, kernel\_size=(5, 5), activation='relu'))

lenet\_model.add(AveragePooling2D(pool\_size=(2, 2)))

lenet\_model.add(Flatten())

lenet\_model.add(Dense(120, activation='relu'))

lenet\_model.add(Dense(84, activation='relu'))

lenet\_model.add(Dense(num\_classes, activation='softmax'))

# Compile the dense model

dense\_model.compile(loss='binary\_crossentropy', optimizer='adam', metrics=['accuracy'])

# Compile the xception model

xception\_model.compile(loss='binary\_crossentropy', optimizer='adam', metrics=['accuracy'])

# Compile the lenet model

lenet\_model.compile(loss='binary\_crossentropy', optimizer='adam', metrics=['accuracy'])

# Compile the inception model

inception\_model.compile(loss='binary\_crossentropy', optimizer='adam', metrics=['accuracy'])

# Training the DenseNet model

dense\_model.fit(

datagen.flow(X\_train, y\_train, batch\_size=32, seed = seed),

steps\_per\_epoch=len(X\_train) // 32,

epochs=10,

validation\_data=(X\_test, y\_test),

callbacks=[early\_stopping]

)

# Train the xception model

xception\_model.fit(

datagen.flow(X\_train, y\_train, batch\_size=32, seed = seed),

steps\_per\_epoch=len(X\_train) // 32,

epochs=10,

validation\_data=(X\_test, y\_test),

callbacks=[early\_stopping]

)

# Fit the model with data augmentation

lenet\_model.fit(

datagen.flow(X\_train, y\_train, batch\_size=32, seed = seed),

steps\_per\_epoch=len(X\_train) // 32,

epochs=10,

validation\_data=(X\_test, y\_test),

callbacks=[early\_stopping]

)

# Train the GoogleNet model

inception\_model.fit(

datagen.flow(X\_train, y\_train, batch\_size=32, seed = seed),

steps\_per\_epoch=len(X\_train) // 32,

epochs=10,

validation\_data=(X\_test, y\_test),

callbacks=[early\_stopping]

)

# testing data loss and accuracy on the test data

loss\_value, accuracy\_val = xception\_model.evaluate(X\_test, y\_test)

print(f"testing data loss\_value of xception: {loss\_value:.4f}")

print(f"Testing data accuracy\_val of xception: {accuracy\_val:.4f}")

# testing data loss\_value and accuracy\_val on the test data

loss\_value, accuracy\_val = dense\_model.evaluate(X\_test, y\_test)

print(f"testing data loss\_value of densenet: {loss\_value:.4f}")

print(f"Testing data accuracy\_val of densenet: {accuracy\_val:.4f}")

# testing data loss\_value and accuracy\_val on the test data

loss\_value, accuracy\_val = inception\_model.evaluate(X\_test, y\_test)

print(f"testing data loss\_value of inception: {loss\_value:.4f}")

print(f"Testing data accuracy\_val of inception: {accuracy\_val:.4f}")

# testing data loss\_value and accuracy\_val on the test data

loss\_value, accuracy\_val = lenet\_model.evaluate(X\_test, y\_test)

print(f"testing data loss\_value of lenet\_model: {loss\_value:.4f}")

print(f"Testing data accuracy\_val of lenet\_model: {accuracy\_val:.4f}")

# 1. Evaluate the dense model on the test data

y\_pre\_dense = dense\_model.predict(X\_test)

y\_pre\_dense\_classes = np.argmax(y\_pre\_dense, axis=1)

y\_tru\_dense = np.argmax(y\_test, axis=1)

# find precision, recall, and F1 score for dense

precision\_dense = precision\_score(y\_tru\_dense, y\_pre\_dense\_classes)

recall\_dense = recall\_score(y\_tru\_dense, y\_pre\_dense\_classes)

f1\_dense = f1\_score(y\_tru\_dense, y\_pre\_dense\_classes)

# 2. Evaluate the Inception model on the test data

y\_pre\_inception = inception\_model.predict(X\_test)

y\_pre\_inception\_classes = np.argmax(y\_pre\_inception, axis=1)

y\_tru\_inception = np.argmax(y\_test, axis=1)

# find precision, recall, and F1 score for Inception

precision\_inception = precision\_score(y\_tru\_inception, y\_pre\_inception\_classes)

recall\_inception = recall\_score(y\_tru\_inception, y\_pre\_inception\_classes)

f1\_inception = f1\_score(y\_tru\_inception, y\_pre\_inception\_classes)

# 3. Evaluate the Xception model on the test data

y\_pre\_xception = xception\_model.predict(X\_test)

y\_pre\_xception\_classes = np.argmax(y\_pre\_xception, axis=1)

y\_tru\_xception = np.argmax(y\_test, axis=1)

# find precision, recall, and F1 score for Xception

precision\_xception = precision\_score(y\_tru\_xception, y\_pre\_xception\_classes)

recall\_xception = recall\_score(y\_tru\_xception, y\_pre\_xception\_classes)

f1\_xception = f1\_score(y\_tru\_xception, y\_pre\_xception\_classes)

# 4. Evaluate the LeNet model on the test data

y\_pre\_lenet = lenet\_model.predict(X\_test)

y\_pre\_lenet\_classes = np.argmax(y\_pre\_lenet, axis=1)

y\_tru\_lenet = np.argmax(y\_test, axis=1)

# find precision, recall, and F1 score for LeNet

precision\_lenet = precision\_score(y\_tru\_lenet, y\_pre\_lenet\_classes)

recall\_lenet = recall\_score(y\_tru\_lenet, y\_pre\_lenet\_classes)

f1\_lenet = f1\_score(y\_tru\_lenet, y\_pre\_lenet\_classes)

# Obtain the predicted classes

y\_pre\_classes\_dense = np.argmax(y\_pre\_dense, axis=1)

y\_tru\_classes\_dense = np.argmax(y\_test, axis=1)

# Compute the confusion matrix

cm = confusion\_matrix(y\_tru\_classes\_dense, y\_pre\_classes\_dense)

# Create a function to plot the confusion matrix

def plot\_confusion\_matrix(cm, classes):

plt.imshow(cm, interpolation='nearest', cmap=plt.cm.Blues)

plt.colorbar()

tick\_marks = np.arange(len(classes))

plt.xticks(tick\_marks, classes, rotation=45)

plt.yticks(tick\_marks, classes)

plt.xlabel('Predicted Label')

plt.ylabel('True Label')

# Plot the confusion matrix

plt.figure()

plot\_confusion\_matrix(cm, classes=class\_names)

plt.title('Confusion Matrix of dense')

plt.show()

#Precision, Recall, F1 Score of DenseNet Model

print(f"precision of dense: {precision\_dense:.4f}")

print(f"recall dense: {recall\_dense:.4f}")

print(f"F1 Score of dense: {f1\_dense:.4f}")

#Precision, Recall, F1 Score of Xception Model

print(f"Precision of xception: {precision\_xception:.4f}")

print(f"Recall of xception: {recall\_xception:.4f}")

print(f"F1 Score of xception: {f1\_xception:.4f}")

#Precision, Recall, F1 Score of LeNet Model

print(f"Precision of Lenet: {precision\_lenet:.4f}")

print(f"Recall of Lenet: {recall\_lenet:.4f}")

print(f"F1 Score of Lenet: {f1\_lenet:.4f}")

#Precision, Recall, F1 Score of Inception Model

print(f"Precision of Inception: {precision\_inception:.4f}")

print(f"Recall of Inception: {recall\_inception:.4f}")

print(f"F1 Score of Inception: {f1\_inception:.4f}")

# Precision, Recall, and F1-score values for each model

models = ['Dense', 'Xception', 'LeNet', 'Inception']

precision\_values = [precision\_dense, precision\_xception, precision\_lenet, precision\_inception]

recall\_values = [recall\_dense, recall\_xception, recall\_lenet, recall\_inception]

f1\_values = [f1\_dense, f1\_xception, f1\_lenet, f1\_inception]

# Define different shades of blue for each model

colors = ['#1f77b4', '#1f4e77', '#1f3552', '#1f2235']

# Initialize plots

plt.figure(figsize=(12, 6))

# Plot Precision scores

plt.subplot(1, 3, 1)

plt.bar(models, precision\_values, color=colors)

plt.xlabel('Models')

plt.ylabel('Precision')

plt.title('Precision Scores')

# Plot Recall scores

plt.subplot(1, 3, 2)

plt.bar(models, recall\_values, color=colors)

plt.xlabel('Models')

plt.ylabel('Recall')

plt.title('Recall Scores')

# Plot F1 scores

plt.subplot(1, 3, 3)

plt.bar(models, f1\_values, color=colors)

plt.xlabel('Models')

plt.ylabel('F1-Score')

plt.title('F1-Score Scores')

plt.tight\_layout()

plt.show()

#ROC and AUC Curve

# Obtain the predicted probabilities for the positive class of DenseNet model

y\_pre\_prob\_dense = dense\_model.predict(X\_test)[:, 1]

# Obtain the predicted probabilities for the positive class of Xception Model

y\_pre\_prob\_xception = xception\_model.predict(X\_test)[:, 1]

# Obtain the predicted probabilities for the positive class of LeNet model

y\_pre\_prob\_lenet = lenet\_model.predict(X\_test)[:, 1]

# Obtain the predicted probabilities for the positive class of Inception Model

y\_pre\_prob\_inception = inception\_model.predict(X\_test)[:, 1]

# Compute the false positive rate, true positive rate, and classification threshold of dense

fpr\_dense, tpr\_dense, thresholds\_dense = roc\_curve(y\_tru\_dense, y\_pre\_prob\_dense)

# Compute the AUC score

roc\_auc = auc(fpr\_dense, tpr\_dense)

# Plot the ROC curve

plt.figure()

plt.plot(fpr\_dense, tpr\_dense, color='darkorange', lw=2, label='ROC curve (AUC = %0.2f)' % roc\_auc)

plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')

plt.xlim([0.0, 1.0])

plt.ylim([0.0, 1.05])

plt.xlabel('False Positive Rate')

plt.ylabel('True Positive Rate')

plt.title('Receiver Operating Characteristic of dense')

plt.legend(loc="lower right")

plt.show()

# Compute the false positive rate, true positive rate, and classification threshold of Xception

fpr\_xception, tpr\_xception, thresholds\_xception = roc\_curve(y\_tru\_xception, y\_pre\_prob\_xception)

# Compute the AUC score

roc\_auc = auc(fpr\_xception, tpr\_xception)

# Plot the ROC curve

plt.figure()

plt.plot(fpr\_xception, tpr\_xception, color='darkorange', lw=2, label='ROC curve (AUC = %0.2f)' % roc\_auc)

plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')

plt.xlim([0.0, 1.0])

plt.ylim([0.0, 1.05])

plt.xlabel('False Positive Rate')

plt.ylabel('True Positive Rate')

plt.title('Receiver Operating Characteristic')

plt.legend(loc="lower right")

plt.show()

# Compute the false positive rate, true positive rate, and classification threshold of Inception

fpr\_inception, tpr\_inception, thresholds\_inception = roc\_curve(y\_tru\_inception, y\_pre\_prob\_inception)

# Compute the AUC score

roc\_auc = auc(fpr\_inception, tpr\_inception)

# Plot the ROC curve

plt.figure()

plt.plot(fpr\_inception, tpr\_inception, color='darkorange', lw=2, label='ROC curve (AUC = %0.2f)' % roc\_auc)

plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')

plt.xlim([0.0, 1.0])

plt.ylim([0.0, 1.05])

plt.xlabel('False Positive Rate')

plt.ylabel('True Positive Rate')

plt.title('Receiver Operating Characteristic')

plt.legend(loc="lower right")

plt.show()

# Compute the false positive rate, true positive rate, and classification threshold of LeNet.

fpr\_lenet, tpr\_lenet, thresholds\_lenet = roc\_curve(y\_tru\_lenet, y\_pre\_prob\_lenet)

# Compute the AUC score

roc\_auc = auc(fpr\_lenet, tpr\_lenet)

# Plot the ROC curve

plt.figure()

plt.plot(fpr\_lenet, tpr\_lenet, color='darkorange', lw=2, label='ROC curve (AUC = %0.2f)' % roc\_auc)

plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')

plt.xlim([0.0, 1.0])

plt.ylim([0.0, 1.05])

plt.xlabel('False Positive Rate')

plt.ylabel('True Positive Rate')

plt.title('Receiver Operating Characteristic')

plt.legend(loc="lower right")

plt.show()

#ROC curve for all models for ACRIMA dataset.

# Data for all models

models = ['DenseNet', 'Xception', 'Inception', 'LeNet']

fprs = [fpr\_dense, fpr\_xception, fpr\_inception, fpr\_lenet]

tprs = [tpr\_dense, tpr\_xception, tpr\_inception, tpr\_lenet]

auc\_scores = [auc(fpr, tpr) for fpr, tpr in zip(fprs, tprs)]

# Plotting ROC curves for all models

plt.figure(figsize=(10, 6))

for model, fpr, tpr, auc\_score in zip(models, fprs, tprs, auc\_scores):

plt.plot(fpr, tpr, lw=2, label=f'{model} (AUC = {auc\_score:.2f})')

plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')

plt.xlim([0.0, 1.0])

plt.ylim([0.0, 1.05])

plt.xlabel('False Positive Rate')

plt.ylabel('True Positive Rate')

plt.title('Receiver Operating Characteristic of Different Models')

plt.legend(loc="lower right")

plt.show()

#Confusion Matrix

# Define the class labels

class\_names = ['No', 'Yes']

# Obtain the predicted classes for dense.

y\_pre\_classes\_dense = np.argmax(y\_pre\_dense, axis=1)

y\_tru\_classes\_dense = np.argmax(y\_test, axis=1)

# Compute the confusion matrix

cm = confusion\_matrix(y\_tru\_classes\_dense, y\_pre\_classes\_dense)

# Create a function to plot the confusion matrix

def plot\_confusion\_matrix(cm, classes):

plt.imshow(cm, interpolation='nearest', cmap=plt.cm.Blues)

plt.colorbar()

tick\_marks = np.arange(len(classes))

plt.xticks(tick\_marks, classes, rotation=45)

plt.yticks(tick\_marks, classes)

plt.xlabel('Predicted Label')

plt.ylabel('True Label')

# Plot the confusion matrix

plt.figure()

plot\_confusion\_matrix(cm, classes=class\_names)

plt.title('Confusion Matrix of dense')

plt.show()

# Obtain the predicted classes for lenet

y\_pre\_classes\_lenet = np.argmax(y\_pre\_lenet, axis=1)

y\_tru\_classes\_lenet = np.argmax(y\_test, axis=1)

# Compute the confusion matrix

cm = confusion\_matrix(y\_tru\_classes\_lenet, y\_pre\_classes\_lenet)

# Create a function to plot the confusion matrix

def plot\_confusion\_matrix(cm, classes):

plt.imshow(cm, interpolation='nearest', cmap=plt.cm.Blues)

plt.colorbar()

tick\_marks = np.arange(len(classes))

plt.xticks(tick\_marks, classes, rotation=45)

plt.yticks(tick\_marks, classes)

plt.xlabel('Predicted Label')

plt.ylabel('True Label')

# Plot the confusion matrix

plt.figure()

plot\_confusion\_matrix(cm, classes=class\_names)

plt.title('Confusion Matrix of lenet')

plt.show()

# Obtain the predicted classes for Xception

y\_pre\_classes\_xception = np.argmax(y\_pre\_xception, axis=1)

y\_tru\_classes\_xception = np.argmax(y\_test, axis=1)

# Compute the confusion matrix

cm = confusion\_matrix(y\_tru\_classes\_xception, y\_pre\_classes\_xception)

# Create a function to plot the confusion matrix

def plot\_confusion\_matrix(cm, classes):

plt.imshow(cm, interpolation='nearest', cmap=plt.cm.Blues)

plt.colorbar()

tick\_marks = np.arange(len(classes))

plt.xticks(tick\_marks, classes, rotation=45)

plt.yticks(tick\_marks, classes)

plt.xlabel('Predicted Label')

plt.ylabel('True Label')

# Plot the confusion matrix

plt.figure()

plot\_confusion\_matrix(cm, classes=class\_names)

plt.title('Confusion Matrix of xception')

plt.show()

# Obtain the predicted classes for Inception

y\_pre\_classes\_inception = np.argmax(y\_pre\_inception, axis=1)

y\_tru\_classes\_inception = np.argmax(y\_test, axis=1)

# Compute the confusion matrix

cm = confusion\_matrix(y\_tru\_classes\_inception, y\_pre\_classes\_inception)

# Create a function to plot the confusion matrix

def plot\_confusion\_matrix(cm, classes):

plt.imshow(cm, interpolation='nearest', cmap=plt.cm.Blues)

plt.colorbar()

tick\_marks = np.arange(len(classes))

plt.xticks(tick\_marks, classes, rotation=45)

plt.yticks(tick\_marks, classes)

plt.xlabel('Predicted Label')

plt.ylabel('True Label')

# Plot the confusion matrix

plt.figure()

plot\_confusion\_matrix(cm, classes=class\_names)

plt.title('Confusion Matrix of inception')

plt.show()

#confusion matrix for all models

cm\_xception = confusion\_matrix(y\_tru\_classes\_xception, y\_pre\_classes\_xception)

cm\_inception = confusion\_matrix(y\_tru\_classes\_inception, y\_pre\_classes\_inception)

cm\_lenet = confusion\_matrix(y\_tru\_classes\_lenet, y\_pre\_classes\_lenet)

cm\_dense = confusion\_matrix(y\_tru\_classes\_dense, y\_pre\_classes\_dense)

# List of model names

model\_names = ['Inception', 'Xception', 'LeNet', 'DenseNet']

# List of confusion matrices for each model

conf\_matrices = [

cm\_inception,

cm\_xception,

cm\_lenet,

cm\_dense

]

# Plot confusion matrices

plt.figure(figsize=(15, 10))

for i, (model\_name, conf\_matrix) in enumerate(zip(model\_names, conf\_matrices), start=1):

plt.subplot(2, 2, i)

sns.heatmap(conf\_matrix, annot=True, fmt='d', cmap='Blues',

xticklabels=["Class 0", "Class 1"], yticklabels=["Class 0", "Class 1"])

plt.title(f'Confusion Matrix - {model\_name}')

plt.xlabel('Predicted')

plt.ylabel('Actual')

plt.tight\_layout()

plt.show()

from sklearn.ensemble import BaggingClassifier

from sklearn.metrics import accuracy\_score

# Convert one-hot encoded y to a 1D array of class labels

y\_train\_labels = np.argmax(y\_train, axis=1)

y\_test\_labels = np.argmax(y\_test, axis=1)

# Flatten the image data

X\_train\_flattened = X\_train.reshape(X\_train.shape[0], -1)

X\_test\_flattened = X\_test.reshape(X\_test.shape[0], -1)

# Create bagging ensemble (adjust n\_estimators as needed)

bagging\_models = [

dense\_model,

xception\_model,

inception\_model,

lenet\_model

]

bagging\_ensemble = BaggingClassifier(

base\_estimator=None, # Specify None as the base\_estimator since using our pre-loaded models

n\_estimators=len(bagging\_models), # Number of models in the ensemble

random\_state=42

)

# Fit the bagging ensemble

bagging\_ensemble.fit(X\_train\_flattened, y\_train\_labels)

# Evaluate the ensemble on the flattened test data

y\_pred\_bagging = bagging\_ensemble.predict(X\_test\_flattened)

# find accuracy\_score

accuracy\_bagging = accuracy\_score(y\_test\_labels, y\_pred\_bagging)

print(f"Accuracy of the Bagging Ensemble of acrima: {accuracy\_bagging:.4f}")

**Code for Drishti-GS1 dataset with DenseNet, Xception, LeNet and Inception-V3.**

#Importing packages

import os

import random

import numpy as np

import seaborn as sns

from PIL import Image

from keras.models import Model

import matplotlib.pyplot as plt

from keras.models import Sequential

from imblearn.over\_sampling import SMOTE

from keras.callbacks import EarlyStopping

from sklearn.metrics import roc\_curve, auc

from sklearn.metrics import confusion\_matrix

from keras.applications.xception import Xception

from keras.applications.densenet import DenseNet121

from sklearn.model\_selection import train\_test\_split

from keras.applications.inception\_v3 import InceptionV3

from keras.preprocessing.image import ImageDataGenerator

from keras.layers import Conv2D, AveragePooling2D, Flatten, Dense

from sklearn.metrics import precision\_score, recall\_score, f1\_score

from keras.layers import Input, Conv2D, MaxPooling2D, UpSampling2D, concatenate

from google.colab import drive

drive.mount('/content/drive')

# Define early stopping callback

early\_stopping = EarlyStopping(monitor='val\_loss', patience=3, verbose=1, restore\_best\_weights=True)

# Set the dimensions of the input images

img\_width, img\_height = 224, 224

# Set the number of classes

num\_classes = 2

# Set the path to the directory containing the images

data\_path = "/content/drive/MyDrive/DS\_GS1\_sp"

# Load the image data and labels

X = []

y = []

for label, folder\_name in enumerate(["Normal\_ds", "Glaucoma\_ds"]):

folder\_path = os.path.join(data\_path, folder\_name)

for filename in os.listdir(folder\_path):

image = Image.open(os.path.join(folder\_path, filename))

image = image.resize((img\_width, img\_height))

image = np.array(image)

X.append(image)

y.append(label)

# Set a fixed random seed for reproducibility

seed = 42

random.seed(seed)

np.random.seed(seed)

# Convert the image data and labels to NumPy arrays

X = np.array(X)

y = np.array(y)

# Apply SMOTE to balance the data

smote = SMOTE(random\_state=42)

X, y = smote.fit\_resample(X.reshape(X.shape[0], -1), y)

# Reshape X back to 3D format

X = X.reshape(X.shape[0], img\_width, img\_height, 3)

# Split the data into training and testing sets

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

# Normalize the image data to the range [0, 1]

X\_train = X\_train.astype('float32') / 255.0

X\_test = X\_test.astype('float32') / 255.0

# Convert the labels to one-hot encoding

y\_train = np.eye(num\_classes)[y\_train]

y\_test = np.eye(num\_classes)[y\_test]

# Create an ImageDataGenerator for data augmentation

datagen = ImageDataGenerator(

rotation\_range=10,

width\_shift\_range=0.1,

height\_shift\_range=0.1,

shear\_range=0.1,

zoom\_range=0.1,

horizontal\_flip=True,

vertical\_flip=True

)

# Create the DenseNet121 model

base\_dense\_model = DenseNet121(include\_top=False, weights='imagenet', input\_shape=(img\_width, img\_height, 3))

# Create the Xception model

base\_xception\_model = Xception(include\_top=False, weights='imagenet', input\_shape=(img\_width, img\_height, 3))

# Load the InceptionV3 model without the top classification layer

base\_inception\_model = InceptionV3(weights='imagenet', include\_top=False, input\_shape=(img\_width, img\_height, 3))

# Add custom classification layers on top of the base dense model

dense\_model = Sequential()

dense\_model.add(base\_dense\_model)

dense\_model.add(Flatten())

dense\_model.add(Dense(256, activation='relu'))

dense\_model.add(Dense(num\_classes, activation='softmax'))

# Add custom classification layers on top of the base inception model

inception\_model = Sequential()

inception\_model.add(base\_inception\_model)

inception\_model.add(Flatten())

inception\_model.add(Dense(256, activation='relu'))

inception\_model.add(Dense(num\_classes, activation='softmax'))

# Add custom classification layers on top of the Xception base model

xception\_model = Sequential()

xception\_model.add(base\_xception\_model)

xception\_model.add(Flatten())

xception\_model.add(Dense(256, activation='relu'))

xception\_model.add(Dense(num\_classes, activation='softmax'))

# Create the LeNet-5 model

lenet\_model = Sequential()

lenet\_model.add(Conv2D(6, kernel\_size=(5, 5), activation='relu', input\_shape=(img\_width, img\_height, 3)))

lenet\_model.add(AveragePooling2D(pool\_size=(2, 2)))

lenet\_model.add(Conv2D(16, kernel\_size=(5, 5), activation='relu'))

lenet\_model.add(AveragePooling2D(pool\_size=(2, 2)))

lenet\_model.add(Conv2D(32, kernel\_size=(5, 5), activation='relu'))

lenet\_model.add(AveragePooling2D(pool\_size=(2, 2)))

lenet\_model.add(Flatten())

lenet\_model.add(Dense(120, activation='relu'))

lenet\_model.add(Dense(84, activation='relu'))

lenet\_model.add(Dense(num\_classes, activation='softmax'))

# Compile the dense model

dense\_model.compile(loss='binary\_crossentropy', optimizer='adam', metrics=['accuracy'])

# Compile the xception model

xception\_model.compile(loss='binary\_crossentropy', optimizer='adam', metrics=['accuracy'])

# Compile the lenet model

lenet\_model.compile(loss='binary\_crossentropy', optimizer='adam', metrics=['accuracy'])

# Compile the inception model

inception\_model.compile(loss='binary\_crossentropy', optimizer='adam', metrics=['accuracy'])

# Training the DenseNet model

dense\_model.fit(

datagen.flow(X\_train, y\_train, batch\_size=32, seed = seed),

steps\_per\_epoch=len(X\_train) // 32,

epochs=10,

validation\_data=(X\_test, y\_test),

callbacks=[early\_stopping]

)

# Train the xception model

xception\_model.fit(

datagen.flow(X\_train, y\_train, batch\_size=32, seed = seed),

steps\_per\_epoch=len(X\_train) // 32,

epochs=10,

validation\_data=(X\_test, y\_test),

callbacks=[early\_stopping]

)

# Fit the model with data augmentation

lenet\_model.fit(

datagen.flow(X\_train, y\_train, batch\_size=32, seed = seed),

steps\_per\_epoch=len(X\_train) // 32,

epochs=10,

validation\_data=(X\_test, y\_test),

callbacks=[early\_stopping]

)

# Train the GoogleNet model

inception\_model.fit(

datagen.flow(X\_train, y\_train, batch\_size=32, seed = seed),

steps\_per\_epoch=len(X\_train) // 32,

epochs=10,

validation\_data=(X\_test, y\_test),

callbacks=[early\_stopping]

)

# testing data loss and accuracy on the test data

loss\_value, accuracy\_val = xception\_model.evaluate(X\_test, y\_test)

print(f"testing data loss\_value of xception: {loss\_value:.4f}")

print(f"Testing data accuracy\_val of xception: {accuracy\_val:.4f}")

# testing data loss\_value and accuracy\_val on the test data

loss\_value, accuracy\_val = dense\_model.evaluate(X\_test, y\_test)

print(f"testing data loss\_value of densenet: {loss\_value:.4f}")

print(f"Testing data accuracy\_val of densenet: {accuracy\_val:.4f}")

# testing data loss\_value and accuracy\_val on the test data

loss\_value, accuracy\_val = inception\_model.evaluate(X\_test, y\_test)

print(f"testing data loss\_value of inception: {loss\_value:.4f}")

print(f"Testing data accuracy\_val of inception: {accuracy\_val:.4f}")

# testing data loss\_value and accuracy\_val on the test data

loss\_value, accuracy\_val = lenet\_model.evaluate(X\_test, y\_test)

print(f"testing data loss\_value of lenet\_model: {loss\_value:.4f}")

print(f"Testing data accuracy\_val of lenet\_model: {accuracy\_val:.4f}")

# 1. Evaluate the dense model on the test data

y\_pre\_dense = dense\_model.predict(X\_test)

y\_pre\_dense\_classes = np.argmax(y\_pre\_dense, axis=1)

y\_tru\_dense = np.argmax(y\_test, axis=1)

# find precision, recall, and F1 score for dense

precision\_dense = precision\_score(y\_tru\_dense, y\_pre\_dense\_classes)

recall\_dense = recall\_score(y\_tru\_dense, y\_pre\_dense\_classes)

f1\_dense = f1\_score(y\_tru\_dense, y\_pre\_dense\_classes)

# 2. Evaluate the Inception model on the test data

y\_pre\_inception = inception\_model.predict(X\_test)

y\_pre\_inception\_classes = np.argmax(y\_pre\_inception, axis=1)

y\_tru\_inception = np.argmax(y\_test, axis=1)

# find precision, recall, and F1 score for Inception

precision\_inception = precision\_score(y\_tru\_inception, y\_pre\_inception\_classes)

recall\_inception = recall\_score(y\_tru\_inception, y\_pre\_inception\_classes)

f1\_inception = f1\_score(y\_tru\_inception, y\_pre\_inception\_classes)

# 3. Evaluate the Xception model on the test data

y\_pre\_xception = xception\_model.predict(X\_test)

y\_pre\_xception\_classes = np.argmax(y\_pre\_xception, axis=1)

y\_tru\_xception = np.argmax(y\_test, axis=1)

# find precision, recall, and F1 score for Xception

precision\_xception = precision\_score(y\_tru\_xception, y\_pre\_xception\_classes)

recall\_xception = recall\_score(y\_tru\_xception, y\_pre\_xception\_classes)

f1\_xception = f1\_score(y\_tru\_xception, y\_pre\_xception\_classes)

# 4. Evaluate the LeNet model on the test data

y\_pre\_lenet = lenet\_model.predict(X\_test)

y\_pre\_lenet\_classes = np.argmax(y\_pre\_lenet, axis=1)

y\_tru\_lenet = np.argmax(y\_test, axis=1)

# find precision, recall, and F1 score for LeNet

precision\_lenet = precision\_score(y\_tru\_lenet, y\_pre\_lenet\_classes)

recall\_lenet = recall\_score(y\_tru\_lenet, y\_pre\_lenet\_classes)

f1\_lenet = f1\_score(y\_tru\_lenet, y\_pre\_lenet\_classes)

# Obtain the predicted classes

y\_pre\_classes\_dense = np.argmax(y\_pre\_dense, axis=1)

y\_tru\_classes\_dense = np.argmax(y\_test, axis=1)

# Compute the confusion matrix

cm = confusion\_matrix(y\_tru\_classes\_dense, y\_pre\_classes\_dense)

# Create a function to plot the confusion matrix

def plot\_confusion\_matrix(cm, classes):

plt.imshow(cm, interpolation='nearest', cmap=plt.cm.Blues)

plt.colorbar()

tick\_marks = np.arange(len(classes))

plt.xticks(tick\_marks, classes, rotation=45)

plt.yticks(tick\_marks, classes)

plt.xlabel('Predicted Label')

plt.ylabel('True Label')

# Plot the confusion matrix

plt.figure()

plot\_confusion\_matrix(cm, classes=class\_names)

plt.title('Confusion Matrix of dense')

plt.show()

#Precision, Recall, F1 Score of DenseNet Model

print(f"precision of dense: {precision\_dense:.4f}")

print(f"recall dense: {recall\_dense:.4f}")

print(f"F1 Score of dense: {f1\_dense:.4f}")

#Precision, Recall, F1 Score of Xception Model

print(f"Precision of xception: {precision\_xception:.4f}")

print(f"Recall of xception: {recall\_xception:.4f}")

print(f"F1 Score of xception: {f1\_xception:.4f}")

#Precision, Recall, F1 Score of LeNet Model

print(f"Precision of Lenet: {precision\_lenet:.4f}")

print(f"Recall of Lenet: {recall\_lenet:.4f}")

print(f"F1 Score of Lenet: {f1\_lenet:.4f}")

#Precision, Recall, F1 Score of Inception Model

print(f"Precision of Inception: {precision\_inception:.4f}")

print(f"Recall of Inception: {recall\_inception:.4f}")

print(f"F1 Score of Inception: {f1\_inception:.4f}")

# Precision, Recall, and F1-score values for each model

models = ['Dense', 'Xception', 'LeNet', 'Inception']

precision\_values = [precision\_dense, precision\_xception, precision\_lenet, precision\_inception]

recall\_values = [recall\_dense, recall\_xception, recall\_lenet, recall\_inception]

f1\_values = [f1\_dense, f1\_xception, f1\_lenet, f1\_inception]

# Define different shades of blue for each model

colors = ['#1f77b4', '#1f4e77', '#1f3552', '#1f2235']

# Initialize plots

plt.figure(figsize=(12, 6))

# Plot Precision scores

plt.subplot(1, 3, 1)

plt.bar(models, precision\_values, color=colors)

plt.xlabel('Models')

plt.ylabel('Precision')

plt.title('Precision Scores')

# Plot Recall scores

plt.subplot(1, 3, 2)

plt.bar(models, recall\_values, color=colors)

plt.xlabel('Models')

plt.ylabel('Recall')

plt.title('Recall Scores')

# Plot F1 scores

plt.subplot(1, 3, 3)

plt.bar(models, f1\_values, color=colors)

plt.xlabel('Models')

plt.ylabel('F1-Score')

plt.title('F1-Score Scores')

plt.tight\_layout()

plt.show()

#ROC and AUC Curve

# Obtain the predicted probabilities for the positive class of DenseNet model

y\_pre\_prob\_dense = dense\_model.predict(X\_test)[:, 1]

# Obtain the predicted probabilities for the positive class of Xception Model

y\_pre\_prob\_xception = xception\_model.predict(X\_test)[:, 1]

# Obtain the predicted probabilities for the positive class of LeNet model

y\_pre\_prob\_lenet = lenet\_model.predict(X\_test)[:, 1]

# Obtain the predicted probabilities for the positive class of Inception Model

y\_pre\_prob\_inception = inception\_model.predict(X\_test)[:, 1]

# Compute the false positive rate, true positive rate, and classification threshold of dense

fpr\_dense, tpr\_dense, thresholds\_dense = roc\_curve(y\_tru\_dense, y\_pre\_prob\_dense)

# Compute the AUC score

roc\_auc = auc(fpr\_dense, tpr\_dense)

# Plot the ROC curve

plt.figure()

plt.plot(fpr\_dense, tpr\_dense, color='darkorange', lw=2, label='ROC curve (AUC = %0.2f)' % roc\_auc)

plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')

plt.xlim([0.0, 1.0])

plt.ylim([0.0, 1.05])

plt.xlabel('False Positive Rate')

plt.ylabel('True Positive Rate')

plt.title('Receiver Operating Characteristic of dense')

plt.legend(loc="lower right")

plt.show()

# Compute the false positive rate, true positive rate, and classification threshold of Xception

fpr\_xception, tpr\_xception, thresholds\_xception = roc\_curve(y\_tru\_xception, y\_pre\_prob\_xception)

# Compute the AUC score

roc\_auc = auc(fpr\_xception, tpr\_xception)

# Plot the ROC curve

plt.figure()

plt.plot(fpr\_xception, tpr\_xception, color='darkorange', lw=2, label='ROC curve (AUC = %0.2f)' % roc\_auc)

plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')

plt.xlim([0.0, 1.0])

plt.ylim([0.0, 1.05])

plt.xlabel('False Positive Rate')

plt.ylabel('True Positive Rate')

plt.title('Receiver Operating Characteristic')

plt.legend(loc="lower right")

plt.show()

# Compute the false positive rate, true positive rate, and classification threshold of Inception

fpr\_inception, tpr\_inception, thresholds\_inception = roc\_curve(y\_tru\_inception, y\_pre\_prob\_inception)

# Compute the AUC score

roc\_auc = auc(fpr\_inception, tpr\_inception)

# Plot the ROC curve

plt.figure()

plt.plot(fpr\_inception, tpr\_inception, color='darkorange', lw=2, label='ROC curve (AUC = %0.2f)' % roc\_auc)

plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')

plt.xlim([0.0, 1.0])

plt.ylim([0.0, 1.05])

plt.xlabel('False Positive Rate')

plt.ylabel('True Positive Rate')

plt.title('Receiver Operating Characteristic')

plt.legend(loc="lower right")

plt.show()

# Compute the false positive rate, true positive rate, and classification threshold of LeNet.

fpr\_lenet, tpr\_lenet, thresholds\_lenet = roc\_curve(y\_tru\_lenet, y\_pre\_prob\_lenet)

# Compute the AUC score

roc\_auc = auc(fpr\_lenet, tpr\_lenet)

# Plot the ROC curve

plt.figure()

plt.plot(fpr\_lenet, tpr\_lenet, color='darkorange', lw=2, label='ROC curve (AUC = %0.2f)' % roc\_auc)

plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')

plt.xlim([0.0, 1.0])

plt.ylim([0.0, 1.05])

plt.xlabel('False Positive Rate')

plt.ylabel('True Positive Rate')

plt.title('Receiver Operating Characteristic')

plt.legend(loc="lower right")

plt.show()

#ROC curve for all models for Drishti-GS1 dataset.

# Data for all models

models = ['DenseNet', 'Xception', 'Inception', 'LeNet']

fprs = [fpr\_dense, fpr\_xception, fpr\_inception, fpr\_lenet]

tprs = [tpr\_dense, tpr\_xception, tpr\_inception, tpr\_lenet]

auc\_scores = [auc(fpr, tpr) for fpr, tpr in zip(fprs, tprs)]

# Plotting ROC curves for all models

plt.figure(figsize=(10, 6))

for model, fpr, tpr, auc\_score in zip(models, fprs, tprs, auc\_scores):

plt.plot(fpr, tpr, lw=2, label=f'{model} (AUC = {auc\_score:.2f})')

plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')

plt.xlim([0.0, 1.0])

plt.ylim([0.0, 1.05])

plt.xlabel('False Positive Rate')

plt.ylabel('True Positive Rate')

plt.title('Receiver Operating Characteristic of Different Models')

plt.legend(loc="lower right")

plt.show()

#Confusion Matrix

# Define the class labels

class\_names = ['No', 'Yes']

# Obtain the predicted classes for dense.

y\_pre\_classes\_dense = np.argmax(y\_pre\_dense, axis=1)

y\_tru\_classes\_dense = np.argmax(y\_test, axis=1)

# Compute the confusion matrix

cm = confusion\_matrix(y\_tru\_classes\_dense, y\_pre\_classes\_dense)

# Create a function to plot the confusion matrix

def plot\_confusion\_matrix(cm, classes):

plt.imshow(cm, interpolation='nearest', cmap=plt.cm.Blues)

plt.colorbar()

tick\_marks = np.arange(len(classes))

plt.xticks(tick\_marks, classes, rotation=45)

plt.yticks(tick\_marks, classes)

plt.xlabel('Predicted Label')

plt.ylabel('True Label')

# Plot the confusion matrix

plt.figure()

plot\_confusion\_matrix(cm, classes=class\_names)

plt.title('Confusion Matrix of dense')

plt.show()

# Obtain the predicted classes for lenet

y\_pre\_classes\_lenet = np.argmax(y\_pre\_lenet, axis=1)

y\_tru\_classes\_lenet = np.argmax(y\_test, axis=1)

# Compute the confusion matrix

cm = confusion\_matrix(y\_tru\_classes\_lenet, y\_pre\_classes\_lenet)

# Create a function to plot the confusion matrix

def plot\_confusion\_matrix(cm, classes):

plt.imshow(cm, interpolation='nearest', cmap=plt.cm.Blues)

plt.colorbar()

tick\_marks = np.arange(len(classes))

plt.xticks(tick\_marks, classes, rotation=45)

plt.yticks(tick\_marks, classes)

plt.xlabel('Predicted Label')

plt.ylabel('True Label')

# Plot the confusion matrix

plt.figure()

plot\_confusion\_matrix(cm, classes=class\_names)

plt.title('Confusion Matrix of lenet')

plt.show()

# Obtain the predicted classes for Xception

y\_pre\_classes\_xception = np.argmax(y\_pre\_xception, axis=1)

y\_tru\_classes\_xception = np.argmax(y\_test, axis=1)

# Compute the confusion matrix

cm = confusion\_matrix(y\_tru\_classes\_xception, y\_pre\_classes\_xception)

# Create a function to plot the confusion matrix

def plot\_confusion\_matrix(cm, classes):

plt.imshow(cm, interpolation='nearest', cmap=plt.cm.Blues)

plt.colorbar()

tick\_marks = np.arange(len(classes))

plt.xticks(tick\_marks, classes, rotation=45)

plt.yticks(tick\_marks, classes)

plt.xlabel('Predicted Label')

plt.ylabel('True Label')

# Plot the confusion matrix

plt.figure()

plot\_confusion\_matrix(cm, classes=class\_names)

plt.title('Confusion Matrix of xception')

plt.show()

# Obtain the predicted classes for Inception

y\_pre\_classes\_inception = np.argmax(y\_pre\_inception, axis=1)

y\_tru\_classes\_inception = np.argmax(y\_test, axis=1)

# Compute the confusion matrix

cm = confusion\_matrix(y\_tru\_classes\_inception, y\_pre\_classes\_inception)

# Create a function to plot the confusion matrix

def plot\_confusion\_matrix(cm, classes):

plt.imshow(cm, interpolation='nearest', cmap=plt.cm.Blues)

plt.colorbar()

tick\_marks = np.arange(len(classes))

plt.xticks(tick\_marks, classes, rotation=45)

plt.yticks(tick\_marks, classes)

plt.xlabel('Predicted Label')

plt.ylabel('True Label')

# Plot the confusion matrix

plt.figure()

plot\_confusion\_matrix(cm, classes=class\_names)

plt.title('Confusion Matrix of inception')

plt.show()

#confusion matrix for all models

cm\_xception = confusion\_matrix(y\_tru\_classes\_xception, y\_pre\_classes\_xception)

cm\_inception = confusion\_matrix(y\_tru\_classes\_inception, y\_pre\_classes\_inception)

cm\_lenet = confusion\_matrix(y\_tru\_classes\_lenet, y\_pre\_classes\_lenet)

cm\_dense = confusion\_matrix(y\_tru\_classes\_dense, y\_pre\_classes\_dense)

# List of model names

model\_names = ['Inception', 'Xception', 'LeNet', 'DenseNet']

# List of confusion matrices for each model

conf\_matrices = [

cm\_inception,

cm\_xception,

cm\_lenet,

cm\_dense

]

# Plot confusion matrices

plt.figure(figsize=(15, 10))

for i, (model\_name, conf\_matrix) in enumerate(zip(model\_names, conf\_matrices), start=1):

plt.subplot(2, 2, i)

sns.heatmap(conf\_matrix, annot=True, fmt='d', cmap='Blues',

xticklabels=["Class 0", "Class 1"], yticklabels=["Class 0", "Class 1"])

plt.title(f'Confusion Matrix - {model\_name}')

plt.xlabel('Predicted')

plt.ylabel('Actual')

plt.tight\_layout()

plt.show()

from sklearn.ensemble import BaggingClassifier

from sklearn.metrics import accuracy\_score

# Convert one-hot encoded y to a 1D array of class labels

y\_train\_labels = np.argmax(y\_train, axis=1)

y\_test\_labels = np.argmax(y\_test, axis=1)

# Flatten the image data

X\_train\_flattened = X\_train.reshape(X\_train.shape[0], -1)

X\_test\_flattened = X\_test.reshape(X\_test.shape[0], -1)

# Create bagging ensemble (adjust n\_estimators as needed)

bagging\_models = [

dense\_model,

xception\_model,

inception\_model,

lenet\_model

]

bagging\_ensemble = BaggingClassifier(

base\_estimator=None, # Specify None as the base\_estimator since using our pre-loaded models

n\_estimators=len(bagging\_models), # Number of models in the ensemble

random\_state=42

)

# Fit the bagging ensemble

bagging\_ensemble.fit(X\_train\_flattened, y\_train\_labels)

# Evaluate the ensemble on the flattened test data

y\_pred\_bagging = bagging\_ensemble.predict(X\_test\_flattened)

# find accuracy\_score

accuracy\_bagging = accuracy\_score(y\_test\_labels, y\_pred\_bagging)

print(f"Accuracy of the Bagging Ensemble of Drishti-GS1: {accuracy\_bagging:.4f}")

**Code for RIM-ONE dataset with DenseNet, Xception, LeNet and Inception-V3.**

#Importing packages

import os

import random

import numpy as np

import seaborn as sns

from PIL import Image

from keras.models import Model

import matplotlib.pyplot as plt

from keras.models import Sequential

from imblearn.over\_sampling import SMOTE

from keras.callbacks import EarlyStopping

from sklearn.metrics import roc\_curve, auc

from sklearn.metrics import confusion\_matrix

from keras.applications.xception import Xception

from keras.applications.densenet import DenseNet121

from sklearn.model\_selection import train\_test\_split

from keras.applications.inception\_v3 import InceptionV3

from keras.preprocessing.image import ImageDataGenerator

from keras.layers import Conv2D, AveragePooling2D, Flatten, Dense

from sklearn.metrics import precision\_score, recall\_score, f1\_score

from keras.layers import Input, Conv2D, MaxPooling2D, UpSampling2D, concatenate

from google.colab import drive

drive.mount('/content/drive')

# Define early stopping callback

early\_stopping = EarlyStopping(monitor='val\_loss', patience=3, verbose=1, restore\_best\_weights=True)

# Set the dimensions of the input images

img\_width, img\_height = 224, 224

# Set the number of classes

num\_classes = 2

# Set the path to the directory containing the images

data\_path = "/content/drive/MyDrive/Rimone\_sp"

# Load the image data and labels

X = []

y = []

for label, folder\_name in enumerate(["no", "yes"]):

folder\_path = os.path.join(data\_path, folder\_name)

for filename in os.listdir(folder\_path):

image = Image.open(os.path.join(folder\_path, filename))

image = image.resize((img\_width, img\_height))

image = np.array(image)

X.append(image)

y.append(label)

# Set a fixed random seed for reproducibility

seed = 42

random.seed(seed)

np.random.seed(seed)

# Convert the image data and labels to NumPy arrays

X = np.array(X)

y = np.array(y)

# Apply SMOTE to balance the data

smote = SMOTE(random\_state=42)

X, y = smote.fit\_resample(X.reshape(X.shape[0], -1), y)

# Reshape X back to 3D format

X = X.reshape(X.shape[0], img\_width, img\_height, 3)

# Split the data into training and testing sets

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

# Normalize the image data to the range [0, 1]

X\_train = X\_train.astype('float32') / 255.0

X\_test = X\_test.astype('float32') / 255.0

# Convert the labels to one-hot encoding

y\_train = np.eye(num\_classes)[y\_train]

y\_test = np.eye(num\_classes)[y\_test]

# Create an ImageDataGenerator for data augmentation

datagen = ImageDataGenerator(

rotation\_range=10,

width\_shift\_range=0.1,

height\_shift\_range=0.1,

shear\_range=0.1,

zoom\_range=0.1,

horizontal\_flip=True,

vertical\_flip=True

)

# Create the DenseNet121 model

base\_dense\_model = DenseNet121(include\_top=False, weights='imagenet', input\_shape=(img\_width, img\_height, 3))

# Create the Xception model

base\_xception\_model = Xception(include\_top=False, weights='imagenet', input\_shape=(img\_width, img\_height, 3))

# Load the InceptionV3 model without the top classification layer

base\_inception\_model = InceptionV3(weights='imagenet', include\_top=False, input\_shape=(img\_width, img\_height, 3))

# Add custom classification layers on top of the base dense model

dense\_model = Sequential()

dense\_model.add(base\_dense\_model)

dense\_model.add(Flatten())

dense\_model.add(Dense(256, activation='relu'))

dense\_model.add(Dense(num\_classes, activation='softmax'))

# Add custom classification layers on top of the base inception model

inception\_model = Sequential()

inception\_model.add(base\_inception\_model)

inception\_model.add(Flatten())

inception\_model.add(Dense(256, activation='relu'))

inception\_model.add(Dense(num\_classes, activation='softmax'))

# Add custom classification layers on top of the Xception base model

xception\_model = Sequential()

xception\_model.add(base\_xception\_model)

xception\_model.add(Flatten())

xception\_model.add(Dense(256, activation='relu'))

xception\_model.add(Dense(num\_classes, activation='softmax'))

# Create the LeNet-5 model

lenet\_model = Sequential()

lenet\_model.add(Conv2D(6, kernel\_size=(5, 5), activation='relu', input\_shape=(img\_width, img\_height, 3)))

lenet\_model.add(AveragePooling2D(pool\_size=(2, 2)))

lenet\_model.add(Conv2D(16, kernel\_size=(5, 5), activation='relu'))

lenet\_model.add(AveragePooling2D(pool\_size=(2, 2)))

lenet\_model.add(Conv2D(32, kernel\_size=(5, 5), activation='relu'))

lenet\_model.add(AveragePooling2D(pool\_size=(2, 2)))

lenet\_model.add(Flatten())

lenet\_model.add(Dense(120, activation='relu'))

lenet\_model.add(Dense(84, activation='relu'))

lenet\_model.add(Dense(num\_classes, activation='softmax'))

# Compile the dense model

dense\_model.compile(loss='binary\_crossentropy', optimizer='adam', metrics=['accuracy'])

# Compile the xception model

xception\_model.compile(loss='binary\_crossentropy', optimizer='adam', metrics=['accuracy'])

# Compile the lenet model

lenet\_model.compile(loss='binary\_crossentropy', optimizer='adam', metrics=['accuracy'])

# Compile the inception model

inception\_model.compile(loss='binary\_crossentropy', optimizer='adam', metrics=['accuracy'])

# Training the DenseNet model

dense\_model.fit(

datagen.flow(X\_train, y\_train, batch\_size=32, seed = seed),

steps\_per\_epoch=len(X\_train) // 32,

epochs=10,

validation\_data=(X\_test, y\_test),

callbacks=[early\_stopping]

)

# Train the xception model

xception\_model.fit(

datagen.flow(X\_train, y\_train, batch\_size=32, seed = seed),

steps\_per\_epoch=len(X\_train) // 32,

epochs=10,

validation\_data=(X\_test, y\_test),

callbacks=[early\_stopping]

)

# Fit the model with data augmentation

lenet\_model.fit(

datagen.flow(X\_train, y\_train, batch\_size=32, seed = seed),

steps\_per\_epoch=len(X\_train) // 32,

epochs=10,

validation\_data=(X\_test, y\_test),

callbacks=[early\_stopping]

)

# Train the GoogleNet model

inception\_model.fit(

datagen.flow(X\_train, y\_train, batch\_size=32, seed = seed),

steps\_per\_epoch=len(X\_train) // 32,

epochs=10,

validation\_data=(X\_test, y\_test),

callbacks=[early\_stopping]

)

# testing data loss and accuracy on the test data

loss\_value, accuracy\_val = xception\_model.evaluate(X\_test, y\_test)

print(f"testing data loss\_value of xception: {loss\_value:.4f}")

print(f"Testing data accuracy\_val of xception: {accuracy\_val:.4f}")

# testing data loss\_value and accuracy\_val on the test data

loss\_value, accuracy\_val = dense\_model.evaluate(X\_test, y\_test)

print(f"testing data loss\_value of densenet: {loss\_value:.4f}")

print(f"Testing data accuracy\_val of densenet: {accuracy\_val:.4f}")

# testing data loss\_value and accuracy\_val on the test data

loss\_value, accuracy\_val = inception\_model.evaluate(X\_test, y\_test)

print(f"testing data loss\_value of inception: {loss\_value:.4f}")

print(f"Testing data accuracy\_val of inception: {accuracy\_val:.4f}")

# testing data loss\_value and accuracy\_val on the test data

loss\_value, accuracy\_val = lenet\_model.evaluate(X\_test, y\_test)

print(f"testing data loss\_value of lenet\_model: {loss\_value:.4f}")

print(f"Testing data accuracy\_val of lenet\_model: {accuracy\_val:.4f}")

# 1. Evaluate the dense model on the test data

y\_pre\_dense = dense\_model.predict(X\_test)

y\_pre\_dense\_classes = np.argmax(y\_pre\_dense, axis=1)

y\_tru\_dense = np.argmax(y\_test, axis=1)

# find precision, recall, and F1 score for dense

precision\_dense = precision\_score(y\_tru\_dense, y\_pre\_dense\_classes)

recall\_dense = recall\_score(y\_tru\_dense, y\_pre\_dense\_classes)

f1\_dense = f1\_score(y\_tru\_dense, y\_pre\_dense\_classes)

# 2. Evaluate the Inception model on the test data

y\_pre\_inception = inception\_model.predict(X\_test)

y\_pre\_inception\_classes = np.argmax(y\_pre\_inception, axis=1)

y\_tru\_inception = np.argmax(y\_test, axis=1)

# find precision, recall, and F1 score for Inception

precision\_inception = precision\_score(y\_tru\_inception, y\_pre\_inception\_classes)

recall\_inception = recall\_score(y\_tru\_inception, y\_pre\_inception\_classes)

f1\_inception = f1\_score(y\_tru\_inception, y\_pre\_inception\_classes)

# 3. Evaluate the Xception model on the test data

y\_pre\_xception = xception\_model.predict(X\_test)

y\_pre\_xception\_classes = np.argmax(y\_pre\_xception, axis=1)

y\_tru\_xception = np.argmax(y\_test, axis=1)

# find precision, recall, and F1 score for Xception

precision\_xception = precision\_score(y\_tru\_xception, y\_pre\_xception\_classes)

recall\_xception = recall\_score(y\_tru\_xception, y\_pre\_xception\_classes)

f1\_xception = f1\_score(y\_tru\_xception, y\_pre\_xception\_classes)

# 4. Evaluate the LeNet model on the test data

y\_pre\_lenet = lenet\_model.predict(X\_test)

y\_pre\_lenet\_classes = np.argmax(y\_pre\_lenet, axis=1)

y\_tru\_lenet = np.argmax(y\_test, axis=1)

# find precision, recall, and F1 score for LeNet

precision\_lenet = precision\_score(y\_tru\_lenet, y\_pre\_lenet\_classes)

recall\_lenet = recall\_score(y\_tru\_lenet, y\_pre\_lenet\_classes)

f1\_lenet = f1\_score(y\_tru\_lenet, y\_pre\_lenet\_classes)

# Obtain the predicted classes

y\_pre\_classes\_dense = np.argmax(y\_pre\_dense, axis=1)

y\_tru\_classes\_dense = np.argmax(y\_test, axis=1)

# Compute the confusion matrix

cm = confusion\_matrix(y\_tru\_classes\_dense, y\_pre\_classes\_dense)

# Create a function to plot the confusion matrix

def plot\_confusion\_matrix(cm, classes):

plt.imshow(cm, interpolation='nearest', cmap=plt.cm.Blues)

plt.colorbar()

tick\_marks = np.arange(len(classes))

plt.xticks(tick\_marks, classes, rotation=45)

plt.yticks(tick\_marks, classes)

plt.xlabel('Predicted Label')

plt.ylabel('True Label')

# Plot the confusion matrix

plt.figure()

plot\_confusion\_matrix(cm, classes=class\_names)

plt.title('Confusion Matrix of dense')

plt.show()

#Precision, Recall, F1 Score of DenseNet Model

print(f"precision of dense: {precision\_dense:.4f}")

print(f"recall dense: {recall\_dense:.4f}")

print(f"F1 Score of dense: {f1\_dense:.4f}")

#Precision, Recall, F1 Score of Xception Model

print(f"Precision of xception: {precision\_xception:.4f}")

print(f"Recall of xception: {recall\_xception:.4f}")

print(f"F1 Score of xception: {f1\_xception:.4f}")

#Precision, Recall, F1 Score of LeNet Model

print(f"Precision of Lenet: {precision\_lenet:.4f}")

print(f"Recall of Lenet: {recall\_lenet:.4f}")

print(f"F1 Score of Lenet: {f1\_lenet:.4f}")

#Precision, Recall, F1 Score of Inception Model

print(f"Precision of Inception: {precision\_inception:.4f}")

print(f"Recall of Inception: {recall\_inception:.4f}")

print(f"F1 Score of Inception: {f1\_inception:.4f}")

# Precision, Recall, and F1-score values for each model

models = ['Dense', 'Xception', 'LeNet', 'Inception']

precision\_values = [precision\_dense, precision\_xception, precision\_lenet, precision\_inception]

recall\_values = [recall\_dense, recall\_xception, recall\_lenet, recall\_inception]

f1\_values = [f1\_dense, f1\_xception, f1\_lenet, f1\_inception]

# Define different shades of blue for each model

colors = ['#1f77b4', '#1f4e77', '#1f3552', '#1f2235']

# Initialize plots

plt.figure(figsize=(12, 6))

# Plot Precision scores

plt.subplot(1, 3, 1)

plt.bar(models, precision\_values, color=colors)

plt.xlabel('Models')

plt.ylabel('Precision')

plt.title('Precision Scores')

# Plot Recall scores

plt.subplot(1, 3, 2)

plt.bar(models, recall\_values, color=colors)

plt.xlabel('Models')

plt.ylabel('Recall')

plt.title('Recall Scores')

# Plot F1 scores

plt.subplot(1, 3, 3)

plt.bar(models, f1\_values, color=colors)

plt.xlabel('Models')

plt.ylabel('F1-Score')

plt.title('F1-Score Scores')

plt.tight\_layout()

plt.show()

#ROC and AUC Curve

# Obtain the predicted probabilities for the positive class of DenseNet model

y\_pre\_prob\_dense = dense\_model.predict(X\_test)[:, 1]

# Obtain the predicted probabilities for the positive class of Xception Model

y\_pre\_prob\_xception = xception\_model.predict(X\_test)[:, 1]

# Obtain the predicted probabilities for the positive class of LeNet model

y\_pre\_prob\_lenet = lenet\_model.predict(X\_test)[:, 1]

# Obtain the predicted probabilities for the positive class of Inception Model

y\_pre\_prob\_inception = inception\_model.predict(X\_test)[:, 1]

# Compute the false positive rate, true positive rate, and classification threshold of dense

fpr\_dense, tpr\_dense, thresholds\_dense = roc\_curve(y\_tru\_dense, y\_pre\_prob\_dense)

# Compute the AUC score

roc\_auc = auc(fpr\_dense, tpr\_dense)

# Plot the ROC curve

plt.figure()

plt.plot(fpr\_dense, tpr\_dense, color='darkorange', lw=2, label='ROC curve (AUC = %0.2f)' % roc\_auc)

plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')

plt.xlim([0.0, 1.0])

plt.ylim([0.0, 1.05])

plt.xlabel('False Positive Rate')

plt.ylabel('True Positive Rate')

plt.title('Receiver Operating Characteristic of dense')

plt.legend(loc="lower right")

plt.show()

# Compute the false positive rate, true positive rate, and classification threshold of Xception

fpr\_xception, tpr\_xception, thresholds\_xception = roc\_curve(y\_tru\_xception, y\_pre\_prob\_xception)

# Compute the AUC score

roc\_auc = auc(fpr\_xception, tpr\_xception)

# Plot the ROC curve

plt.figure()

plt.plot(fpr\_xception, tpr\_xception, color='darkorange', lw=2, label='ROC curve (AUC = %0.2f)' % roc\_auc)

plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')

plt.xlim([0.0, 1.0])

plt.ylim([0.0, 1.05])

plt.xlabel('False Positive Rate')

plt.ylabel('True Positive Rate')

plt.title('Receiver Operating Characteristic')

plt.legend(loc="lower right")

plt.show()

# Compute the false positive rate, true positive rate, and classification threshold of Inception

fpr\_inception, tpr\_inception, thresholds\_inception = roc\_curve(y\_tru\_inception, y\_pre\_prob\_inception)

# Compute the AUC score

roc\_auc = auc(fpr\_inception, tpr\_inception)

# Plot the ROC curve

plt.figure()

plt.plot(fpr\_inception, tpr\_inception, color='darkorange', lw=2, label='ROC curve (AUC = %0.2f)' % roc\_auc)

plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')

plt.xlim([0.0, 1.0])

plt.ylim([0.0, 1.05])

plt.xlabel('False Positive Rate')

plt.ylabel('True Positive Rate')

plt.title('Receiver Operating Characteristic')

plt.legend(loc="lower right")

plt.show()

# Compute the false positive rate, true positive rate, and classification threshold of LeNet.

fpr\_lenet, tpr\_lenet, thresholds\_lenet = roc\_curve(y\_tru\_lenet, y\_pre\_prob\_lenet)

# Compute the AUC score

roc\_auc = auc(fpr\_lenet, tpr\_lenet)

# Plot the ROC curve

plt.figure()

plt.plot(fpr\_lenet, tpr\_lenet, color='darkorange', lw=2, label='ROC curve (AUC = %0.2f)' % roc\_auc)

plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')

plt.xlim([0.0, 1.0])

plt.ylim([0.0, 1.05])

plt.xlabel('False Positive Rate')

plt.ylabel('True Positive Rate')

plt.title('Receiver Operating Characteristic')

plt.legend(loc="lower right")

plt.show()

#ROC curve for all models for RIM-ONE dataset.

# Data for all models

models = ['DenseNet', 'Xception', 'Inception', 'LeNet']

fprs = [fpr\_dense, fpr\_xception, fpr\_inception, fpr\_lenet]

tprs = [tpr\_dense, tpr\_xception, tpr\_inception, tpr\_lenet]

auc\_scores = [auc(fpr, tpr) for fpr, tpr in zip(fprs, tprs)]

# Plotting ROC curves for all models

plt.figure(figsize=(10, 6))

for model, fpr, tpr, auc\_score in zip(models, fprs, tprs, auc\_scores):

plt.plot(fpr, tpr, lw=2, label=f'{model} (AUC = {auc\_score:.2f})')

plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')

plt.xlim([0.0, 1.0])

plt.ylim([0.0, 1.05])

plt.xlabel('False Positive Rate')

plt.ylabel('True Positive Rate')

plt.title('Receiver Operating Characteristic of Different Models')

plt.legend(loc="lower right")

plt.show()

#Confusion Matrix

# Define the class labels

class\_names = ['No', 'Yes']

# Obtain the predicted classes for dense.

y\_pre\_classes\_dense = np.argmax(y\_pre\_dense, axis=1)

y\_tru\_classes\_dense = np.argmax(y\_test, axis=1)

# Compute the confusion matrix

cm = confusion\_matrix(y\_tru\_classes\_dense, y\_pre\_classes\_dense)

# Create a function to plot the confusion matrix

def plot\_confusion\_matrix(cm, classes):

plt.imshow(cm, interpolation='nearest', cmap=plt.cm.Blues)

plt.colorbar()

tick\_marks = np.arange(len(classes))

plt.xticks(tick\_marks, classes, rotation=45)

plt.yticks(tick\_marks, classes)

plt.xlabel('Predicted Label')

plt.ylabel('True Label')

# Plot the confusion matrix

plt.figure()

plot\_confusion\_matrix(cm, classes=class\_names)

plt.title('Confusion Matrix of dense')

plt.show()

# Obtain the predicted classes for lenet

y\_pre\_classes\_lenet = np.argmax(y\_pre\_lenet, axis=1)

y\_tru\_classes\_lenet = np.argmax(y\_test, axis=1)

# Compute the confusion matrix

cm = confusion\_matrix(y\_tru\_classes\_lenet, y\_pre\_classes\_lenet)

# Create a function to plot the confusion matrix

def plot\_confusion\_matrix(cm, classes):

plt.imshow(cm, interpolation='nearest', cmap=plt.cm.Blues)

plt.colorbar()

tick\_marks = np.arange(len(classes))

plt.xticks(tick\_marks, classes, rotation=45)

plt.yticks(tick\_marks, classes)

plt.xlabel('Predicted Label')

plt.ylabel('True Label')

# Plot the confusion matrix

plt.figure()

plot\_confusion\_matrix(cm, classes=class\_names)

plt.title('Confusion Matrix of lenet')

plt.show()

# Obtain the predicted classes for Xception

y\_pre\_classes\_xception = np.argmax(y\_pre\_xception, axis=1)

y\_tru\_classes\_xception = np.argmax(y\_test, axis=1)

# Compute the confusion matrix

cm = confusion\_matrix(y\_tru\_classes\_xception, y\_pre\_classes\_xception)

# Create a function to plot the confusion matrix

def plot\_confusion\_matrix(cm, classes):

plt.imshow(cm, interpolation='nearest', cmap=plt.cm.Blues)

plt.colorbar()

tick\_marks = np.arange(len(classes))

plt.xticks(tick\_marks, classes, rotation=45)

plt.yticks(tick\_marks, classes)

plt.xlabel('Predicted Label')

plt.ylabel('True Label')

# Plot the confusion matrix

plt.figure()

plot\_confusion\_matrix(cm, classes=class\_names)

plt.title('Confusion Matrix of xception')

plt.show()

# Obtain the predicted classes for Inception

y\_pre\_classes\_inception = np.argmax(y\_pre\_inception, axis=1)

y\_tru\_classes\_inception = np.argmax(y\_test, axis=1)

# Compute the confusion matrix

cm = confusion\_matrix(y\_tru\_classes\_inception, y\_pre\_classes\_inception)

# Create a function to plot the confusion matrix

def plot\_confusion\_matrix(cm, classes):

plt.imshow(cm, interpolation='nearest', cmap=plt.cm.Blues)

plt.colorbar()

tick\_marks = np.arange(len(classes))

plt.xticks(tick\_marks, classes, rotation=45)

plt.yticks(tick\_marks, classes)

plt.xlabel('Predicted Label')

plt.ylabel('True Label')

# Plot the confusion matrix

plt.figure()

plot\_confusion\_matrix(cm, classes=class\_names)

plt.title('Confusion Matrix of inception')

plt.show()

#confusion matrix for all models

cm\_xception = confusion\_matrix(y\_tru\_classes\_xception, y\_pre\_classes\_xception)

cm\_inception = confusion\_matrix(y\_tru\_classes\_inception, y\_pre\_classes\_inception)

cm\_lenet = confusion\_matrix(y\_tru\_classes\_lenet, y\_pre\_classes\_lenet)

cm\_dense = confusion\_matrix(y\_tru\_classes\_dense, y\_pre\_classes\_dense)

# List of model names

model\_names = ['Inception', 'Xception', 'LeNet', 'DenseNet']

# List of confusion matrices for each model

conf\_matrices = [

cm\_inception,

cm\_xception,

cm\_lenet,

cm\_dense

]

# Plot confusion matrices

plt.figure(figsize=(15, 10))

for i, (model\_name, conf\_matrix) in enumerate(zip(model\_names, conf\_matrices), start=1):

plt.subplot(2, 2, i)

sns.heatmap(conf\_matrix, annot=True, fmt='d', cmap='Blues',

xticklabels=["Class 0", "Class 1"], yticklabels=["Class 0", "Class 1"])

plt.title(f'Confusion Matrix - {model\_name}')

plt.xlabel('Predicted')

plt.ylabel('Actual')

plt.tight\_layout()

plt.show()

from sklearn.ensemble import BaggingClassifier

from sklearn.metrics import accuracy\_score

# Convert one-hot encoded y to a 1D array of class labels

y\_train\_labels = np.argmax(y\_train, axis=1)

y\_test\_labels = np.argmax(y\_test, axis=1)

# Flatten the image data

X\_train\_flattened = X\_train.reshape(X\_train.shape[0], -1)

X\_test\_flattened = X\_test.reshape(X\_test.shape[0], -1)

# Create bagging ensemble (adjust n\_estimators as needed)

bagging\_models = [

dense\_model,

xception\_model,

inception\_model,

lenet\_model

]

bagging\_ensemble = BaggingClassifier(

base\_estimator=None, # Specify None as the base\_estimator since using our pre-loaded models

n\_estimators=len(bagging\_models), # Number of models in the ensemble

random\_state=42

)

# Fit the bagging ensemble

bagging\_ensemble.fit(X\_train\_flattened, y\_train\_labels)

# Evaluate the ensemble on the flattened test data

y\_pred\_bagging = bagging\_ensemble.predict(X\_test\_flattened)

# find accuracy\_score

accuracy\_bagging = accuracy\_score(y\_test\_labels, y\_pred\_bagging)

print(f"Accuracy of the Bagging Ensemble of RIM-ONE: {accuracy\_bagging:.4f}")

**Code for ORIGA-R dataset with DenseNet, Xception, LeNet and Inception-V3**.

#Importing packages

import os

import random

import numpy as np

import seaborn as sns

from PIL import Image

from keras.models import Model

import matplotlib.pyplot as plt

from keras.models import Sequential

from imblearn.over\_sampling import SMOTE

from keras.callbacks import EarlyStopping

from sklearn.metrics import roc\_curve, auc

from sklearn.metrics import confusion\_matrix

from keras.applications.xception import Xception

from keras.applications.densenet import DenseNet121

from sklearn.model\_selection import train\_test\_split

from keras.applications.inception\_v3 import InceptionV3

from keras.preprocessing.image import ImageDataGenerator

from keras.layers import Conv2D, AveragePooling2D, Flatten, Dense

from sklearn.metrics import precision\_score, recall\_score, f1\_score

from keras.layers import Input, Conv2D, MaxPooling2D, UpSampling2D, concatenate

from google.colab import drive

drive.mount('/content/drive')

# Define early stopping callback

early\_stopping = EarlyStopping(monitor='val\_loss', patience=3, verbose=1, restore\_best\_weights=True)

# Set the dimensions of the input images

img\_width, img\_height = 224, 224

# Set the number of classes

num\_classes = 2

# Set the path to the directory containing the images

data\_path = "/content/drive/MyDrive/Origa"

# Load the image data and labels

X = []

y = []

for label, folder\_name in enumerate(["no", "yes"]):

folder\_path = os.path.join(data\_path, folder\_name)

for filename in os.listdir(folder\_path):

image = Image.open(os.path.join(folder\_path, filename))

image = image.resize((img\_width, img\_height))

image = np.array(image)

X.append(image)

y.append(label)

# Set a fixed random seed for reproducibility

seed = 42

random.seed(seed)

np.random.seed(seed)

# Convert the image data and labels to NumPy arrays

X = np.array(X)

y = np.array(y)

# Apply SMOTE to balance the data

smote = SMOTE(random\_state=42)

X, y = smote.fit\_resample(X.reshape(X.shape[0], -1), y)

# Reshape X back to 3D format

X = X.reshape(X.shape[0], img\_width, img\_height, 3)

# Split the data into training and testing sets

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

# Normalize the image data to the range [0, 1]

X\_train = X\_train.astype('float32') / 255.0

X\_test = X\_test.astype('float32') / 255.0

# Convert the labels to one-hot encoding

y\_train = np.eye(num\_classes)[y\_train]

y\_test = np.eye(num\_classes)[y\_test]

# Create an ImageDataGenerator for data augmentation

datagen = ImageDataGenerator(

rotation\_range=10,

width\_shift\_range=0.1,

height\_shift\_range=0.1,

shear\_range=0.1,

zoom\_range=0.1,

horizontal\_flip=True,

vertical\_flip=True

)

# Create the DenseNet121 model

base\_dense\_model = DenseNet121(include\_top=False, weights='imagenet', input\_shape=(img\_width, img\_height, 3))

# Create the Xception model

base\_xception\_model = Xception(include\_top=False, weights='imagenet', input\_shape=(img\_width, img\_height, 3))

# Load the InceptionV3 model without the top classification layer

base\_inception\_model = InceptionV3(weights='imagenet', include\_top=False, input\_shape=(img\_width, img\_height, 3))

# Add custom classification layers on top of the base dense model

dense\_model = Sequential()

dense\_model.add(base\_dense\_model)

dense\_model.add(Flatten())

dense\_model.add(Dense(256, activation='relu'))

dense\_model.add(Dense(num\_classes, activation='softmax'))

# Add custom classification layers on top of the base inception model

inception\_model = Sequential()

inception\_model.add(base\_inception\_model)

inception\_model.add(Flatten())

inception\_model.add(Dense(256, activation='relu'))

inception\_model.add(Dense(num\_classes, activation='softmax'))

# Add custom classification layers on top of the Xception base model

xception\_model = Sequential()

xception\_model.add(base\_xception\_model)

xception\_model.add(Flatten())

xception\_model.add(Dense(256, activation='relu'))

xception\_model.add(Dense(num\_classes, activation='softmax'))

# Create the LeNet-5 model

lenet\_model = Sequential()

lenet\_model.add(Conv2D(6, kernel\_size=(5, 5), activation='relu', input\_shape=(img\_width, img\_height, 3)))

lenet\_model.add(AveragePooling2D(pool\_size=(2, 2)))

lenet\_model.add(Conv2D(16, kernel\_size=(5, 5), activation='relu'))

lenet\_model.add(AveragePooling2D(pool\_size=(2, 2)))

lenet\_model.add(Conv2D(32, kernel\_size=(5, 5), activation='relu'))

lenet\_model.add(AveragePooling2D(pool\_size=(2, 2)))

lenet\_model.add(Flatten())

lenet\_model.add(Dense(120, activation='relu'))

lenet\_model.add(Dense(84, activation='relu'))

lenet\_model.add(Dense(num\_classes, activation='softmax'))

# Compile the dense model

dense\_model.compile(loss='binary\_crossentropy', optimizer='adam', metrics=['accuracy'])

# Compile the xception model

xception\_model.compile(loss='binary\_crossentropy', optimizer='adam', metrics=['accuracy'])

# Compile the lenet model

lenet\_model.compile(loss='binary\_crossentropy', optimizer='adam', metrics=['accuracy'])

# Compile the inception model

inception\_model.compile(loss='binary\_crossentropy', optimizer='adam', metrics=['accuracy'])

# Training the DenseNet model

dense\_model.fit(

datagen.flow(X\_train, y\_train, batch\_size=32, seed = seed),

steps\_per\_epoch=len(X\_train) // 32,

epochs=10,

validation\_data=(X\_test, y\_test),

callbacks=[early\_stopping]

)

# Train the xception model

xception\_model.fit(

datagen.flow(X\_train, y\_train, batch\_size=32, seed = seed),

steps\_per\_epoch=len(X\_train) // 32,

epochs=10,

validation\_data=(X\_test, y\_test),

callbacks=[early\_stopping]

)

# Fit the model with data augmentation

lenet\_model.fit(

datagen.flow(X\_train, y\_train, batch\_size=32, seed = seed),

steps\_per\_epoch=len(X\_train) // 32,

epochs=10,

validation\_data=(X\_test, y\_test),

callbacks=[early\_stopping]

)

# Train the GoogleNet model

inception\_model.fit(

datagen.flow(X\_train, y\_train, batch\_size=32, seed = seed),

steps\_per\_epoch=len(X\_train) // 32,

epochs=10,

validation\_data=(X\_test, y\_test),

callbacks=[early\_stopping]

)

# testing data loss and accuracy on the test data

loss\_value, accuracy\_val = xception\_model.evaluate(X\_test, y\_test)

print(f"testing data loss\_value of xception: {loss\_value:.4f}")

print(f"Testing data accuracy\_val of xception: {accuracy\_val:.4f}")

# testing data loss\_value and accuracy\_val on the test data

loss\_value, accuracy\_val = dense\_model.evaluate(X\_test, y\_test)

print(f"testing data loss\_value of densenet: {loss\_value:.4f}")

print(f"Testing data accuracy\_val of densenet: {accuracy\_val:.4f}")

# testing data loss\_value and accuracy\_val on the test data

loss\_value, accuracy\_val = inception\_model.evaluate(X\_test, y\_test)

print(f"testing data loss\_value of inception: {loss\_value:.4f}")

print(f"Testing data accuracy\_val of inception: {accuracy\_val:.4f}")

# testing data loss\_value and accuracy\_val on the test data

loss\_value, accuracy\_val = lenet\_model.evaluate(X\_test, y\_test)

print(f"testing data loss\_value of lenet\_model: {loss\_value:.4f}")

print(f"Testing data accuracy\_val of lenet\_model: {accuracy\_val:.4f}")

# 1. Evaluate the dense model on the test data

y\_pre\_dense = dense\_model.predict(X\_test)

y\_pre\_dense\_classes = np.argmax(y\_pre\_dense, axis=1)

y\_tru\_dense = np.argmax(y\_test, axis=1)

# find precision, recall, and F1 score for dense

precision\_dense = precision\_score(y\_tru\_dense, y\_pre\_dense\_classes)

recall\_dense = recall\_score(y\_tru\_dense, y\_pre\_dense\_classes)

f1\_dense = f1\_score(y\_tru\_dense, y\_pre\_dense\_classes)

# 2. Evaluate the Inception model on the test data

y\_pre\_inception = inception\_model.predict(X\_test)

y\_pre\_inception\_classes = np.argmax(y\_pre\_inception, axis=1)

y\_tru\_inception = np.argmax(y\_test, axis=1)

# find precision, recall, and F1 score for Inception

precision\_inception = precision\_score(y\_tru\_inception, y\_pre\_inception\_classes)

recall\_inception = recall\_score(y\_tru\_inception, y\_pre\_inception\_classes)

f1\_inception = f1\_score(y\_tru\_inception, y\_pre\_inception\_classes)

# 3. Evaluate the Xception model on the test data

y\_pre\_xception = xception\_model.predict(X\_test)

y\_pre\_xception\_classes = np.argmax(y\_pre\_xception, axis=1)

y\_tru\_xception = np.argmax(y\_test, axis=1)

# find precision, recall, and F1 score for Xception

precision\_xception = precision\_score(y\_tru\_xception, y\_pre\_xception\_classes)

recall\_xception = recall\_score(y\_tru\_xception, y\_pre\_xception\_classes)

f1\_xception = f1\_score(y\_tru\_xception, y\_pre\_xception\_classes)

# 4. Evaluate the LeNet model on the test data

y\_pre\_lenet = lenet\_model.predict(X\_test)

y\_pre\_lenet\_classes = np.argmax(y\_pre\_lenet, axis=1)

y\_tru\_lenet = np.argmax(y\_test, axis=1)

# find precision, recall, and F1 score for LeNet

precision\_lenet = precision\_score(y\_tru\_lenet, y\_pre\_lenet\_classes)

recall\_lenet = recall\_score(y\_tru\_lenet, y\_pre\_lenet\_classes)

f1\_lenet = f1\_score(y\_tru\_lenet, y\_pre\_lenet\_classes)

# Obtain the predicted classes

y\_pre\_classes\_dense = np.argmax(y\_pre\_dense, axis=1)

y\_tru\_classes\_dense = np.argmax(y\_test, axis=1)

# Compute the confusion matrix

cm = confusion\_matrix(y\_tru\_classes\_dense, y\_pre\_classes\_dense)

# Create a function to plot the confusion matrix

def plot\_confusion\_matrix(cm, classes):

plt.imshow(cm, interpolation='nearest', cmap=plt.cm.Blues)

plt.colorbar()

tick\_marks = np.arange(len(classes))

plt.xticks(tick\_marks, classes, rotation=45)

plt.yticks(tick\_marks, classes)

plt.xlabel('Predicted Label')

plt.ylabel('True Label')

# Plot the confusion matrix

plt.figure()

plot\_confusion\_matrix(cm, classes=class\_names)

plt.title('Confusion Matrix of dense')

plt.show()

#Precision, Recall, F1 Score of DenseNet Model

print(f"precision of dense: {precision\_dense:.4f}")

print(f"recall dense: {recall\_dense:.4f}")

print(f"F1 Score of dense: {f1\_dense:.4f}")

#Precision, Recall, F1 Score of Xception Model

print(f"Precision of xception: {precision\_xception:.4f}")

print(f"Recall of xception: {recall\_xception:.4f}")

print(f"F1 Score of xception: {f1\_xception:.4f}")

#Precision, Recall, F1 Score of LeNet Model

print(f"Precision of Lenet: {precision\_lenet:.4f}")

print(f"Recall of Lenet: {recall\_lenet:.4f}")

print(f"F1 Score of Lenet: {f1\_lenet:.4f}")

#Precision, Recall, F1 Score of Inception Model

print(f"Precision of Inception: {precision\_inception:.4f}")

print(f"Recall of Inception: {recall\_inception:.4f}")

print(f"F1 Score of Inception: {f1\_inception:.4f}")

# Precision, Recall, and F1-score values for each model

models = ['Dense', 'Xception', 'LeNet', 'Inception']

precision\_values = [precision\_dense, precision\_xception, precision\_lenet, precision\_inception]

recall\_values = [recall\_dense, recall\_xception, recall\_lenet, recall\_inception]

f1\_values = [f1\_dense, f1\_xception, f1\_lenet, f1\_inception]

# Define different shades of blue for each model

colors = ['#1f77b4', '#1f4e77', '#1f3552', '#1f2235']

# Initialize plots

plt.figure(figsize=(12, 6))

# Plot Precision scores

plt.subplot(1, 3, 1)

plt.bar(models, precision\_values, color=colors)

plt.xlabel('Models')

plt.ylabel('Precision')

plt.title('Precision Scores')

# Plot Recall scores

plt.subplot(1, 3, 2)

plt.bar(models, recall\_values, color=colors)

plt.xlabel('Models')

plt.ylabel('Recall')

plt.title('Recall Scores')

# Plot F1 scores

plt.subplot(1, 3, 3)

plt.bar(models, f1\_values, color=colors)

plt.xlabel('Models')

plt.ylabel('F1-Score')

plt.title('F1-Score Scores')

plt.tight\_layout()

plt.show()

#ROC and AUC Curve

# Obtain the predicted probabilities for the positive class of DenseNet model

y\_pre\_prob\_dense = dense\_model.predict(X\_test)[:, 1]

# Obtain the predicted probabilities for the positive class of Xception Model

y\_pre\_prob\_xception = xception\_model.predict(X\_test)[:, 1]

# Obtain the predicted probabilities for the positive class of LeNet model

y\_pre\_prob\_lenet = lenet\_model.predict(X\_test)[:, 1]

# Obtain the predicted probabilities for the positive class of Inception Model

y\_pre\_prob\_inception = inception\_model.predict(X\_test)[:, 1]

# Compute the false positive rate, true positive rate, and classification threshold of dense

fpr\_dense, tpr\_dense, thresholds\_dense = roc\_curve(y\_tru\_dense, y\_pre\_prob\_dense)

# Compute the AUC score

roc\_auc = auc(fpr\_dense, tpr\_dense)

# Plot the ROC curve

plt.figure()

plt.plot(fpr\_dense, tpr\_dense, color='darkorange', lw=2, label='ROC curve (AUC = %0.2f)' % roc\_auc)

plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')

plt.xlim([0.0, 1.0])

plt.ylim([0.0, 1.05])

plt.xlabel('False Positive Rate')

plt.ylabel('True Positive Rate')

plt.title('Receiver Operating Characteristic of dense')

plt.legend(loc="lower right")

plt.show()

# Compute the false positive rate, true positive rate, and classification threshold of Xception

fpr\_xception, tpr\_xception, thresholds\_xception = roc\_curve(y\_tru\_xception, y\_pre\_prob\_xception)

# Compute the AUC score

roc\_auc = auc(fpr\_xception, tpr\_xception)

# Plot the ROC curve

plt.figure()

plt.plot(fpr\_xception, tpr\_xception, color='darkorange', lw=2, label='ROC curve (AUC = %0.2f)' % roc\_auc)

plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')

plt.xlim([0.0, 1.0])

plt.ylim([0.0, 1.05])

plt.xlabel('False Positive Rate')

plt.ylabel('True Positive Rate')

plt.title('Receiver Operating Characteristic')

plt.legend(loc="lower right")

plt.show()

# Compute the false positive rate, true positive rate, and classification threshold of Inception

fpr\_inception, tpr\_inception, thresholds\_inception = roc\_curve(y\_tru\_inception, y\_pre\_prob\_inception)

# Compute the AUC score

roc\_auc = auc(fpr\_inception, tpr\_inception)

# Plot the ROC curve

plt.figure()

plt.plot(fpr\_inception, tpr\_inception, color='darkorange', lw=2, label='ROC curve (AUC = %0.2f)' % roc\_auc)

plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')

plt.xlim([0.0, 1.0])

plt.ylim([0.0, 1.05])

plt.xlabel('False Positive Rate')

plt.ylabel('True Positive Rate')

plt.title('Receiver Operating Characteristic')

plt.legend(loc="lower right")

plt.show()

# Compute the false positive rate, true positive rate, and classification threshold of LeNet.

fpr\_lenet, tpr\_lenet, thresholds\_lenet = roc\_curve(y\_tru\_lenet, y\_pre\_prob\_lenet)

# Compute the AUC score

roc\_auc = auc(fpr\_lenet, tpr\_lenet)

# Plot the ROC curve

plt.figure()

plt.plot(fpr\_lenet, tpr\_lenet, color='darkorange', lw=2, label='ROC curve (AUC = %0.2f)' % roc\_auc)

plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')

plt.xlim([0.0, 1.0])

plt.ylim([0.0, 1.05])

plt.xlabel('False Positive Rate')

plt.ylabel('True Positive Rate')

plt.title('Receiver Operating Characteristic')

plt.legend(loc="lower right")

plt.show()

#ROC curve for all models for ORIGA dataset.

# Data for all models

models = ['DenseNet', 'Xception', 'Inception', 'LeNet']

fprs = [fpr\_dense, fpr\_xception, fpr\_inception, fpr\_lenet]

tprs = [tpr\_dense, tpr\_xception, tpr\_inception, tpr\_lenet]

auc\_scores = [auc(fpr, tpr) for fpr, tpr in zip(fprs, tprs)]

# Plotting ROC curves for all models

plt.figure(figsize=(10, 6))

for model, fpr, tpr, auc\_score in zip(models, fprs, tprs, auc\_scores):

plt.plot(fpr, tpr, lw=2, label=f'{model} (AUC = {auc\_score:.2f})')

plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')

plt.xlim([0.0, 1.0])

plt.ylim([0.0, 1.05])

plt.xlabel('False Positive Rate')

plt.ylabel('True Positive Rate')

plt.title('Receiver Operating Characteristic of Different Models')

plt.legend(loc="lower right")

plt.show()

#Confusion Matrix

# Define the class labels

class\_names = ['No', 'Yes']

# Obtain the predicted classes for dense.

y\_pre\_classes\_dense = np.argmax(y\_pre\_dense, axis=1)

y\_tru\_classes\_dense = np.argmax(y\_test, axis=1)

# Compute the confusion matrix

cm = confusion\_matrix(y\_tru\_classes\_dense, y\_pre\_classes\_dense)

# Create a function to plot the confusion matrix

def plot\_confusion\_matrix(cm, classes):

plt.imshow(cm, interpolation='nearest', cmap=plt.cm.Blues)

plt.colorbar()

tick\_marks = np.arange(len(classes))

plt.xticks(tick\_marks, classes, rotation=45)

plt.yticks(tick\_marks, classes)

plt.xlabel('Predicted Label')

plt.ylabel('True Label')

# Plot the confusion matrix

plt.figure()

plot\_confusion\_matrix(cm, classes=class\_names)

plt.title('Confusion Matrix of dense')

plt.show()

# Obtain the predicted classes for lenet

y\_pre\_classes\_lenet = np.argmax(y\_pre\_lenet, axis=1)

y\_tru\_classes\_lenet = np.argmax(y\_test, axis=1)

# Compute the confusion matrix

cm = confusion\_matrix(y\_tru\_classes\_lenet, y\_pre\_classes\_lenet)

# Create a function to plot the confusion matrix

def plot\_confusion\_matrix(cm, classes):

plt.imshow(cm, interpolation='nearest', cmap=plt.cm.Blues)

plt.colorbar()

tick\_marks = np.arange(len(classes))

plt.xticks(tick\_marks, classes, rotation=45)

plt.yticks(tick\_marks, classes)

plt.xlabel('Predicted Label')

plt.ylabel('True Label')

# Plot the confusion matrix

plt.figure()

plot\_confusion\_matrix(cm, classes=class\_names)

plt.title('Confusion Matrix of lenet')

plt.show()

# Obtain the predicted classes for Xception

y\_pre\_classes\_xception = np.argmax(y\_pre\_xception, axis=1)

y\_tru\_classes\_xception = np.argmax(y\_test, axis=1)

# Compute the confusion matrix

cm = confusion\_matrix(y\_tru\_classes\_xception, y\_pre\_classes\_xception)

# Create a function to plot the confusion matrix

def plot\_confusion\_matrix(cm, classes):

plt.imshow(cm, interpolation='nearest', cmap=plt.cm.Blues)

plt.colorbar()

tick\_marks = np.arange(len(classes))

plt.xticks(tick\_marks, classes, rotation=45)

plt.yticks(tick\_marks, classes)

plt.xlabel('Predicted Label')

plt.ylabel('True Label')

# Plot the confusion matrix

plt.figure()

plot\_confusion\_matrix(cm, classes=class\_names)

plt.title('Confusion Matrix of xception')

plt.show()

# Obtain the predicted classes for Inception

y\_pre\_classes\_inception = np.argmax(y\_pre\_inception, axis=1)

y\_tru\_classes\_inception = np.argmax(y\_test, axis=1)

# Compute the confusion matrix

cm = confusion\_matrix(y\_tru\_classes\_inception, y\_pre\_classes\_inception)

# Create a function to plot the confusion matrix

def plot\_confusion\_matrix(cm, classes):

plt.imshow(cm, interpolation='nearest', cmap=plt.cm.Blues)

plt.colorbar()

tick\_marks = np.arange(len(classes))

plt.xticks(tick\_marks, classes, rotation=45)

plt.yticks(tick\_marks, classes)

plt.xlabel('Predicted Label')

plt.ylabel('True Label')

# Plot the confusion matrix

plt.figure()

plot\_confusion\_matrix(cm, classes=class\_names)

plt.title('Confusion Matrix of inception')

plt.show()

#confusion matrix for all models

cm\_xception = confusion\_matrix(y\_tru\_classes\_xception, y\_pre\_classes\_xception)

cm\_inception = confusion\_matrix(y\_tru\_classes\_inception, y\_pre\_classes\_inception)

cm\_lenet = confusion\_matrix(y\_tru\_classes\_lenet, y\_pre\_classes\_lenet)

cm\_dense = confusion\_matrix(y\_tru\_classes\_dense, y\_pre\_classes\_dense)

# List of model names

model\_names = ['Inception', 'Xception', 'LeNet', 'DenseNet']

# List of confusion matrices for each model

conf\_matrices = [

cm\_inception,

cm\_xception,

cm\_lenet,

cm\_dense

]

# Plot confusion matrices

plt.figure(figsize=(15, 10))

for i, (model\_name, conf\_matrix) in enumerate(zip(model\_names, conf\_matrices), start=1):

plt.subplot(2, 2, i)

sns.heatmap(conf\_matrix, annot=True, fmt='d', cmap='Blues',

xticklabels=["Class 0", "Class 1"], yticklabels=["Class 0", "Class 1"])

plt.title(f'Confusion Matrix - {model\_name}')

plt.xlabel('Predicted')

plt.ylabel('Actual')

plt.tight\_layout()

plt.show()

from sklearn.ensemble import BaggingClassifier

from sklearn.metrics import accuracy\_score

# Convert one-hot encoded y to a 1D array of class labels

y\_train\_labels = np.argmax(y\_train, axis=1)

y\_test\_labels = np.argmax(y\_test, axis=1)

# Flatten the image data

X\_train\_flattened = X\_train.reshape(X\_train.shape[0], -1)

X\_test\_flattened = X\_test.reshape(X\_test.shape[0], -1)

# Create bagging ensemble (adjust n\_estimators as needed)

bagging\_models = [

dense\_model,

xception\_model,

inception\_model,

lenet\_model

]

bagging\_ensemble = BaggingClassifier(

base\_estimator=None, # Specify None as the base\_estimator since using our pre-loaded models

n\_estimators=len(bagging\_models), # Number of models in the ensemble

random\_state=42

)

# Fit the bagging ensemble

bagging\_ensemble.fit(X\_train\_flattened, y\_train\_labels)

# Evaluate the ensemble on the flattened test data

y\_pred\_bagging = bagging\_ensemble.predict(X\_test\_flattened)

# find accuracy\_score

accuracy\_bagging = accuracy\_score(y\_test\_labels, y\_pred\_bagging)

print(f"Accuracy of the Bagging Ensemble of origa: {accuracy\_bagging:.4f}")