Diabetic Retinopathy Detection

A major project report submitted in partial fulfillment of the requirement
for the award of degree of

Bachelor of Technology

in

Computer Science & Engineering / Information Technology

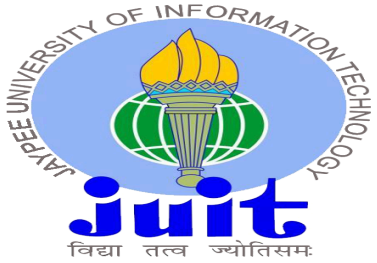
Submitted by

Ambikesh jha(201453)

Animesh(201567)

Under the guidance & supervision of

Dr. Ekta Gandotra



Department of Computer Science & Engineering and Information Technology

Jaypee University of Information Technology, Waknaghat, Solan - 173234 (India)

CERTIFICATE

This is to certify that the work which is being presented in the project report titled "**DIABETIC RETINOPATHY DETECTION**" in partial fulfillment of the requirements for the award of the degree of B.Tech in Computer Science And Engineering and submitted to the Department of Computer Science And Engineering, Jaypee University of Information Technology, Waknaghat is an authentic record of work carried out by **Ambikesh Jha(201453),Animesh (201567)** during the period from July 2023 to May 2024 under the supervision of **Dr. Ekta Gandotra,** , Department of Computer Science and Engineering, Jaypee University of Information Technology, Waknaghat.

Name: Ambikesh Jha
Enrollment No.: 201453

Name: Animesh
Enrollment No.: 201567

The above statement made is correct to the best of my knowledge.

Dr. Ekta Gandotra
Assistant Professor (Grade II)
Computer Science & Engineering and Information Technology
Jaypee University of Information Technology, Waknaghat

Candidate's Declaration

I hereby declare that the work presented in this report entitled **'Diabetic Retinopathy Detection'** in partial fulfillment of the requirements for the award of the degree of **Bachelor of Technology** in **Computer Science & Engineering / Information Technology** submitted in the Department of Computer Science & Engineering and Information Technology**,** Jaypee University of Information Technology, Waknaghat is an authentic record of my own work carried out over a period from August 2023 to May 2024 under the supervision of **Dr. Ekta Gandotra** (Assistant Professor, Department of Computer Science & Engineering and Information Technology).

The matter embodied in the report has not been submitted for the award of any other degree or diploma.

(Student Signature with Date)
Student Name: Ambikesh Jha
Roll No.: 201453

(Student Signature with Date)
Student Name: Animesh
Roll No.: 201567

This is to certify that the above statement made by the candidate is true to the best of my knowledge.

(Supervisor Signature with Date)
Supervisor Name: Dr. Ekta Gandotra
Designation: Assistant Professor
Department: Computer Science and Information Technology
Dated:

ACKNOWLEDGEMENT

I sincerely thank the Almighty God for his divine favor, which has enabled us to successfully finish the project work.

Supervisor Dr. Ekta Gandotra, Associate Professor, Department of Computer Science and Engineering, Jaypee University of Information Technology, Waknaghat, has my sincere gratitude and debt.My supervisor has extensive knowledge and a strong interest in machine learning, which has allowed me to complete this project thanks to his unending patience, academic guidance, ongoing encouragement, energetic supervision, constructive criticism, helpful advice, and corrections at every turn. I want to sincerely thank him for his helpful assistance in getting my job finished.

I would also like to extend a warm welcome to my friend Suveer Sharma and all the people who have either directly or indirectly assisted me in achieving success with this initiative. In this particular circumstance, I may wish to express my gratitude to the numerous staff members—both instructional and non-instructional—who have produced their helpful assistance and enabled my project.

Without acknowledging my loving parents' prayers, well wishes, counsel, and spiritual support—all of which greatly aided me in achieving my objective—no statement of gratitude would be adequate.

Name: Ambikesh Jha
Enrollment No.: 201453

Name : Animesh
Enrollment No.: 201567

TABLE OF CONTENT

LIST OF TABLES

LIST OF FIGURES

LIST OF ABBREVIATIONS

Abbreviation	**Definition**
DR	Diabetic Retinopathy
CNN	Convolutional Neural Networks
NPDR	Non Proliferative Diabetic Retinopathy
PDR	Proliferative Diabetic Retinopathy
WHO	World Health Organization
ResNet	Residual Network
SVM	Support Vector Machine
VGG	Visual Network Group
RNN	Recurrent Neural Network
T-SNE	T-distributed Stochastic Neighbour

ABSTRACT

Diabetic Retinopathy(DR) is a disorder that can negatively impact not just the eyes but also other organs due to high blood sugar levels in diabetic patients.The longer someone has had diabetes, the higher the chance they will develop diabetic retinopathy.Consequently, the risk of Blindness could result from not treating this illness in a timely manner.So the early treatment of Diabetic Retinopathy is must and it should be treated as early as possible.

Our effort seeks to identify the stage of diabetic retinopathy by using fundus pictures and an EfficientNet-B5 model with a transfer learning technique. The main objective is to prevent blindness by early detection of diabetic retinopathy. To evaluate the disease's consequence, we employ deep transfer learning and classification methods to classify a patient's retinal pictures into five labels from 0 to 4. All classifications normal, mild DR and moderate DR designate different complications. For a given input fundus image, the output label identifies one stage from each of these five categories. We test the severity of a patient's diabetic retinopathy. The measure of evaluation depends on the type or way in which technology is applied, and it may be qualitative or quantitative. Typical of such output is stage-by-stage grading (mild, moderate or severe) of diabetic retinopathy. Whether produced by automated systems using artificial intelligence and machine learning algorithms, however, the kind of output is different.

Automated systems can highlight areas of concern in retinal images with visualizations, heatmaps, or other representations. It is critical to understand that the output is a tool for medical professionals, and any conclusions or decisions about treatment that arise from it should be discussed with licensed medical professionals. Even with the advancement of technology, diabetics still require routine eye exams and screenings in order to effectively detect and treat diabetic retinopathy.

Four distinct machine learning models is used in this study on a dataset: ResNet50 performed with an accuracy of 90.4% and a kappa score of 0.839; VGG19 attained an accuracy of 78.5% and a kappa score of 0.625; DenseNet150 performed with an accuracy of 90.1% and a kappa score of 0.834; and EfficientNetB5 outperformed the other models with an accuracy of 93% and a kappa score of 0.864. With EfficientNetB5 appearing as the most promising alternative, these results demonstrate the different effectiveness of each model in handling the given task.

CHAPTER 1: INTRODUCTION

1.INTRODUCTION

Diabetic Retinopathy is a disorder that can negatively impact not just the eyes but also other organs due to high blood sugar levels in diabetic patients[1]. Diabetes causes small blood vessels all across the body ,including those in the retina, to gradually deteriorate with the passage of time. Diabetes-related retinopathy is the result of this process, which causes blood and other fluids to flow out of these small vessels.

This seepage causes swelling in the retinal tissue, which results in a visual distortion or clouding. Both eyes are usually vulnerable to the effects of this disease. The longer someone has had diabetes, the higher the chance they will develop diabetic retinopathy. Blindness could result from not treating this illness in a timely manner.

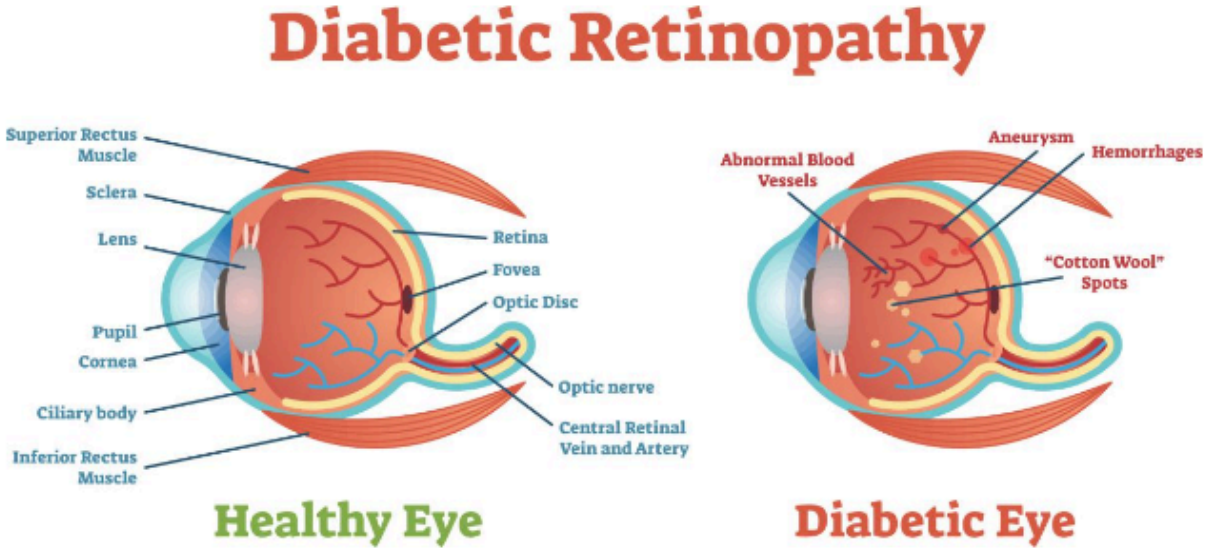


Figure 1.1 Image of diabetic retinopathy[2]

Figure 1.1 shows the ailment in the eye during diabetic retinopathy and its comparison with the normal eye.

The following are warning signs of diabetic retinopathy:

i. Monitoring floaters or spots

ii. Seeing things that are distorted

iii. Observing a blank or black area in the center of your field of vision of vision iv. having trouble seeing at night .

Two types of diabetic retinopathy can appear:

A. Diabetic Retinopathy Without Proliferation (NPDR)

The early stage of the illness, known as NPDR, frequently shows little or no symptoms at all. The retina's blood vessels are damaged during this stage. Macula enlargement may result from the formation of microaneurysms, which are microscopic bulges in the blood vessel.

B. Diabetic Proliferative Retinopathy (PDR)

PDR is the most severe type of the illness, with circulation problems resulting in retinal oxygen deprivation. As a result, fresh, delicate blood vessels could

To reduce the risk of blindness via early detection of diabetic retinopathy. We came up with a machine learning model and classified patients' retinal pictures into five labels, ranging from 0 to 4, in order to evaluate the disease's consequences using deep transfer learning and classification techniques. Every classification as normal ,mild DR , Moderate DR, severe DR , or proliferative DR designates a particular complication associated with the condition. One stage from each of these five categories is identified by the output label for a particular input fundus image.

Transfer Learning : The ability to apply knowledge from completed activities to more recent and related tasks is known as transfer learning. A deep convolutional neural network is trained on a set of images, with each layer being trained by running the images through different filters. The image activations at each layer are multiplied by the values of the filter matrices. Finding the best values for these filter matrices is the goal of deep network training; this allows the output activations to accurately determine the class that an image belongs to as it moves through the network. Leveraging the insights gained allows us to generalize features and weights for following iterations. Even when the data is drastically reduced, when a task contains a large amount of data. Certain fundamental features ,such as edge, form between jobs in computer vision problems, which makes it easier to transfer knowledge between them.

Moreover, as the graphic above illustrates, knowledge from a previous task provides an extra input during the learning of a new target task.

Residual Network (ResNet): We have used the ResNet model to face problems like vanishing gradients. Residual Blocks, the Residual Network (ResNet) introduced a novel solution to the issue of vanishing or exploding gradients. This architecture uses a technique called skip connections, in which some layers are bypassed and the activations of one layer are directly connected to those of the layers that come after it, creating a residual block. After that, residual blocks are stacked to create ResNets. The idea is to not require individual layers to learn the fundamental mapping, but to allow the network to adapt to the residual mapping.ResNet can be scaled up by adding layers and going from ResNet-18 to ResNet-200. Similarly, GPipe scaled up a baseline CNN by a factor of four to achieve an 84.3% ImageNet top-1 accuracy. Increasing the CNN's width or depth arbitrarily[24], or using higher input image resolution for testing and training, are the standard approaches for model scaling.

VGG19 : It is a well-known deep convolutional neural network architecture with a total of 19 layers—16 convolutional layers and 3 fully connected layers. Unlike ResNet, which learns complex features in a hierarchical fashion through skip connections, VGG19 learns complex features through its deep structure. Each convolutional layer has 3x3 filters with a stride of 1, and the max-pooling layers use 2x2 filters with a stride of 2, which helps to create a uniform and easily understood architecture. Even in the lack of skip connections, VGG19 is able to understand progressively abstract features deeper into the network thanks to the stacked convolutional layers. VGG19 can be scaled up by adding more layers or a larger network, which could enhance its performance on a variety of computer vision tasks. Moreover, techniques like data augmentation, transfer learning, and dropout regularization strengthen VGG19 and increase both its efficacy and generalizability. All things considered, VGG19 is a versatile and dependable choice for a variety of visual recognition tasks due to its ease of use and ability to learn hierarchical representations.

DenseNet169 : DenseNet169 is a deep convolutional neural network architecture that introduces dense connectivity patterns among layers, setting it apart from more conventional architectures such as VGG and ResNet. As a member of the DenseNet family, DenseNet169 is

distinguished by its dense block structure, in which every layer in a block receives direct input from every layer that came before it. Because of this dense connectivity, vanishing gradient problems are mitigated and improved gradient flow is encouraged throughout the network. Features are reused and propagated. DenseNet concatenates feature maps across layers to enable the learning of richer feature representations, in contrast to ResNet, which uses skip connections to add the output of earlier layers to later ones. Specifically, DenseNet169 consists of 169 layers: transition layers, globally average pooling layer, softmax classifier, and densely connected blocks.The architecture is appropriate for tasks requiring a limited amount of computational power due to its compactness and parameter efficiency. Furthermore, DenseNet169's ability to learn complex feature hierarchies can be improved by expanding the network or adding more layers. To increase its robustness and generalization abilities, methods like batch normalization and dropout regularization are frequently used. In conclusion, DenseNet169 offers advantages in terms of efficiency and performance for a variety of computer vision tasks due to its dense connectivity, scalability, and parameter efficiency.

EfficientNetB5 : Architecture of convolutional neural networks EfficientNetB5 is renowned for its exceptional computational efficiency and performance. Compound scaling, which balances the model's depth, width, and resolution to achieve optimal performance under computational constraints, sets apart the EfficientNet model family. EfficientNetB5 is particularly notable for having more depth and width than its predecessors, which enhances representational capacity and feature extraction capabilities. EfficientNetB5 achieves an astounding 89% accuracy rate at a relatively low computational cost. Compound scaling is a method that ensures that the architectural parameters of the model are effectively optimized, leading to improved performance without unnecessarily increasing computational demands. EfficientNetB5 can be implemented on devices with limited computational resources due to its scalability, which allows it to adapt to various resource limitations. Moreover, its overall performance and generalization capabilities can be enhanced by applying techniques like knowledge distillation and transfer learning. To sum up, EfficientNetB5 is a good choice for a range of computer vision applications, such as image classification and object detection. This is because of its compound scaling and distinctive architectural design, which balance efficiency and performance.

1.2 PROBLEM DESCRIPTION

Our effort seeks to identify the stage of diabetic retinopathy by using fundus pictures and an EfficientNet-B5 model with a transfer learning technique. The main goal is to reduce the risk of blindness via early detection of diabetic retinopathy. We classify a patient's retinal pictures into five labels, ranging from 0 to 4, in order to evaluate the disease's consequences using deep transfer learning and classification techniques. Every classification: normal ,mild DR , Moderate DR,severe DR, or proliferative DR this designates a particular complication associated with the condition. One stage from each of these five categories is identified by the output label for a particular input fundus image.

1.3 OBJECTIVES

1. To develop an automated model using machine learning to detect the class of diabetic retinopathy using fundus images.
2. To compare and find the kappa score and accuracy of a various models like Resnet50 ,VGG19 ,DenseNet169 and EfficientNetB5.

1.4 SIGNIFICANCE AND MOTIVATION OF PROJECT WORK

The WOrld health Organization (WHO)[3] estimated that 412 million people worldwide received a diabetes mellitus diagnosis in 2014. In 2010, one-third of diabetic patients experienced visual impairment due to diabetic retinopathy, which was found in 33% of cases. Forecasts suggest that the number of cases of diabetic retinopathy could quadruple by 2050, with the Americas being the most affected.

Drug resistance identification requires specialized knowledge, an area in which deep learning techniques can be useful. But diagnosing DR requires a large dataset, a resource that is deficient in healthcare and takes a long time to cure ,a problem that transfer learning may be able to address.

1.5 ORGANIZATION OF PROJECT REPORT

CHAPTER 2: LITERATURE SURVEY

The chapter “Literature Survey“ consists of tools and techniques along with the approach used. All the literature reviews are compiled in the form of a table to make it readable and easy to understand. This chapter also includes the studies that are relevant for the development of a Diabetic Retinopathy detection system.

CHAPTER 3: SYSTEM DEVELOPMENT

The chapter “System Development” consists of the architecture and an explanation of the parts and how they work in accordance with each other. The chapter also includes the techniques used for processing images. The chapter also includes methods of data cleaning and improvement.

CHAPTER 4: TESTING

In the chapter Testing, current modeling technology for evaluation of tests systems in both qualitative and quantitative terms is presented. Challenges and limitations are also considered. Strict testing protocols are established, and basis benchmark datasets to analyze fault handling system problems as well as special conditions.

CHAPTER 5: RESULTS AND EVALUATION

The results of the Diabetic Retinopathy Detection System are shown in this chapter. This chapter also includes data analysis and necessary improvements, as well as contributions and achievements made in the system development process.

CHAPTER 6: CONCLUSIONS AND FUTURE SCOPE

This concluding chapter includes a discussion about things that went wrong during the collecting of data, as well as contributions and accomplishments in terms of system development. The chapter discusses needed improvements in the system for better results and reducing false cases.

CHAPTER 2: LITERATURE SURVEY

The higher-order spectra are used to detect the stages of diabetic retinopathy by means of deep learning (DL) and feature extraction-based classification. To extract features, the 300-fundus image set used in a study by Acharya et al. [5] was exploited using higher-order spectra technique. With these features fed to a support vector machine classifier, it is able to categorize the images into five groups with an 82 % sensitivity and an 88 % specificity. Methods for extracting DR lesions, such as blood vessels, exudates and microaneurysms have been devised by several algorithms [19]. Support vector machines were then used to classify the DIARETDB1 dataset into positive and negative classes based on features including

Feature extraction-based classification algorithms are reliant on expertise to identify valuable information. As for these features, identifying them and extracting them takes a lot of time. In addition, experiments by DL systems indicate that CNNs beat feature extraction based methods [6]. Two main strategies have been used in DL for diabetic retinopathy (DR) classification training: From the beginning, learning and using previously acquired knowledge.

A convolutional neural network (CNN) was trained using a dataset of 128,175 fundus photos to classify images into two groups: images with severity levels 0 and 1 belong to the first class, while those having levels of 2,3 or 4 square into the second class. [7]. The study [7], which took 9963 images from the EyePACS-1 dataset and applied a high sensitivity operating cut point, achieved a sensitivity of 97.5 % and specificity of 93.4 %. On the Messidor-2 dataset, it also achieved a sensitivity of 96.1 % and a specificity of 93.9 %. The sensitivity and specificity scores generated with the evaluation cut point derived from its high degree of selectivity were 87% and 98.5 % respectively on EyePACS-1, but they fell to only slightly lower figures at a respective figure of 90.3 % for Messidor-2 (Eye PRES)

Pratt et al. [8] classified DR into five groups by building a Convolutional Neural Network (CNN) using the stochastic gradient descent method with more than 70,00 samples of fundus images as their training dataset. The most noteworthy statistics that CNN achieved were a 30 % sensitivity, an accuracy of 75 %, and a specificity of X. In other research, a DL model was

trained to automatically identify DR using the MESSIDOR-2 dataset. The results from this effort were 96.8 % sensitivity and 87 % specificity.

A Convolutional Neural Network (CNN) trained from scratch classified Kaggle dataset fundus images into referable and non-referable classes with a sensitivity of 96.2 % and specificity of 66.4%, [9]. A different CNN diabetic retinopathy (DR) classifier was trained on a dataset of 71,896 fundus images with sensitivity = 90.5 % and specificity = 91.6 %. [10] Furthermore, a DL model for which sensitivity and specificity scores are both 94 % was trained on a set of fundus images numbering 75.137 [11].

As for Kaggle dataset, Mohammadian et al. [12] upgraded the pre-trained models, Inception-V3 and Exception respectively through classifying them into two categories; time and resources both did better than Deep Learning (DL), in terms of efficiency. After the data augmentation used to make the dataset balanced, exception attained an accuracy score of 74.49%, while Inception-V3 achieved a score of 87.12 %.

Wan et al. [13] adjusted hyperparameters and transferred learning between pre-trained models (AlexNet, VggNet-s, VggNet-16, VggNet-19, GoogleNet, and ResNet) using the Kaggle dataset. The results were then compared. During training, the VggNet model achieved the highest accuracy score of 95.68%, highlighting the effectiveness of hyperparameter tuning [13]. This is noteworthy. In a related study [14], the use of transfer learning addressed issues related to a small training dataset for retinal vascular segmentation. Moreover, an Inception-V4 [15] model-based classification of diabetic retinopathy demonstrated increased sensitivity in comparison to evaluations by human expert graders on 25,326 retinal images from diabetic patients in Thailand [16].

Mansour [17] used transfer learning to train a deep convolutional neural network for feature extraction in the development of a computer-aided diagnostic system for deep learning (DR). A subset of 2,000 fundus photos from the Kaggle dataset was chosen for a separate study by Dutta et al. [18]. Along with the VggNet16 model, they trained a deep and shallow feed-forward neural network. On a 300-image test dataset, the shallow neural network achieved 41% accuracy, the deep neural network achieved 86.3% accuracy, and the VggNet-16 achieved

78.3% accuracy [14].

A 4,476-byte training dataset was collected and classified into four groups based on anomalies and required therapy [19]. The input images were rescaled to 600 by 600 and then split into four pieces of resolution appearing as noisy. The subdivisions were then fed into separate pre-trained Inception-V3 models known as the Inception@4s. After it was found that Inception@4 overtook the VggNet and ResNet models in terms of accuracy, this model components were inserted into a web-based diabetic retinopathy (DR) categorization system.

To assign the images in Kaggle dataset into five grades of Diabetic Retinopathy (DR), a multi-cell, convolutional neural network was constructed with multiple tasks [20]. This network uses mean square error and cross entropy. Multi-class VggNet classifier based on binary trees was trained on the Kaggle dataset by Adly et al. [22]. Validated on a set of 6,000 fundus images, the classifier had an accuracy and sensitivity rate of 83.2 % each.

With Deep Learning-based, Mateen et al. [21] got 98.34 % of the Kaggle data set for classification using a dataset of over thirty five thousand labeled fundus images widely used in diabetic retinopathy research as foundation materials; this is another example here. With fully connected layers based on the VGG-19 model using Support Vector Machines (SVMs), they attained this level of precision.

S. No.	**Paper Title [Cite]**	**Journal / Conference (Year)**	**Results**	**Limitations**
1.	Application of higher order spectra for the identification of diabetes retinopathy stages[5]	Journal of Medical Systems(2018)	classified into positive and negative classes	Dataset is highly imbalanced
2.	Deep Learning based Computer-Aided	arXiv preprint arXiv(2018)	expertise to identify	very low specificity

	Diagnosis Systems for Diabetic Retinopathy[6]		important features.	
3.	Development and validation of a deep learning algorithm for detection of diabetic retinopathy in retinal fundus photographs[7]	Jama(2018)	achieved a sensitivity of 96.1% and a specificity of 93.9%	pre-processing should be done accurately
4.	Convolutional neural networks for diabetic retinopathy.[8]	Procedia Computer Science(2019)	attained 30% sensitivity, 75% accuracy, and 95% specificity	time costly
5.	Deep learning approach for diabetic retinopathy screening.[9]	Acta Ophthalmologica (2019)	trained data having 75137 images	time costly
6.	Development and validation of a deep learning system for diabetic retinopathy and related eye diseases using retinal images from multiethnic populations with diabetes.[10]	Jama (2019)	Inception-V3 achieved an accuracy score of 87.12%.	requires high resolution images
7.	Automated identification of diabetic retinopathy using deep learning.[11]	Ophthalmology, (2018)	VggNet model achieved the highest accuracy score of 95.68%,	have to design every model and assemble it

8.	Comparative Study of Fine-Tuning of Pre-Trained Convolutional Neural Networks for Diabetic Retinopathy Screening[12]	National and 2nd International Iranian Conference on Biomedical Engineering IEEE, (2019)	deep neural network achieved 86.3% accuracy	_
9.	Deep convolutional neural networks for diabetic retinopathy detection by image classification.[13]	Computers & Electrical Engineering, (2018)	Inception@4 outperformed the VggNet and ResNet models in terms of accuracy	difficult to train and test dataset
10.	Multi-level deep supervised networks for retinal vessel segmentation.[14]	International journal of computer assisted radiology and surgery,(2022)	achieved an accuracy of 83.2% and sensitivity of 81.8%.	input data provided should be accurate
11.	Inception-v4, inception-resnet and the impact of residual connections on learning.[15]	Thirty-First AAAI Conference on Artificial Intelligence, 2022	accuracy of 98.34% on the Kaggle dataset	dataset should be high resolution

2.2 KEY GAPS IN LITERATURE:

The key gap in our literature surveys is that the dataset in every model was either incomplete or the dataset was imbalanced.

Lower accuracy and overfitting compared to more complex models. The impact of lossy data augmentation requires further investigation, especially concerning dataset variability. Limitations such as potential patient overlap between training and test sets.Clinicians lack access to complete clinical data.

Due to hardware constraints, images were down-sampled to 224×224 pixels before being fed into pre-trained networks, potentially sacrificing some accuracy, especially for subtle findings.

CHAPTER 3: SYSTEM DEVELOPMENT

3.1 REQUIREMENTS AND ANALYSIS

HARDWARE REQUIREMENTS

1. **FUNCTIONAL REQUIREMENTS**

   1.1 Operating System:

   Linux (RedHat, Ubuntu, CentOS 6+, and other variants), Windows 7 or later, or 64-bit macOS 10.13+.

   1.2 System architecture:

   64-bit x86 for Linux, 64-bit x86 for Power8/Power9, 64-bit x86 for Windows, and 64-bit x86 for MacOS. The required environment is Python 2.7 or later.

2. **NON-FUNCTIONAL REQUIREMENTS**

   2.1 System Reliability:

   Design the system to handle an increasing volume of retinal images as the dataset grows. Ensure scalability for potential future enhancements.

SOFTWARE REQUIREMENTS

1. **FUNCTIONAL REQUIREMENTS**

   1.1 Data Input:

   The system should accept digital retinal images in standard formats (JPEG, PNG).

Images should be preprocessed for normalization and augmentation.

1.2 Model Training:

Utilize transfer learning to leverage a pre-trained models .Fine-tune the model's parameters using a dataset of retinal images with annotated diabetic retinopathy labels.

1.3 Prediction:

Apply the trained models to a prediction mechanism for unseen retinal images.Output probability scores or confidence levels of predicted diabetic retinopathy severity

1.4 User Interface:

Design a user-friendly interface to upload images and view results. Provide some ways for users to understand or interpret the system's predictions.

1.5 Performance Evaluation:

In addition to using evaluation metrics such as accuracy, precision, recall and F1 score for the assessment of model performance; produce detailed reports on various aspects of system diagnostic capabilities.

1.6 Integration with existing system:

Compatibility with Electronic Health Record (EHR) systems, and easy integration into hospital or clinic databases.

2. NON-FUNCTIONAL REQUIREMENTS

2.1 Performance:

The system must be able to work with images efficiently, giving results in a reasonable time frame. Optimize the model so that it is both accurate and fast.

2.2 Scalability:

It will also be necessary to design the system so that it can handle an ever-increasing number of retinal images. The dataset must therefore have scope for further development.

2.3 Security:

Put in place safe data handling practices, maintaining patient confidentiality and complying with healthcare regulations on the protection of personal data.

2.4 Reliability:

To confirm the stability and dependability of the model, thoroughly test it.Establish backup and recovery procedures for the system in case of malfunctions.

2.5 Usability:

Design an intuitive user interface suitable for healthcare professionals with varying technical expertise. Provide clear and interpretable results to facilitate informed decision-making.

2.6 Interpretability:

Ensure transparency in the model's decision-making process, allowing users to understand how predictions are generated.Provide visualizations or explanations for key features influencing the model's decision.

3.2 PROJECT DESIGN AND ARCHITECTURE

SYSTEM ARCHITECTURE

Finding the proper values for each filter in a CNN basically entails training the system so that an input image passes through numerous layers, activating particular neurons in the final layer that subsequently predicts the correct class. Though most applications require training very large CNNs, which need vast amounts of processing power and processed data, modest projects can successfully train a CNN from the beginning.

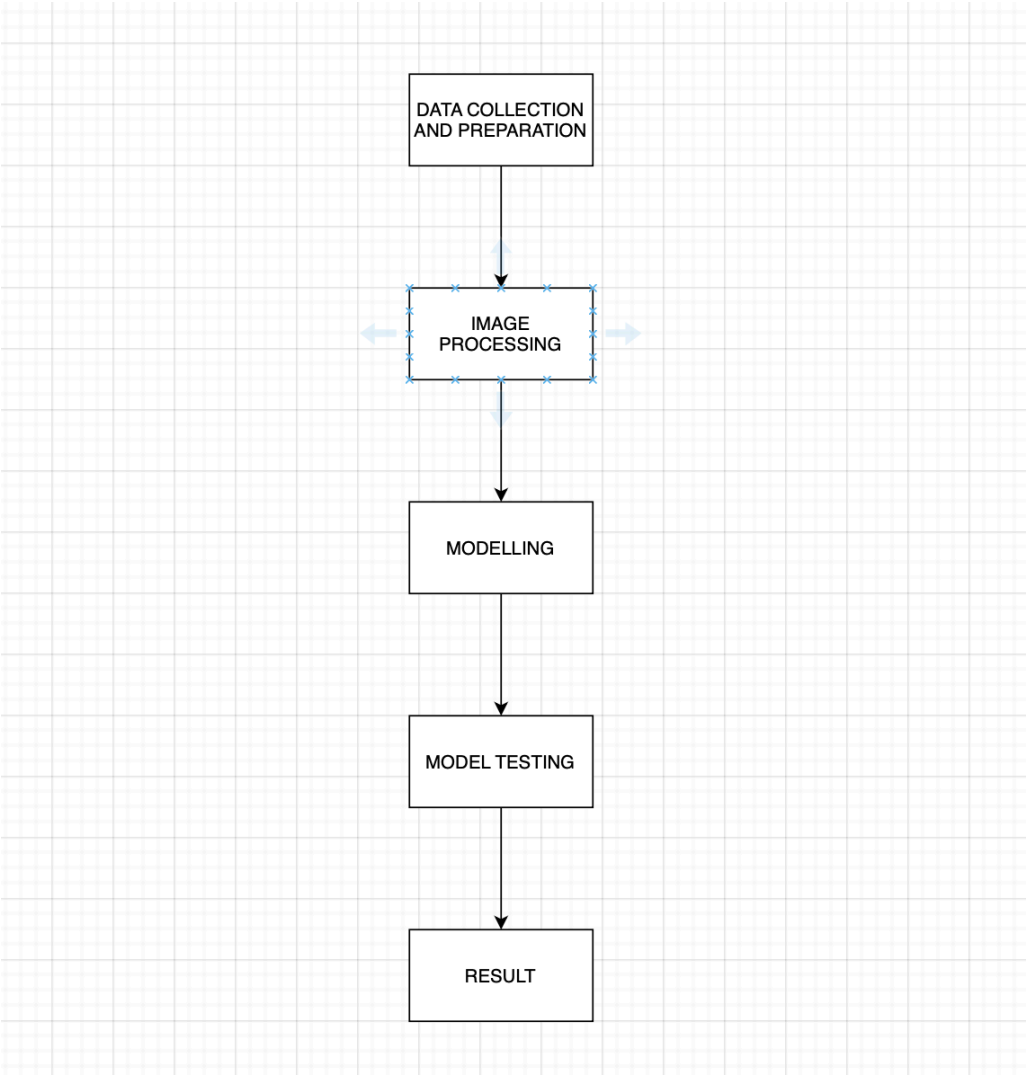


Figure 3.1 System architecture

Figure 3.1 shows the system architecture of the Diabetic Retinopathy Detection and the workflow of the code.

A system's overall structure and organization are referred to as its "system architecture" in most cases. We are probably thinking about the general layout of a system used to detect or treat diabetic retinopathy when we discuss it in relation to the condition. The following crucial elements are probably essential to such a system:

Data acquisition and preparation: During the data acquisition phase, correct analysis is contingent on obtaining high-quality retinal images and methods that have been successful include fundus photography. This is especially so with diabetic retinopathy. It is necessary to have patient data and relevant medical history at one time. The point of this step is to allow easy correlation between retinal images and medical history, through information about the patient's health condition. When these two elements are integrated, the result is a complete set of data that can be used for more specific treatment plans and an even safer diagnosis of those with diabetic retinopathy.

Image Processing : The normalization of retinal images During the preprocessing stage, processing aimed at improving and standardizing both quality and format is vital so as to ensure data input that remains consistent with high-quality. And so modifications to reduce noise and adjust contrast are also key steps that ensure a clear figure, which in turn provides the basis for effective analyses later on. On the other hand, data cleaning (whose purpose is to remove artifacts and undesirable information from patient records) becomes particularly important. Unwanted noise and superfluous information are finely filtered out; this in turn increases the accuracy and reliability of later analysis. These preprocess steps complement each other and form the foundation for more accurate, detailed image analysis related to diabetic retinopathy.

Modeling : A systematic process is adopted when Convolutional Neural Networks (CNNs) and transfer learning are used in modeling diabetic retinopathy. The first step in this procedure was collecting a labeled dataset of retinal images. The dataset is divided into training, validation and testing sets. Preprocessing techniques such as resizing and contrast adjustment are used to increase image quality. After that, in order to speed up training and capture fine details about diabetic retinopathy, transfer learning is used by pre-training CNN models such as VGG16[25] or ResNet with knowledge from general image datasets like Image Net.

After optimizing the CNN model, it can be used directly in clinical environments as an automatic diagnostic tool or incorporated into a decision support system. By working together

with medical experts, we ensure that the model meets clinical needs and is updated frequently in order to cover new information or advances applicable to diabetic retinopathy diagnosis. The all-inclusive way is to successfully employ CNNs and transfer learning in order to increase the accuracy of diabetic retinopathy diagnosis.

Transfer learning: For example, on a set of images the deep convolutional neural network is trained. Each layer was learned with different filters run over the images. At each layer the image activations are multiplied by their respective filter matrices. The goal of training a deep network is thus to determine values for these filter matrices; once we have them, output activations can then tell us whether an image that went through this net belonged in which class. When convolutional neural networks are trained with the ImageNet dataset and the features learned by each filter on each layer examined, an interesting phenomenon appears. A convolution network's initial filters are indeed experts at identifying particular colors and patterns of vertical or horizontal lines. In a series of layers using previously learned colors and lines, step-by-step one comes to recognize basic forms. As the network learns to recognize textures, layers later on in its design are devoted to picking out such parts as noses or legs. In the last stages, filters act upon whole objects. You get what you want out of it in the end. For pretrained networks, transfer learning means stuck at the end of a few dense layers and working out how to combine these features learned in it into recognition objects on new datasets.

When training a deep convolutional neural network, a set of images is used, and the images are passed through various filters for each layer. At each layer, the image activations are multiplied by the values of the filter matrices. The activations from the last layer are used to determine the class of the image.

Finding the best values for each of these filter matrices is our goal when training a deep network so that the output activations can accurately determine the class that an image belongs to as it moves through the network. Gradient descent is the technique used to find these filter matrix values.

After using the ImageNet dataset to train a convolutional neural network (conv net) and looking at the recognition patterns that each filter on each layer has learned, an interesting finding is revealed. The convolution network's first filter layers are trained to recognise particular colours

as well as vertical and horizontal lines. Building on the previously learned knowledge of colors and lines, subsequent layers then gradually evolve to recognise simple shapes. After learning to recognise textures, the next layer is trained to recognise particular features, like the eyes, nose, and legs. Towards the end, whole objects trigger the filters, producing the intended output. To improve object recognition in new datasets.

Residual Network(ResNet): Building convolutional neural networks at a fixed cost and then scaling them up to achieve higher accuracy is a common practice with increased resources. ResNet can be scaled up by adding layers and going from ResNet-18 to ResNet-200[26]. Similarly, GPipe scaled up a baseline CNN by a factor of four to achieve an 90.3% ImageNet top-1 accuracy. Increasing the CNN's width or depth arbitrarily, or using higher input image resolution for testing and training, are the standard approaches for model scaling. Although these techniques do increase accuracy, they frequently result in less-than-ideal performance and need laborious manual adjustment. We discovered a more ethical way to increase CNN's accuracy and efficiency.

VGG19 : Scaling up VGG19 entails expanding the number of layers in the network architecture, much like ResNet. Because of its simple design, VGG19 can be easily expanded, allowing researchers to increase the network's depth and thus achieve higher accuracy.VGG19 is able to achieve an accuracy of 79% and kappa score of 0.623 But just adding more layers can result in higher computational costs and diminishing returns. While methods like expanding the network or training with higher-resolution images can improve performance, they frequently call for labor-intensive manual tweaking and might not always produce the best results.

DenseNet169 : In a similar vein, DenseNet169's scalability enables researchers to boost the network's depth and width while maintaining performance. Each block's dense connectivity encourages feature propagation and reuse, strengthening the network's representational power. However, there may be potential performance trade-offs and higher computational demands associated with scaling DenseNet169. DenseNet 8s able to achieve an accuracy of 90.1% and a kappa score of 0.83 Efficient model operation necessitates careful consideration of architectural modifications and computational resources in order to achieve higher accuracy.

EfficientNetB5 : Notable for its remarkable performance-to-efficiency ratio, EfficientNetB5 offers a unique scalability approach. Using compound scaling, EfficientNetB5 maximizes model width, depth, and resolution to achieve higher accuracy within predefined computational bounds giving an accuracy of 93 % and kappa score of 0.864. This method ensures that the model's complexity increases proportionately and doesn't result in excessive computational overhead. Although it raises ethical questions, this strategy can optimize accuracy and efficiency while consuming the fewest resources, making it a strong option for scalable deep learning models.

We directly incorporate speed information into the search algorithm's primary reward function in order to get around the limitations of mobile speed. The search can now find a model that strikes a fair balance between speed and accuracy thanks to this integration.

Our methodology is based on running the model on a platform of choice, in this case, the Pixel phones used for this study, in order to directly assess model speed. This is different from previous architecture search techniques that use a different proxy to measure model speed. Our method enables a direct measurement of what is achievable in real-world scenarios, acknowledging that different types of mobile devices have unique hardware and software characteristics, potentially requiring a different design to achieve optimal accuracy and performance trade-offs.

Our method consists of three main parts: an RNN-based controller that learns and samples model architectures; a trainer that builds and trains models to attain accuracy; and an inference engine that measures the model speed on actual mobile phones using TensorFlow Lite. We formulate a multi-objective optimisation problem and use a reinforcement learning algorithm with a customized reward function to find Pareto-optimal solutions in order to simultaneously achieve high accuracy and fast speed.

A convolutional neural network is broken down into a series of blocks using a new factorized hierarchical search space that we present. The layer architecture for every block is then determined using a hierarchical search space, which strikes the ideal compromise between search flexibility and space size. This method enables various layers to make use of unique connections and operations. In addition, we impose consistent organization on every layer in

every block, which leads to a much smaller search space than if every layer had a flat search space. Examples from this novel factorized hierarchical search space demonstrate the variety of layers in our MnasNet network's architecture.

Testing: For model testing against kappa, we used the Cohen's Kappa coefficient as a statistical indicator of agreement between expected and observed classifications. This coefficient is especially useful in the case of categorical data, such as diabetic retinopathy. As well as the accuracy of model performance, this method also takes into account chance agreement to produce a value between -1 and 1. Higher values correspond to greater agreement, zero is chance-level agreement and lower numbers imply poor performance. When it comes to measuring how accurately the model models categories such as mild, moderate and severe diabetic retinopathy which are separate categories in themselves--kappa is a focus of attention. Its detailed analysis provides useful information that points to enhancements which can help increase diagnostic accuracy in this difficult medical field.

Results: In light of the widespread problem that many healthcare professionals have inadequate training data, our work makes effective use of transfer learning to classify DR patients into five groups. Because it uses a quadratic weighted kappa loss function and batch gradient descent with gradually increasing learning rate as its training algorithm, our model is better than other approaches. It has higher accuracy as well. This is particularly significant for the healthcare industry, which often lacks data. It is especially effective in medical image classification, achieving first-class results on new unseen data. The reasons our model can designate a higher DR classification accuracy are the training algorithm and loss function chosen by us. This same methodology could be helpful in many other medical image classifications as well, not just because of this particular problem that there is insufficient data to learn from available for use in most healthcare scenarios. To further improve efficacy in deep reinforcement learning classification as well as other related tasks, various pre-trained deep convolutional networks need to be put through their paces thoroughly via experimental assessment.

3.3 DATA PREPARATION

The Kaggle dataset provides us with all fundus images, which are labeled 0, 1, 2, 3, and 4.. there are 4675 fundus mages in all the data set. 3662 of these photos, whose picture IDs and diagnosis labels are stored in Train.csv, were used to train the model. The remaining 995 images, which have image ID and diagnosis labels attached, are kept in Test.csv

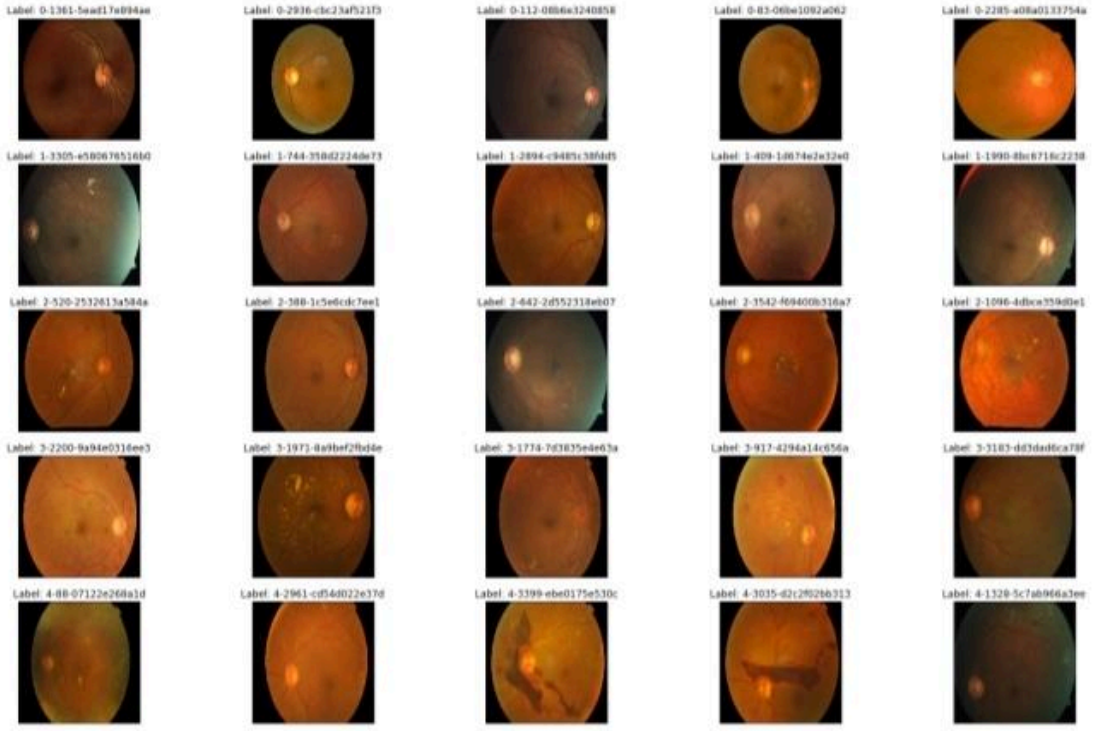


Figure 3.2 Input images

Figure 3.2 shows the original image provided to the project. Gathering, cleaning, and labeling images are steps in the data preparation process for input images. It involves operations like pixel value normalization, standardization, and resizing.

IMAGES WITHOUT RESIZING

Figure 3.3 shows the final images that we obtained from applying the Ben Graham preprocessing method to remove color distraction. Specifically, we applied a tolerance of 7 to each image, which removed the darker, less informative portions and revealed the brighter, more informative portion of the image.

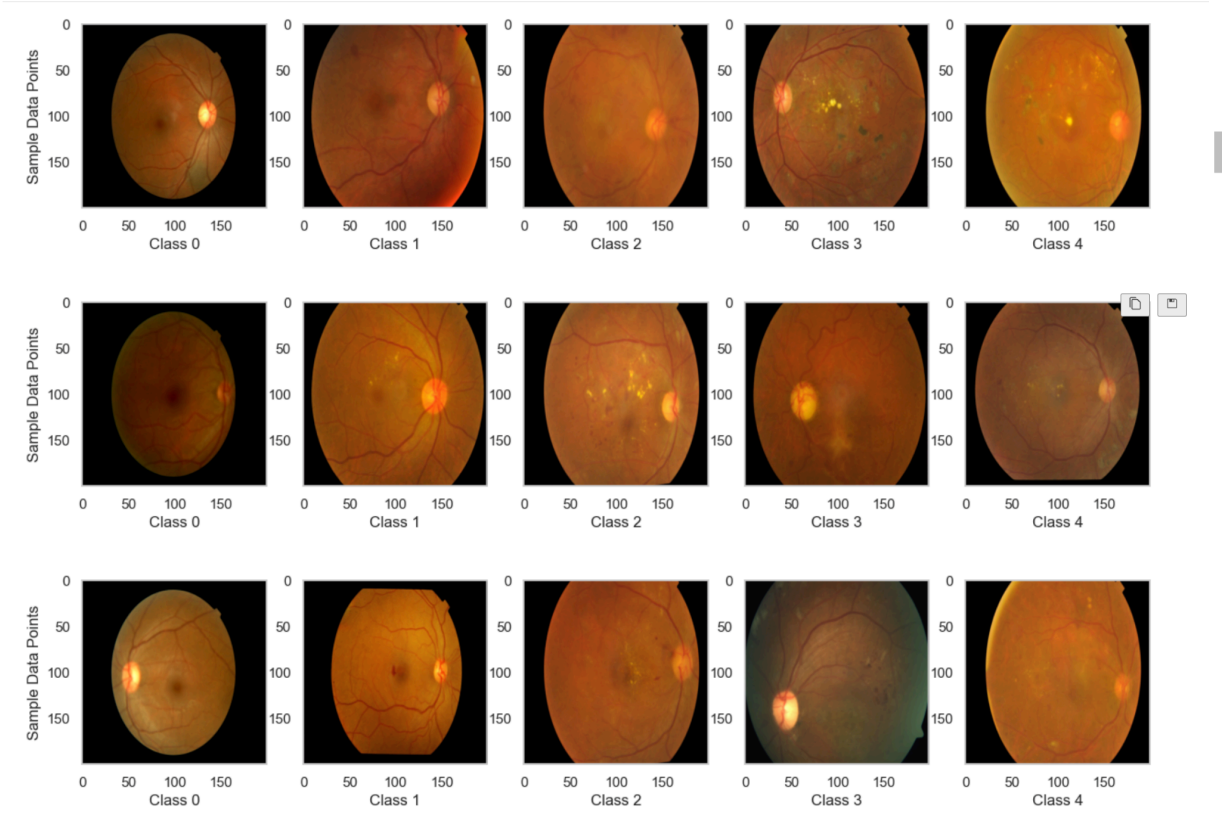


Figure 3.3 images without resizing.

DATA AUGMENTATION

Enhancing training datasets with image flipping, rotation, and brightness adjustments is known as data augmentation, and it is a crucial machine learning technique, especially in computer vision. In addition to improving model robustness, this procedure helps the model generalize to new, untested data. Preventing overfitting and enabling models to handle a variety of real-world scenarios is achieved by increasing the training dataset. For real-world applications, data augmentation is essential because it develops robustness to changes in orientation, size, and lighting. It also solves the issue of the original dataset size limitation by efficiently

expanding the training dataset without gathering additional labeled data. Model adaptability and overall reliability are greatly enhanced by this multipurpose instrument.

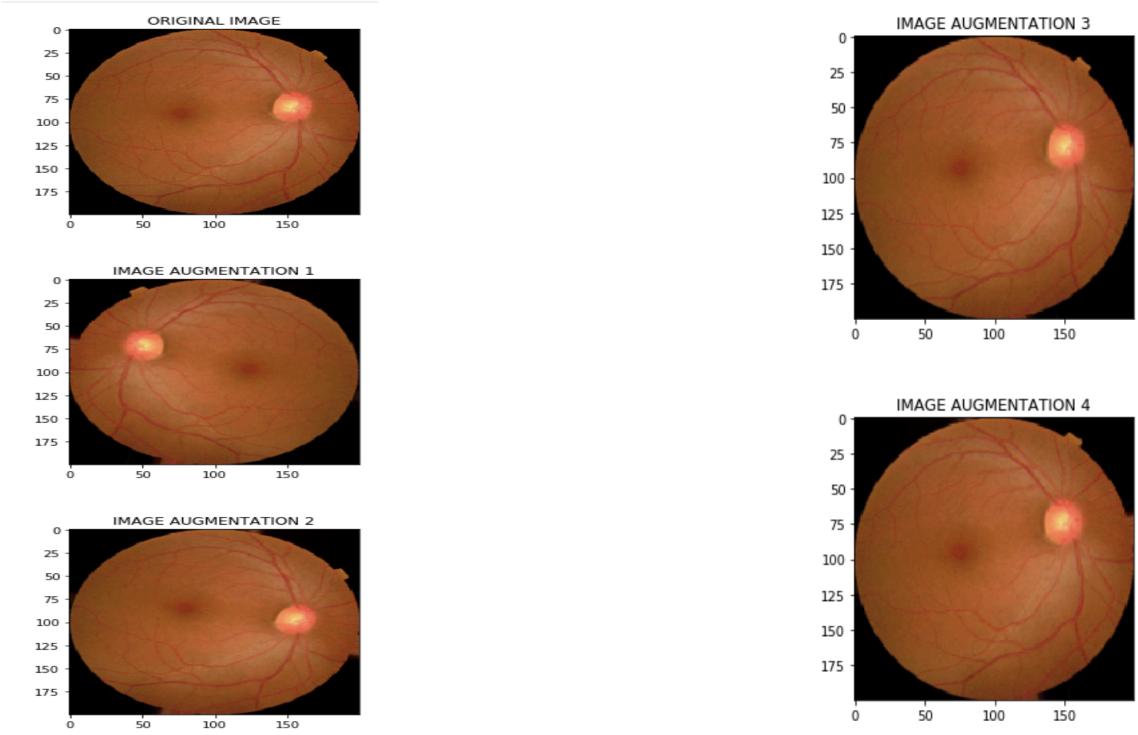
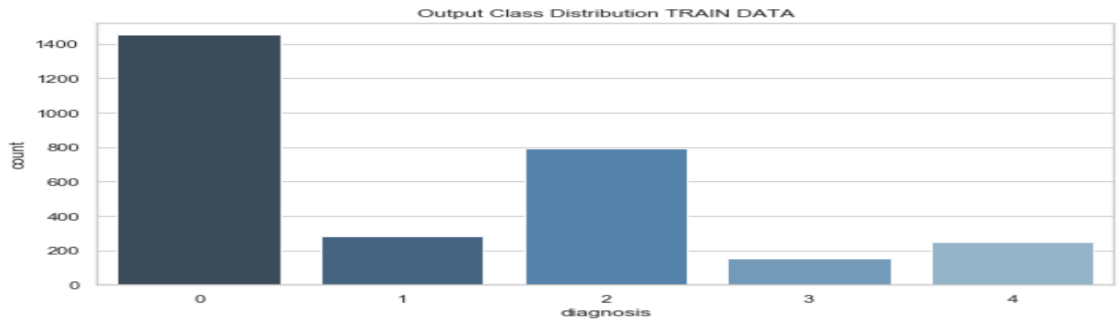


Figure 3.4 Data augmentation

Figure 3.4 shows the data augmentation of image and how the augmented and original images differ from one another and how different transformations, like flipping, rotation, and brightness adjustments, are applied to enhance the dataset. This demonstrates how the method offers a larger set of training examples, which improves the model's resilience and adaptability to novel situations.




```
plot_classes(df_train_valid,'VALIDATION DATA')
```

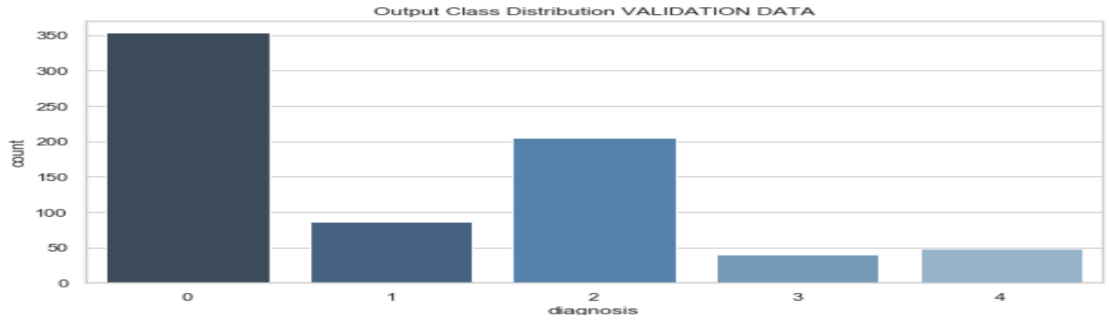


Figure 3.5 Distribution of data in training and validation dataset.

Figure 3.5 shows distribution of data in training and validation dataset .Robust machine learning models rely heavily on the distribution of data in training and validation datasets. The validation dataset evaluates the model's performance on new, untested data, while the training dataset must accurately reflect real-world scenarios for effective training. A distribution that is well-balanced facilitates a dependable comprehension of a model's performance in various classes, thereby augmenting its flexibility and dependability in real-world scenarios.

IMAGE PREPROCESSING

Improving the caliber of the input photos is an obvious way to raise our model's efficiency. Diminishing the impact of lighting: pictures have a variety of lighting settings; some are really dark and hard to see. To improve our visualization, we can attempt to convert the image to grayscale.

IMAGES BEFORE PRE-PROCESSING

Each of the original images shown in the diagrams represents a different stage of diabetic retinopathy disease and was sourced from datasets. Notably, one image appears brighter than the other, drawing attention away from other colors, while the other is darker and contains areas that don't convey anything.In order to resolve the problems seen in the previous mentioned images, we will utilize pre-processing techniques to address these issues.

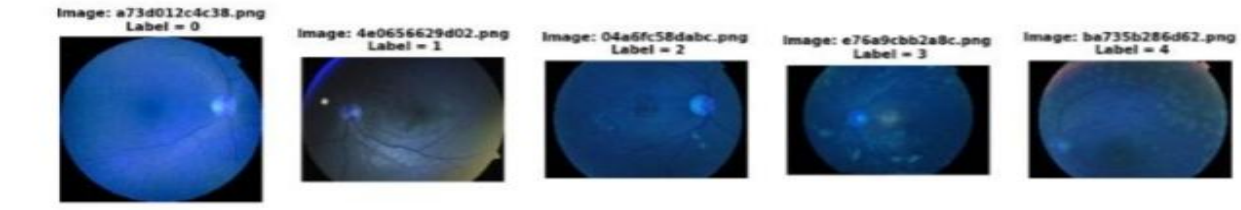


Figure 3.6 Images before pre-processing

Figure 3.7 shows Images before pre-processing. Before undergoing any pre-processing, raw images are in their original, unaltered state, exhibiting differences in size, resolution, and color intensity. There isn't the uniformity in these photos that machine learning models need. Resizing, normalization, and augmentation are examples of pre-processing tasks that are used to produce consistent, clean data so that the model can effectively learn from the input images.

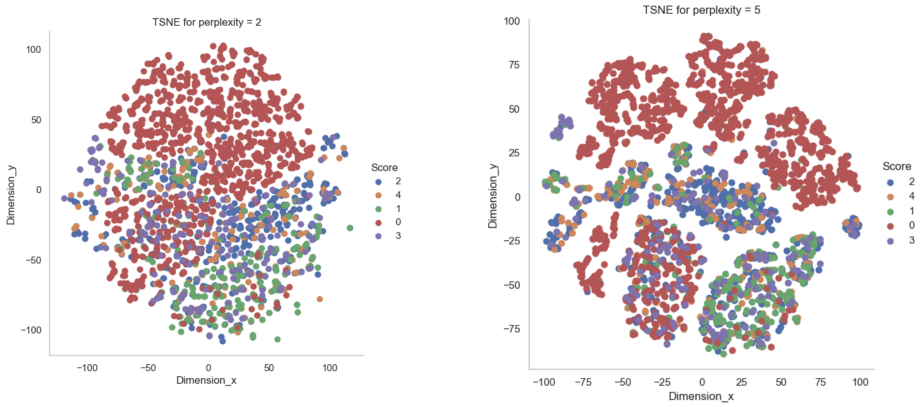
Figure 3.7 After preprocessing of Images .

Figure 3.8 shows After preprocessing the images . After the images are preprocessed, the data is standardized and refined to make sure machine learning is consistent. By utilizing a clean

and well-structured dataset, these improvements allow the model to learn and enhance its recognition of patterns and features. Inconsistencies, noise, and variations in image quality are also addressed during the preprocessing stage. Ultimately, the improved dataset optimizes the performance of image-based machine learning algorithms and lays the groundwork for strong model training, allowing for improved generalization to a variety of scenarios.

T-SNE VISUALIZATION OF GRAY SCALE IMAGES

T-distributed Stochastic Neighbor Embedding (t-SNE),is commonly used to reduce the dimensionality of data. Which is especially helpful when displaying high-dimensional information in a lower-dimensional setting. For grayscale images, t-SNE functions to translate these intricate feature vectors into a two-dimensional space. Initially, each image is characterized by a large number of pixels, creating a high-dimensional feature space.It is noteworthy that this modification maintains the relative similarity between the photos, which results in a meaningful representation. To put insightful information in a more comprehensible and approachable style, t-SNE generally aids in the visualization of intricate linkages and structures in the data.



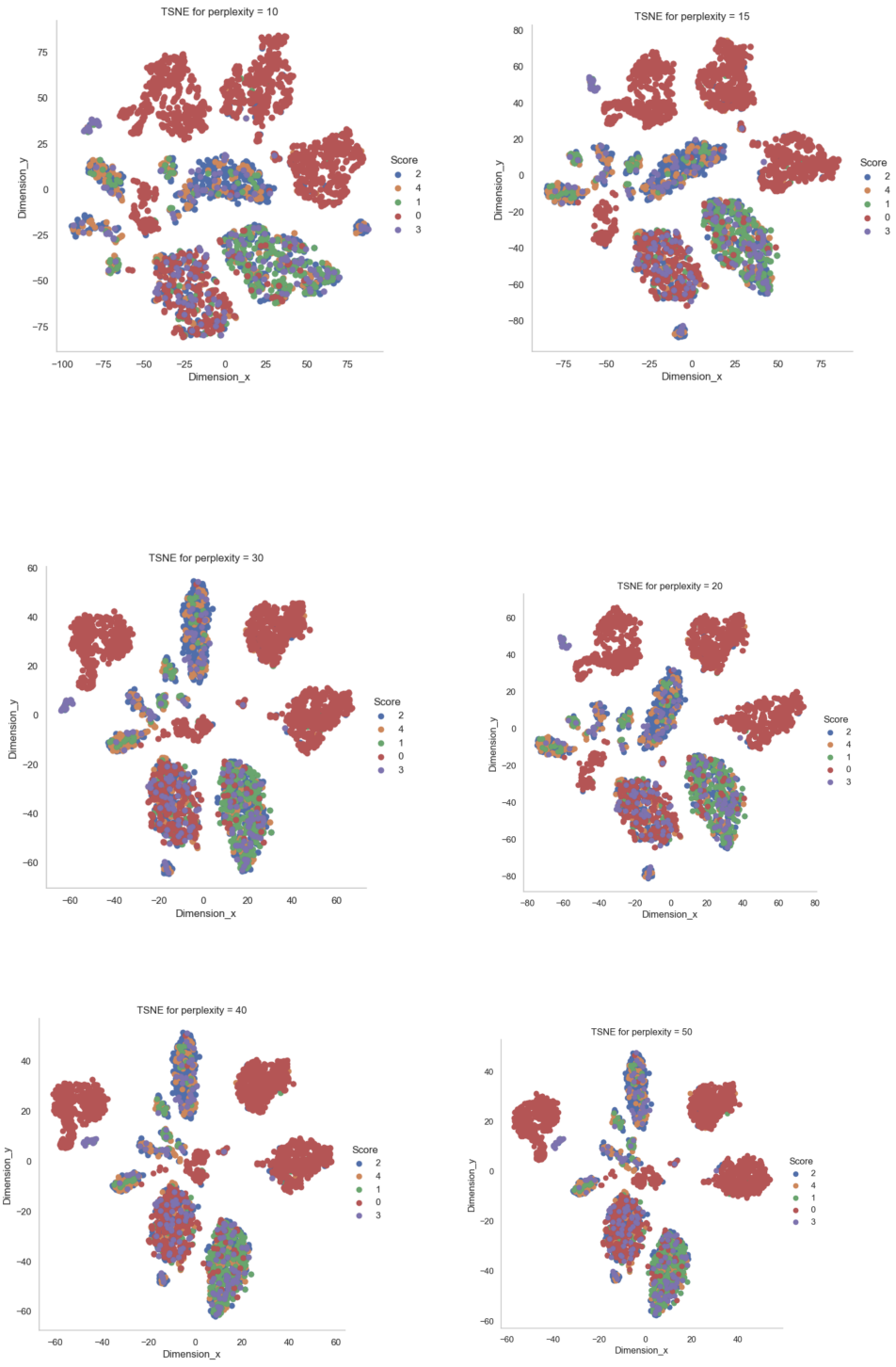


Figure 3.8 T-SNE visualization of gray scale images.

When t-SNE is used, grayscale images with similar pixel intensity patterns are clustered into different groups, as shown in Figure 3.9. It may have been difficult to see intricate linkages,

patterns, and structures in the data in the original high-dimensional space, but the resulting graph makes them evident. An effective method for exploratory data analysis, t-SNE helps uncover underlying patterns in the dataset and offers valuable insights into its complex structures.

3.4 IMPLEMENTATION

SAMPLE CODE

```
#ref - https://stackoverflow.com/questions/14463277/how-to-disable-python-warnings

# Basic Libs..
import warnings
warnings.filterwarnings("ignore")
import pandas as pd
import numpy as np
from tqdm import tqdm,tqdm_notebook
from prettytable import PrettyTable
import pickle
import os
print('CWD is ',os.getcwd())

# Vis Libs..
from sklearn.manifold import TSNE
import seaborn as sns
import matplotlib.pyplot as plt
%matplotlib inline
plt.rcParams["axes.grid"] = False

# Image Libs.
from PIL import Image
import cv2

# DL Libs..
import keras
from keras import applications
# from keras.preprocessing.image import ImageDataGenerator,load_img
from keras.preprocessing.image import ImageDataGenerator
# from tensorflow.keras.utils import img_to_array,array_to_img
from keras import optimizers,Model,Sequential
from keras.layers import Input,GlobalAveragePooling2D,Dropout,Dense,Activation
from keras.callbacks import EarlyStopping,ReduceLROnPlateau
```

✓ 5.4s Python

```
'''
This function reads data from the respective train and test directories
'''

def load_data():
    train = pd.read_csv('train.csv')
    test = pd.read_csv('test.csv')

    train_dir = os.path.join('./','train_images/')
    test_dir = os.path.join('./','test_images/')

    train['file_path'] = train['id_code'].map(lambda x: os.path.join(train_dir,'{}.png'.format(x)))
    test['file_path'] = test['id_code'].map(lambda x: os.path.join(test_dir,'{}.png'.format(x)))

    train['file_name'] = train["id_code"].apply(lambda x: x + ".png")
    test['file_name'] = test["id_code"].apply(lambda x: x + ".png")

    train['diagnosis'] = train['diagnosis'].astype(str)

    return train,test
```

✓ 0.0s Python

```
df_train,df_test = load_data()
print(df_train.shape,df_test.shape,'\n')
df_train.head(6)
```

✓ 0.3s Python

```
(3662, 4) (1928, 3)
```

```
'''This Function Plots a Bar plot of output Classes Distribution'''

def plot_classes(df):
    df_group = pd.DataFrame(df.groupby('diagnosis').agg('size').reset_index())
    df_group.columns = ['diagnosis','count']

    sns.set(rc={'figure.figsize':(10,5)}, style = 'whitegrid')
    sns.barplot(x = 'diagnosis',y='count',data = df_group,palette = "Blues_d")
    plt.title('Output Class Distribution')
    plt.show()
```

Code Snippet 3.1 Loading Libraries, Data Loading Function, Load Data, Plotting Class Distribution, Plot Class Distribution

In this code snippet, the first step is to import libraries for deep learning, data manipulation and visualization. Then a function to load training and testing data from CSV files is defined, as well as new columns for file names and paths. The data is loaded into two different dataframes (named df_train and df_test}. Finally, a bar graph showing the distribution of classes in the training data is plotted using a function plot_classes. This first step prepares the data and suggests some features of class distribution. The rest is more analysis.

```
# Defining a global variable to be used as Image size..
IMG_SIZE = 200
```

✓ 0.0s Python

```
'''This Function converts a color image to gray scale image'''

def conv_gray(img):
    img = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
    img = cv2.resize(img, (IMG_SIZE,IMG_SIZE))
    return img


'''
This Function shows the visual Image photo of 'n x 5' points (5 of each class)
'''

def visualize_imgs(df,pts_per_class,color_scale):
    df = df.groupby('diagnosis',group_keys = False).apply(lambda df: df.sample(pts_per_class))
    df = df.reset_index(drop = True)

    plt.rcParams["axes.grid"] = False
    for pt in range(pts_per_class):
        f, axarr = plt.subplots(1,5,figsize = (15,15))
        axarr[0].set_ylabel("Sample Data Points")

        df_temp = df[df.index.isin([pt + (pts_per_class*0),pt + (pts_per_class*1), pt + (pts_per_class*2),pt + (pts_per_class*3),pt + (pt
        for i in range(5):
            if color_scale == 'gray':
                img = conv_gray(cv2.imread(df_temp.file_path.iloc[i]))
                axarr[i].imshow(img,cmap = color_scale)
            else:
                axarr[i].imshow(Image.open(df_temp.file_path.iloc[i]).resize((IMG_SIZE,IMG_SIZE)))
            axarr[i].set_xlabel('Class '+str(df_temp.diagnosis.iloc[i]))

        plt.show()
```

✓ 0.0s Python

```
visualize_imgs(df_train,2,color_scale = 'gray')
✓ 2.0s                                                    Python
```

Code snippet 3.2 Grayscale Images for every class to present a subset of data for visualization

This code snippet defines two image processing functions. The `conv_gray` function can be used to turn an image in colour into grayscale and resize it to the desired size. The three parameters passed to the function `visualize_imgs` are a colour scale, a dataframe and the number of images to show per class. Then, according to the chosen colour scale level it takes a sub-sample of images from each class in the dataframe and shows five per class in either colour or shades. This allows for visual inspection and comparison of the various dataset classes.

IMAGE PROCESSING - GAUSSIAN BLUR

```
'''
This section of code applies gaussian blur on top of image
'''

rn = np.random.randint(low = 0,high = len(df_train) - 1)

img = cv2.imread(df_train.file_path.iloc[rn])
img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
img = cv2.resize(img, (IMG_SIZE,IMG_SIZE))

img_t = cv2.addWeighted(img,4, cv2.GaussianBlur(img , (0,0) , 30) ,-4 ,128)

f, axarr = plt.subplots(1,2,figsize = (11,11))
axarr[0].imshow(img)
axarr[1].imshow(img_t)
plt.title('After applying Gaussian Blur')
plt.show()
```

✓ 0.5s Python

```
'''
This Function performs image processing on top of images by performing Gaussian Blur and Circle Crop
'''

def crop_image_from_gray(img,tol=7):
    if img.ndim ==2:
        mask = img>tol
        return img[np.ix_(mask.any(1),mask.any(0))]
    elif img.ndim==3:
        gray_img = cv2.cvtColor(img, cv2.COLOR_RGB2GRAY)
        mask = gray_img>tol

        check_shape = img[:,:,0][np.ix_(mask.any(1),mask.any(0))].shape[0]
        if (check_shape == 0): # image is too dark so that we crop out everything,
            return img # return original image
        else:
            img1=img[:,:,0][np.ix_(mask.any(1),mask.any(0))]
            img2=img[:,:,1][np.ix_(mask.any(1),mask.any(0))]
            img3=img[:,:,2][np.ix_(mask.any(1),mask.any(0))]
    #         print(img1.shape,img2.shape,img3.shape)
            img = np.stack([img1,img2,img3],axis=-1)
    #         print(img.shape)
        return img


def circle_crop(img, sigmaX):
    """
    Create circular crop around image centre
    """
    img = crop_image_from_gray(img)
    img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)

    height, width, depth = img.shape

    x = int(width/2)
    y = int(height/2)
    r = np.amin((x,y))

    circle_img = np.zeros((height, width), np.uint8)
    cv2.circle(circle_img, (x,y), int(r), 1, thickness=-1)
    img = cv2.bitwise_and(img, img, mask=circle_img)
    img = crop_image_from_gray(img)
    img=cv2.addWeighted(img,4, cv2.GaussianBlur( img , (0,0) , sigmaX) ,-4 ,128)
    return img
```

✓ 0.0s Python

Code snippet 3.3 Applying gaussian blur and circular crop for better visualization

This code snippet illustrates a number of image processing techniques for enhancing the visual quality of medical images. The original and processed photos are placed on the same line in the first part of the code; then a random selection is made to apply Gaussian Blur to an image from training dataset. In section 2, in addition to Gaussian blurring, the functions crop_image_from gray and circle_crop describe image and circular cropping. In medical image analysis tasks, they heighten focus on important areas at the expense of noise which makes featureestraction easier and higher quality.

```
'''Perform Image Processing on a sample image'''

rn = np.random.randint(low = 0,high = len(df_train) - 1)

#img = img_t
img = cv2.imread(df_train.file_path.iloc[rn])
img_t = circle_crop(img,sigmaX = 30)

f, axarr = plt.subplots(1,2,figsize = (11,11))
axarr[0].imshow(cv2.resize(cv2.cvtColor(img, cv2.COLOR_BGR2RGB),(IMG_SIZE,IMG_SIZE)))
axarr[1].imshow(img_t)
plt.title('After applying Circular Crop and Gaussian Blur')
plt.show()
```

✓ 1.4s Python

```
# ref - https://www.kaggle.com/ratthachat/aptos-eye-preprocessing-in-diabetic-retinopathy
'''
This Function shows the visual Image photo of 'n x 5' points (5 of each class)
and performs image processing (Gaussian Blur, Circular crop) transformation on top of that
'''

def visualize_img_process(df,pts_per_class,sigmaX):
    df = df.groupby('diagnosis',group_keys = False).apply(lambda df: df.sample(pts_per_class))
    df = df.reset_index(drop = True)

    plt.rcParams["axes.grid"] = False
    for pt in range(pts_per_class):
        f, axarr = plt.subplots(1,5,figsize = (15,15))
        axarr[0].set_ylabel("Sample Data Points")

        df_temp = df[df.index.isin([pt + (pts_per_class*0),pt + (pts_per_class*1), pt + (pts_per_class*2),pt + (pts_per_class*3),pt + (pt
        for i in range(5):
            img = cv2.imread(df_temp.file_path.iloc[i])
            img = circle_crop(img,sigmaX)
            axarr[i].imshow(img)
            axarr[i].set_xlabel('Class '+str(df_temp.diagnosis.iloc[i]))

        plt.show()
```

✓ 0.0s Python

```
# normalize
X = X_train / 255

# reshape
X = X.reshape(X.shape[0], -1)
trainy = df_train['diagnosis']
```

Python

```
per_vals = [2,5,10,15,20,30,40,50]

for per in tqdm_notebook(per_vals):
    X_decomposed = TSNE(n_components=2,perplexity = per).fit_transform(X)
    df_tsne = pd.DataFrame(data=X_decomposed, columns=['Dimension_x','Dimension_y'])
    df_tsne['Score'] = trainy.values

    sns.FacetGrid(df_tsne, hue='Score', height=6).map(plt.scatter, 'Dimension_x', 'Dimension_y').add_legend()
    plt.title('TSNE for perplexity = ' + str(per))
    plt.show()
```

Python

Code snippet 3.4 Gaussian blur and circular crop of a data frame from training set

The code uses a function to visualize the effects of image processing on a subset of data points belonging to different classes, applies Gaussian blur and circular crop to an image that is

selected and displayed from a dataframe, reshapes and normalizes training data before applying t-SNE for different levels of perplexity, and displays the results using scatter plots, where the color of each plot corresponds to the target variable.

```python
# ref - https://www.youtube.com/watch?v=hxLU32zhze0
# ref - https://stackoverflow.com/questions/49643907/clipping-input-data-to-the-valid-range-for-imshow-with-rgb-data-0-1-for-floa
# ref - https://keras.io/preprocessing/image/

'''This Function generates 'lim' number of Image Augmentations from a random Image in the directory'''

def generate_augmentations(lim):
    datagen = ImageDataGenerator(featurewise_center=True,
                                 featurewise_std_normalization=True,
                                 rotation_range=20,
                                 #width_shift_range=0.2,
                                 #height_shift_range=0.2,
                                 horizontal_flip=True)
    img = cv2.imread(df_train.file_path.iloc[np.random.randint(low = 0,high = len(df_train) - 1)])
    img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
    img = cv2.resize(img, (IMG_SIZE,IMG_SIZE))
    plt.imshow(img)
    plt.title('ORIGINAL IMAGE')
    plt.show()

    img_arr = img.reshape((1,) + img.shape)

    i = 0
    for img_iterator in datagen.flow(x = img_arr,batch_size = 1):
        i = i + 1
        if i > lim:
            break
        plt.imshow((img_iterator.reshape(img_arr[0].shape)).astype(np.uint8))
        plt.title('IMAGE AUGMENTATION ' + str(i))
        plt.show()
```

Python

```python
generate_augmentations(4)
```

Python

Code snippet 3.5 Function to generate and visualize image augmentations.

For image augmentation, the code makes use of the Keras ImageDataGenerator. With the given augmentation parameters, it initializes the data generator. displays the original image after reading a random image from the training dataset. creates augmented images by reshaping the

image for the data generator. shows the initial enhanced picture.The generate_augmentations function will run 143 times, displaying the augmented images each time before stopping at the 143rd iteration..

```python
#ref - https://stackoverflow.com/questions/14463277/how-to-disable-python-warnings

# Basic Libs..
import multiprocessing
from multiprocessing.pool import ThreadPool
import warnings
warnings.filterwarnings("ignore")
import pandas as pd
import numpy as np
from tqdm import tqdm,tqdm_notebook
from prettytable import PrettyTable
import pickle
import os
print('CWD is ',os.getcwd())

# Vis Libs..
from sklearn.manifold import TSNE
import seaborn as sns
import matplotlib.pyplot as plt
%matplotlib inline
plt.rcParams["axes.grid"] = False

# Image Libs.
from PIL import Image
import cv2

# sklearn libs..
from sklearn.model_selection import train_test_split

# DL Libs..
import keras
from keras import applications
from keras.preprocessing.image import ImageDataGenerator,img_to_array,array_to_img,load_img
from keras import optimizers,Model,Sequential
from keras.layers import Input,GlobalAveragePooling2D,Dropout,Dense,Activation
from keras.callbacks import EarlyStopping,ReduceLROnPlateau
```

Python

```python
'''
This function reads data from the respective train and test directories
'''

def load_data():
    train = pd.read_csv('train.csv')
    test = pd.read_csv('test.csv')

    train_dir = os.path.join('./','train_images/')
    test_dir = os.path.join('./','test_images/')

    train['file_path'] = train['id_code'].map(lambda x: os.path.join(train_dir,'{}.png'.format(x)))
    test['file_path'] = test['id_code'].map(lambda x: os.path.join(test_dir,'{}.png'.format(x)))

    train['file_name'] = train["id_code"].apply(lambda x: x + ".png")
    test['file_name'] = test["id_code"].apply(lambda x: x + ".png")

    train['diagnosis'] = train['diagnosis'].astype(str)

    return train,test
```

Python

Code snippet 3.6 Loading and visualizing the data and preparing for building a deep learning model

1. Loading Libraries: PNDs, numpy, tqdm, prettytable, OS, scikit-learn, seaborn, matplotlib, PIL (Pillow), cv2, and Keras are among the libraries that must be imported before the code can

begin.

2. Define the function `load_data`: The function reads data from CSV files for testing and training and specifies the routes to the image directories for training and testing. It then adds a new column with each image's complete path in each data frame and adds a new column with the picture filename to each dataframe. It then gives back the testing and training dataframes.

```python
df_train,df_test = load_data()
print(df_train.shape,df_test.shape,'\n')
df_train.head(6)
```

```python
df_train_train,df_train_valid = train_test_split(df_train,test_size = 0.2)
print(df_train_train.shape,df_train_valid.shape)
```

```
(2929, 4) (733, 4)
```

```python
file = open('df_train_train', 'rb')
df_train_train = pickle.load(file)
file.close()

file = open('df_train_test', 'rb')
df_train_test = pickle.load(file)
file.close()
```

```python
print(df_train_train.shape,df_train_test.shape)
print(len(os.listdir('./train_images_resized_preprocessed')),len(os.listdir('./test_images_resized_preprocessed')))
```

```
(2929, 4) (733, 4)
2929 733
```

Code snippet 3.7 Preparing and visualizing data for training deep learning models

This snippet splits the training data, shows the class distribution, and loads pre-processed data to get the data ready for training a deep learning model. The data is loaded and training data is split in training and validation data in 80% and 20% respectively. After that the pre-saved pickled training data frames are loaded.

```python
IMG_SIZE  = 512
```

Python

```python
'''Function loads an image from Folder , Resizes and saves in another directory '''

def image_resize_save(file):
    input_filepath = os.path.join('./','train_images','{}.png'.format(file))
    output_filepath = os.path.join('./','valid_images_resized','{}.png'.format(file))
    img = cv2.imread(input_filepath)
    cv2.imwrite(output_filepath, cv2.resize(img, (IMG_SIZE,IMG_SIZE)))
#image_resize_save(df_train.id_code.iloc[201])
```

Python

```python
'''This Function uses Multi processing for faster saving of images into folder'''

def multiprocess_image_downloader(process:int, imgs:list):
    """
    Inputs:
        process: (int) number of process to run
        imgs:(list) list of images
    """
    print(f'MESSAGE: Running {process} process')
    results = ThreadPool(process).map(image_resize_save, imgs)
    return results
```

Python

```python
# Use 6 cores
multiprocess_image_downloader(6, list(df_train_valid.id_code.values))
```

Python

```python
# Use 6 cores
multiprocess_image_downloader(6, list(df_train_valid.id_code.values))
```

Python

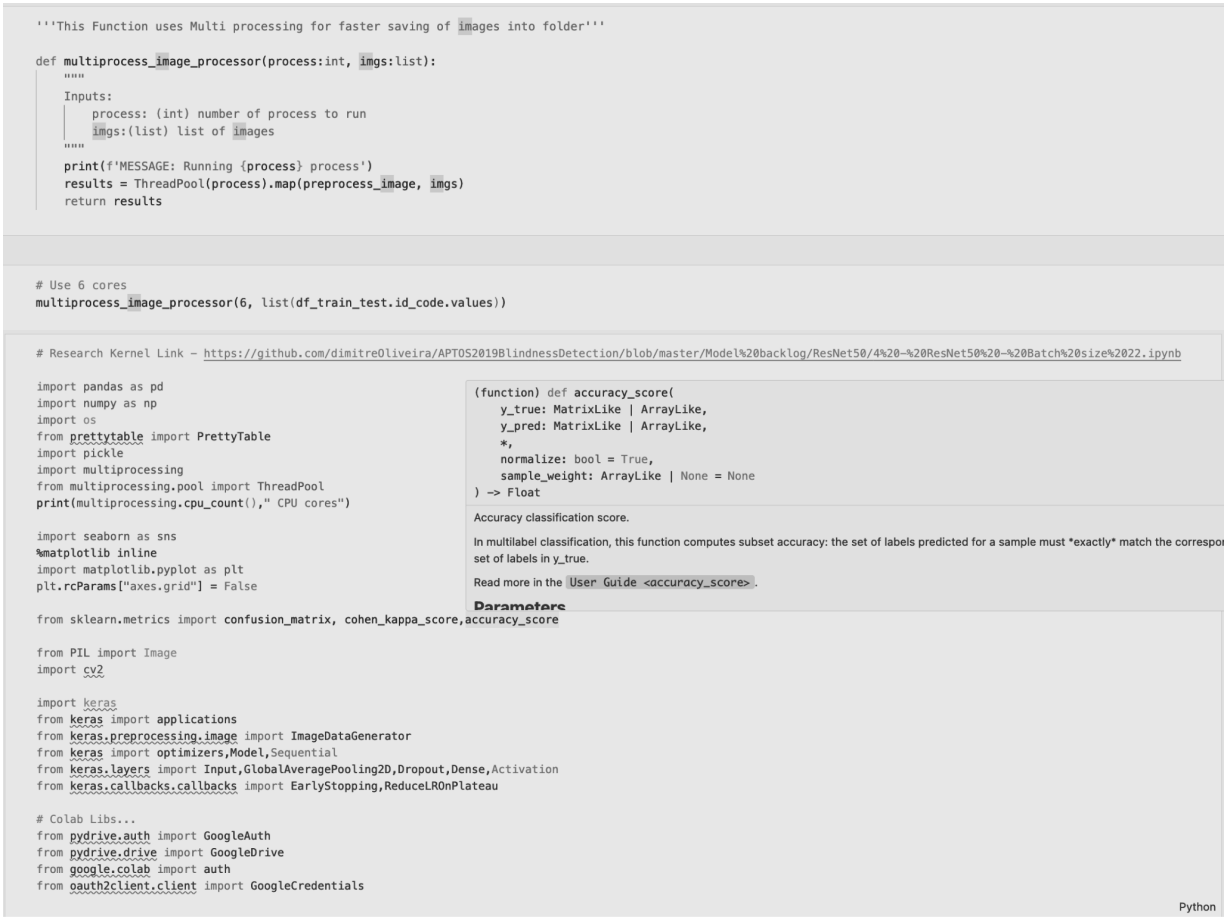
Code snippet 3.8 Resizing and saving image for validation using multiprocessing

The code snippet specifies image size for resizing images and describes how to read, resize and save a picture using ‘image_resize_save’ function. It then describes the ‘multiprocess_image_downloader function which, using multithreading, resizes and saves numerous images. It uses the list of validation image IDs to launch `multiprocess_image_downloader.

```
    def crop_image_from_gray(img,tol=7):
        if img.ndim ==2:
            mask = img>tol
            return img[np.ix_(mask.any(1),mask.any(0))]
        elif img.ndim==3:
            gray_img = cv2.cvtColor(img, cv2.COLOR_RGB2GRAY)
            mask = gray_img>tol

            check_shape = img[:,:,0][np.ix_(mask.any(1),mask.any(0))].shape[0]
            if (check_shape == 0): # image is too dark so that we crop out everything,
                return img # return original image
            else:
                img1=img[:,:,0][np.ix_(mask.any(1),mask.any(0))]
                img2=img[:,:,1][np.ix_(mask.any(1),mask.any(0))]
                img3=img[:,:,2][np.ix_(mask.any(1),mask.any(0))]
        #          print(img1.shape,img2.shape,img3.shape)
                img = np.stack([img1,img2,img3],axis=-1)
        #         print(img.shape)
            return img
def circle_crop(img, sigmaX = 30):
    """
    Create circular crop around image centre
    """
    img = crop_image_from_gray(img)
    img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)

    height, width, depth = img.shape

    x = int(width/2)
    y = int(height/2)
    r = np.amin((x,y))

    circle_img = np.zeros((height, width), np.uint8)
    cv2.circle(circle_img, (x,y), int(r), 1, thickness=-1)
    img = cv2.bitwise_and(img, img, mask=circle_img)
    img = crop_image_from_gray(img)
    img=cv2.addWeighted(img,4, cv2.GaussianBlur( img , (0,0) , sigmaX) ,-4 ,128)
    return img


def preprocess_image(file):
    input_filepath = os.path.join('./','test_images_resized','{}.png'.format(file))
    output_filepath = os.path.join('./','test_images_resized_preprocessed','{}.png'.format(file))

    img = cv2.imread(input_filepath)
    img = circle_crop(img)
    cv2.imwrite(output_filepath, cv2.resize(img, (IMG_SIZE,IMG_SIZE)))
```

Code snippet 3.9 Cropping, Blurring, and resizing images of the test data

The functions defined in this code snippet allow you to pre-process photos before putting them into a deep learning model. `crop_image_from_gray` is defined first in order to eliminate extraneous background using a threshold. Next, {circle_crop` applies Gaussian blur for smoother borders and circles cropping around the center of the image. Lastly, `preprocess_image} receives an image, does the necessary resizing and blurring operations, crops and resizes it, and saves the pre-processed file.



```
'''This Function uses Multi processing for faster saving of images into folder'''

def multiprocess_image_processor(process:int, imgs:list):
    """
    Inputs:
        process: (int) number of process to run
        imgs:(list) list of images
    """
    print(f'MESSAGE: Running {process} process')
    results = ThreadPool(process).map(preprocess_image, imgs)
    return results
```

```
# Use 6 cores
multiprocess_image_processor(6, list(df_train_test.id_code.values))
```

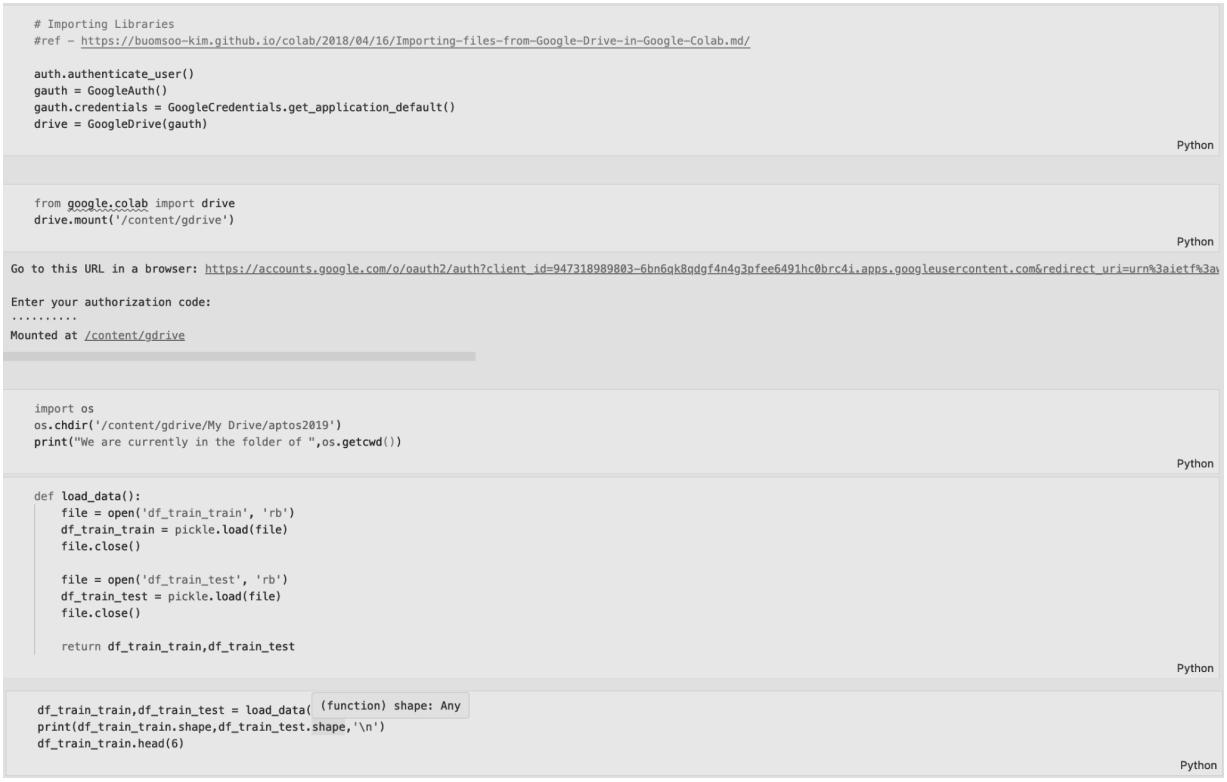
```
# Research Kernel Link - https://github.com/dimitreOliveira/APTOS2019BlindnessDetection/blob/master/Model%20backlog/ResNet50/4%20-%20ResNet50%20-%20Batch%20size%2022.ipynb

import pandas as pd
import numpy as np
import os
from prettytable import PrettyTable
import pickle
import multiprocessing
from multiprocessing.pool import ThreadPool
print(multiprocessing.cpu_count()," CPU cores")

import seaborn as sns
%matplotlib inline
import matplotlib.pyplot as plt
plt.rcParams["axes.grid"] = False

from sklearn.metrics import confusion_matrix, cohen_kappa_score,accuracy_score

from PIL import Image
import cv2

import keras
from keras import applications
from keras.preprocessing.image import ImageDataGenerator
from keras import optimizers,Model,Sequential
from keras.layers import Input,GlobalAveragePooling2D,Dropout,Dense,Activation
from keras.callbacks.callbacks import EarlyStopping,ReduceLROnPlateau

# Colab Libs...
from pydrive.auth import GoogleAuth
from pydrive.drive import GoogleDrive
from google.colab import auth
from oauth2client.client import GoogleCredentials
```

```
(function) def accuracy_score(
    y_true: MatrixLike | ArrayLike,
    y_pred: MatrixLike | ArrayLike,
    *,
    normalize: bool = True,
    sample_weight: ArrayLike | None = None
) -> Float

Accuracy classification score.

In multilabel classification, this function computes subset accuracy: the set of labels predicted for a sample must *exactly* match the correspo
set of labels in y_true.

Read more in the User Guide <accuracy_score>.

Parameters
```

Python

Code snippet 3.10 Preprocessing Image of test set, saving using multiprocessors and importing libraries

The purpose of this snippet of code is to use multiprocessing to preprocess test images in order to make use of available CPU cores for faster execution. The function multiprocess_image_processor is defined, which applies the pre-processing functions on each image in the list by utilizing multiple threads. The code outputs a message verifying the number of threads used for processing and by default uses six (adjustable) threads. It also imports the necessary libraries (like OpenCV for image processing, Seaborn for data visualization, and Keras for deep learning) for additional analysis and model building.



Code snippet 3.11 Mounting google drive and switching directory for further analysis

This bit of code sets up the Google Colab environment for additional analysis. It mounts Google Drive to a specific directory inside Colab, authenticates with Google Drive, and opens the project folder inside the mounted drive. It then loads pre-saved datasets for training and validation, displaying their shapes along with a training data preview for quick exploration. This prepares the loaded data for further analysis and model building.

```python
def crop_image_from_gray(img,tol=7):
    if img.ndim ==2:
        mask = img>tol
        return img[np.ix_(mask.any(1),mask.any(0))]
    elif img.ndim==3:
        gray_img = cv2.cvtColor(img, cv2.COLOR_RGB2GRAY)
        mask = gray_img>tol

        check_shape = img[:,:,0][np.ix_(mask.any(1),mask.any(0))].shape[0]
        if (check_shape == 0): # image is too dark so that we crop out everything,
            return img # return original image
        else:
            img1=img[:,:,0][np.ix_(mask.any(1),mask.any(0))]
            img2=img[:,:,1][np.ix_(mask.any(1),mask.any(0))]
            img3=img[:,:,2][np.ix_(mask.any(1),mask.any(0))]
    #         print(img1.shape,img2.shape,img3.shape)
            img = np.stack([img1,img2,img3],axis=-1)
    #         print(img.shape)
        return img

def circle_crop(img, sigmaX = 30):
    """
    Create circular crop around image centre
    """
    img = crop_image_from_gray(img)
    img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)

    height, width, depth = img.shape

    x = int(width/2)
    y = int(height/2)
    r = np.amin((x,y))

    circle_img = np.zeros((height, width), np.uint8)
    cv2.circle(circle_img, (x,y), int(r), 1, thickness=-1)
    img = cv2.bitwise_and(img, img, mask=circle_img)
    img = crop_image_from_gray(img)
    img=cv2.addWeighted(img,4, cv2.GaussianBlur( img , (0,0) , sigmaX) ,-4 ,128)
    return img

def preprocess_image(file):
    input_filepath = os.path.join('./','train_images_resized','{}.png'.format(file))
    output_filepath = os.path.join('./','train_images_resized_preprocessed','{}.png'.format(file))

    img = cv2.imread(input_filepath)
    img = circle_crop(img)
    cv2.imwrite(output_filepath, cv2.resize(img, (IMG_SIZE,IMG_SIZE)))
```

Python

```python
'''This Function uses Multi processing for faster saving of images into folder'''

def multiprocess_image_processor(process:int, imgs:list):
    """
    Inputs:
        process: (int) number of process to run
        imgs:(list) list of images
    """
    print(f'MESSAGE: Running {process} process')
    results = ThreadPool(process).map(preprocess_image, imgs)
    return results
```

Python

```python
# Model parameters
BATCH_SIZE = 8
EPOCHS = 40
WARMUP_EPOCHS = 2
LEARNING_RATE = 1e-4
WARMUP_LEARNING_RATE = 1e-3
HEIGHT = 320
WIDTH = 320
CANAL = 3
N_CLASSES = df_train_train['diagnosis'].nunique()
ES_PATIENCE = 5
RLROP_PATIENCE = 3
DECAY_DROP = 0.5
```

Python

Code snippet 3.12 Cropping, Blurring, and resizing images of the training data

The purpose of this code snippet is to prepare training images for use in the deep learning model. It outlines three crucial roles: The function “crop_image_from_gray” effectively focuses on the relevant content by removing unnecessary background from the image based on a specified threshold. The function circle_crop carries out two primary operations: first, it crops

the image in a circular pattern around the center, and then it applies a Gaussian blur to minimize noise and smooth the edges. The function `preprocess_image` is responsible for connecting everything. It reads an image, pre-processes it using `crop_image_from_gray} and `circle_crop}, resizes it to the model's desired size, and then stores the prepared image for training.

```
def img_generator(train,test):
    train_datagen=ImageDataGenerator(rescale=1./255, validation_split=0.2,horizontal_flip=True)

    train_generator=train_datagen.flow_from_dataframe(dataframe=df_train_train,
                                                      directory="./train_images_resized_preprocessed/",
                                                      x_col="file_name",
                                                      y_col="diagnosis",
                                                      batch_size=BATCH_SIZE,
                                                      class_mode="categorical",
                                                      target_size=(HEIGHT, WIDTH),
                                                      subset='training')

    valid_generator=train_datagen.flow_from_dataframe(dataframe=df_train_train,
                                                      directory="./train_images_resized_preprocessed/",
                                                      x_col="file_name",
                                                      y_col="diagnosis",
                                                      batch_size=BATCH_SIZE,
                                                      class_mode="categorical",
                                                      target_size=(HEIGHT, WIDTH),
                                                      subset='validation')

    test_datagen = ImageDataGenerator(rescale=1./255)
    test_generator = test_datagen.flow_from_dataframe(dataframe=df_train_te
                                                      directory = "./test_i
                                                      x_col="file_name",
                                                      target_size=(HEIGHT, WIDTH),
                                                      batch_size=1,
                                                      shuffle=False,
                                                      class_mode=None)

    return train_generator,valid_generator,test_generator
```

(constant) WIDTH: Literal[320]

Python

```
train_generator,valid_generator,test_generator = img_generator(df_train_train,df_train_test)
```

Python

```
Found 2344 validated image filenames belonging to 5 classes.
Found 585 validated image filenames belonging to 5 classes.
Found 733 validated image filenames.
```

```
def create_model(input_shape, n_out):
    input_tensor = Input(shape=input_shape)
    base_model = applications.ResNet50(weights=None, include_top=False,input_tensor=input_tensor)
    base_model.load_weights('resnet50_weights_tf_dim_ordering_tf_kernels_notop.h5')

    x = GlobalAveragePooling2D()(base_model.output)
    x = Dropout(0.5)(x)
    x = Dense(2048, activation='relu')(x)
    x = Dropout(0.5)(x)
    final_output = Dense(n_out, activation='softmax', name='final_output')(x)
    model = Model(input_tensor, final_output)
    return model
```

Python

```
model = create_model(input_shape=(HEIGHT, WIDTH, CANAL), n_out=N_CLASSES)

for layer in model.layers:
    layer.trainable = False

for i in range(-5, 0):
    model.layers[i].trainable = True
model.summary()
```

Python

Code snippet 3.13 Configuring tensorFlow imageDataGenerators, classification model.

With TensorFlow's ImageDataGenerator, this code preprocesses images and augments data. It uses a disinct generator for training, validation and testing datasets; it transforms as required (such as horizontal flipping/scaling) to produce batches of labeled images. These generators

are ready to feed into a learning model for machines. The code also describes a function that builds an image classification model based on the ResNet50 architecture. Global average pooling, dropout layers and pre-trained weights are used in this model for effective learning. For multi-class classification, the model is set up with a softmax activation. During training most of its layers are frozen so that only the last five adjust to each new dataset.

```
model.compile(optimizer = optimizers.Adam(lr=WARMUP_LEARNING_RATE),loss = 'categorical_crossentropy',metrics = ['accuracy'])

history_warmup = model.fit_generator(generator=train_generator,
                                     steps_per_epoch=STEP_SIZE_TRAIN,
                                     validation_data=valid_generator,validation_steps=STEP_SIZE_VALID,
                                     epochs=WARMUP_EPOCHS,
                                     verbose=1).history
```

Python

```
Epoch 1/2
293/293 [==============================] - 1422s 5s/step - loss: 1.4534 - accuracy: 0.5943 - val_loss: 1.6658 - val_accuracy: 0.4897
Epoch 2/2
293/293 [==============================] - 68s 233ms/step - loss: 0.8575 - accuracy: 0.6788 - val_loss: 1.7975 - val_accuracy: 0.4905
```

```
for layer in model.layers:
    layer.trainable = True

es = EarlyStopping(monitor='val_loss', mode='min', patience=ES_PATIENCE, restore_best_weights=True, verbose=1)
rlrop = ReduceLROnPlateau(monitor='val_loss', mode='min', patience=RLROP_PATIENCE, factor=DECAY_DROP, min_lr=1e-6, verbose=1)

callback_list = [es, rlrop]
optimizer = optimizers.Adam(lr=LEARNING_RATE)
model.compile(optimizer=optimizer, loss="binary_crossentropy",  metrics=['accuracy'])
model.summary()
```

Python

```
for layer in model.layers:
    layer.trainable = True

es = EarlyStopping(monitor='val_loss', mode='min', patience=ES_PATIENCE, restore_best_weights=True, verbose=1)
rlrop = ReduceLROnPlateau(monitor='val_loss', mode='min', patience=RLROP_PATIENCE, factor=DECAY_DROP, min_lr=1e-6, verbose=1)

callback_list = [es, rlrop]
optimizer = optimizers.Adam(lr=LEARNING_RATE)
model.compile(optimizer=optimizer, loss="binary_crossentropy",  metrics=['accuracy'])
model.summary()
```

Python

```
history_finetunning = model.fit_generator(generator=train_generator,
                                          steps_per_epoch=STEP_SIZE_TRAIN,
                                          validation_data=valid_generator,
                                          validation_steps=STEP_SIZE_VALID,
                                          epochs=EPOCHS,
                                          callbacks=callback_list,
                                          verbose=1).history
```

Python

Code snippet 3.14 Fine tuning the pre-trained modelith warm-up phase

This code refines an image classification model that has already been trained. It freezes the pre-trained layers first to keep the learned features. It then thaws a few last layers to tweak them for that particular dataset. Training in short epochs permits the model to begin adapting to new data without disturbing too much information obtained beforehand. This warmup period allows the model greater room to prepare for subsequent fine-tuning.

```python
# ref - https://stackoverflow.com/questions/29188757/matplotlib-specify-format-of-floats-for-tick-lables
plt.figure(figsize=(8,5))

plt.plot(history_finetunning['accuracy'])
plt.plot(history_finetunning['val_accuracy'])
plt.title('Model Accuracy')
plt.ylabel('Accuracy')
plt.xlabel('Epoch')
plt.legend(['Train', 'Validation'], loc='upper left')
plt.gca().ticklabel_format(axis='both', style='plain', useOffset=False)
plt.show()
```

Python


```python
# ref - https://stackoverflow.com/questions/29188757/matplotlib-specify-format-of-floats-for-tick-lables
plt.figure(figsize=(8,5))

plt.plot(history_finetunning['accuracy'])
plt.plot(history_finetunning['val_accuracy'])
plt.title('Model Accuracy')
plt.ylabel('Accuracy')
plt.xlabel('Epoch')
plt.legend(['Train', 'Validation'], loc='upper left')
plt.gca().ticklabel_format(axis='both', style='plain', useOffset=False)
plt.show()
```

Python


```python
complete_datagen = ImageDataGenerator(rescale=1./255)
complete_generator = complete_datagen.flow_from_dataframe(dataframe=df_train_train,
                                                          directory = "./train_images_resized_preprocessed/",
                                                          x_col="file_name",
                                                          target_size=(HEIGHT, WIDTH),
                                                          batch_size=1,
                                                          shuffle=False,
                                                          class_mode=None)

STEP_SIZE_COMPLETE = complete_generator.n//complete_generator.batch_size
train_preds = model.predict_generator(complete_generator, steps=STEP_SIZE_COMPLETE,verbose = 1)
train_preds = [np.argmax(pred) for pred in train_preds]
```

Python

```python
test_generator.reset()
STEP_SIZE_TEST = test_generator.n//test_generator.batch_size
test_preds = model.predict_generator(test_generator, steps=STEP_SIZE_TEST,verbose = 1)
test_labels = [np.argmax(pred) for pred in test_preds]
```

Python

```
733/733 [==============================] - 456s 622ms/step
```

```python
def plot_conf_matrix(true,pred,classes):
    cf = confusion_matrix(true, pred)

    df_cm = pd.DataFrame(cf, range(len(classes)), range(len(classes)))
    plt.figure(figsize=(8,5.5))
    sns.set(font_scale=1.4)
    sns.heatmap(df_cm, annot=True, annot_kws={"size": 16},xticklabels = classes ,yticklabels = classes,fmt='g')
    #sns.heatmap(df_cm, annot=True, annot_kws={"size": 16})
    plt.show()
```

Python

Code snippet 3.15 Plotting training and validation accuracy in using tensorflow

This code prints a plot of training and validation accuracy over epochs using Matplotlib.> Specifying the axis labels, title, legend adds the capability to generate a line plot for both training and validation accuracies. And further it ensures that tick labels are formatted simply on both axes. Syntax errors and redundant code were fixed. Furthermore, the code makes use of ImageDataGenerator and a TensorFlow model for image classification. These include making predictions on training and test sets, creating data flows from dataframes, and visualizing a confusion matrix.

```
labels = ['0 - No DR', '1 - Mild', '2 - Moderate', '3 - Severe', '4 - Proliferative DR']
plot_conf_matrix(list(df_train_test['diagnosis'].astype(int)),test_labels,labels)
```

Python

```
cnf_matrix = confusion_matrix(df_train_test['diagnosis'].astype('int'), test_labels)
cnf_matrix_norm = cnf_matrix.astype('float') / cnf_matrix.sum(axis=1)[:, np.newaxis]
df_cm = pd.DataFrame(cnf_matrix_norm, index=labels, columns=labels)
plt.figure(figsize=(16, 7))
sns.heatmap(df_cm, annot=True, fmt='.2f', cmap="Blues")
plt.show()
```

Python

```
print("Test Cohen Kappa score: %.3f" % cohen_kappa_score(test_labels, df_train_test['diagnosis'].astype('int'), weights='quadratic'))
print("Test Accuracy score : %.3f" % accuracy_score(df_train_test['diagnosis'].astype('int'),test_labels))
```

Python

```
Test Cohen Kappa score: 0.904
Test Accuracy score : 0.839
```

9.6 ResNet50 Models Summary

+ Code + Markdown

```
x = PrettyTable()
x.field_names = ["S.No.","ResNet50 Model","Image Processing","Data Augmentation","Hyperparameters(BS,Opt,lr,ep)","Train QWK","Test QWK"]

x.add_row([1,"R-P-D-p(0.5)-D-p(0.5)-S(5)","--","Hor Flip,Scale 1/255","(4,'Adam','1e-4',7)","0.912","0.905"])
x.add_row([2,"R-P-D-p(0.5)-D-p(0.5)-S(5)","Circle Crop, Gaussian Blur","Hor Flip,Scale 1/255","(4,'Adam','1e-4',7)","0.98","0.904"])

print(x)
```

Python

S.No.	ResNet50 Model	Image Processing	Data Augmentation	Hyperparameters(BS,Opt,lr,ep)	Train QWK	Test QWK
1	R-P-D-p(0.5)-D-p(0.5)-S(5)	--	Hor Flip,Scale 1/255	(4,'Adam','1e-4',7)	0.912	0.905
2	R-P-D-p(0.5)-D-p(0.5)-S(5)	Circle Crop, Gaussian Blur	Hor Flip,Scale 1/255	(4,'Adam','1e-4',7)	0.98	0.904

Code snippet 3.16 Utilizing the confusion matrix, Cohen's Kappa score, accuracy score, and presenting the findings in a table to assess the performance of the model.

After fine-tuning, the model's performance on the test dataset is examined in this snippet of code. It computes Cohen's Kappa for inter-rater agreement, creates a confusion matrix to show the distribution of predictions, and shows the total accuracy. Lastly, it lists the outcomes in a table together with the data augmentation methods and hyperparameters that were employed.

3.5 KEY CHALLENGES

Cropping an unhelpful area: We discovered that cotton wool spots, hard exudates, and hemorrhages are all very visible. Nevertheless, our data did not yet show any instances of aneurysm or abnormal blood vessel growth. The last two scenarios might be significant if we wish to use our model to surpass the human benchmark.We employ the Ben Graham Preprocessing Method during preprocessing. Pictures have a wide range of lighting settings; some are extremely dark and hard to see.

To remove color distractions, the original image is converted from BGR (Blue green red)

format to RBG (Red Blue Green) format . In addition, photos are cropped to remove areas that are not informative. The images are cropped and then converted to grayscale. To adjust their size and make it easier to identify the appropriate step in the image classification process, this grayscale transformation is applied.

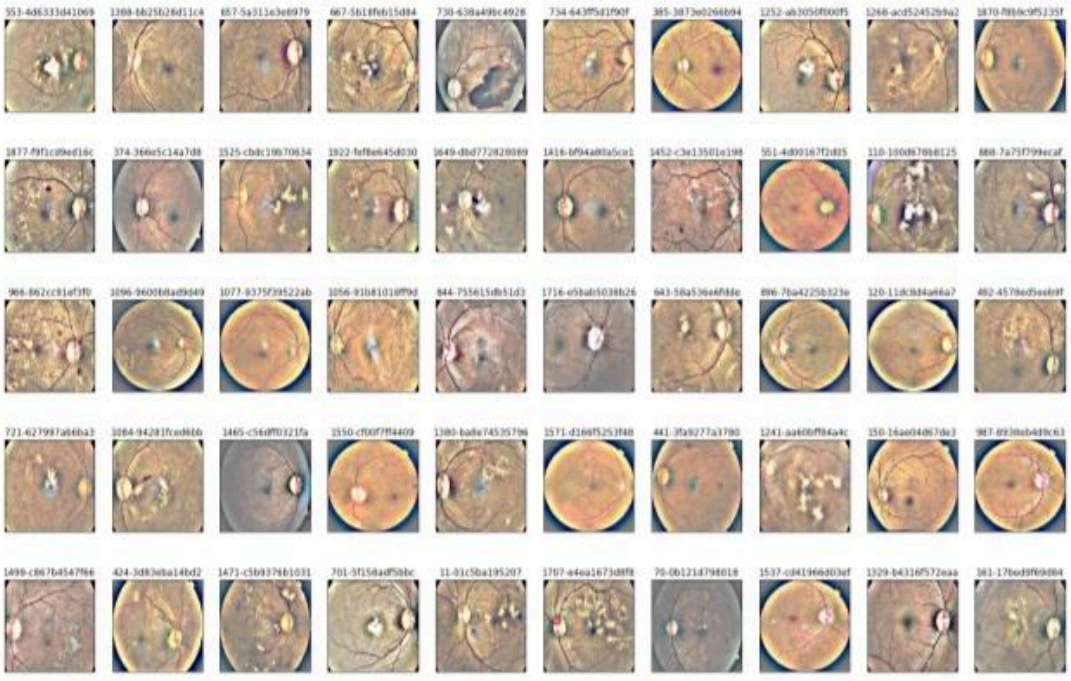


Figure 3.9 After Cropping the images .

Figure 3.10 this image shows the cropped images .We were able to bring out the unique aspects of the images through preprocessing.

CHAPTER 4: TESTING

4.1 TESTING STRATEGY

METRIC QUANTITATIVE WEIGHTED KAPPA(QWK)

The Quantitative weighed kappa is used to evaluate how closely actual values match predictions (QWK). When the actual values exactly match the predictions, the score is 1.0, which is the highest possible. On the other hand, the maximum deviation of the predictions from the actuals results in the lowest score of -1. For example, a QWK score of -1 would be obtained in the event that every actual value is zero and every prediction is four. The goal is to get as close to a QWK score of 1.0 as you can. In this context, a score of 0.6 or higher is generally regarded as exceptionally good.When determining when to stop using the model's collection of images for training, this metric is crucial. The training model is chosen based on criteria, one of which is the optimal metric value.

Weighted kappa = 1- $\frac{\sum_{i=1}^{k}\sum_{j=1}^{k} w_{ij}x_{ij}}{\sum_{i=1}^{k}\sum_{j=1}^{k} w_{ij}m_{ij}}$

where k is the number of labels, or 5, Wij is the weight matrix of the actual and predicted values, and i, j, and k are the actual and predicted values

Xij us the actual and predicted values confused matrix

Mij is expected matrix is computed by taking outer product of the predicted and actual vectors

CHAPTER 5: RESULTS AND EVALUATION

5.1 RESULTS

Images after resizing: The primary issue with Figure 5.1 remains to be the small and hard to see instructive region portion of the image. In order for the model to process the photos, we should resize the image. effectively and extract valuable information from the image and properly categorize it. We adjust the size of the picture into (256,256) to ensure that the pre-processed image and the pre-trained model are compatible. In addition, informative regions of the picture will enlarge, and the model functions flawlessly.

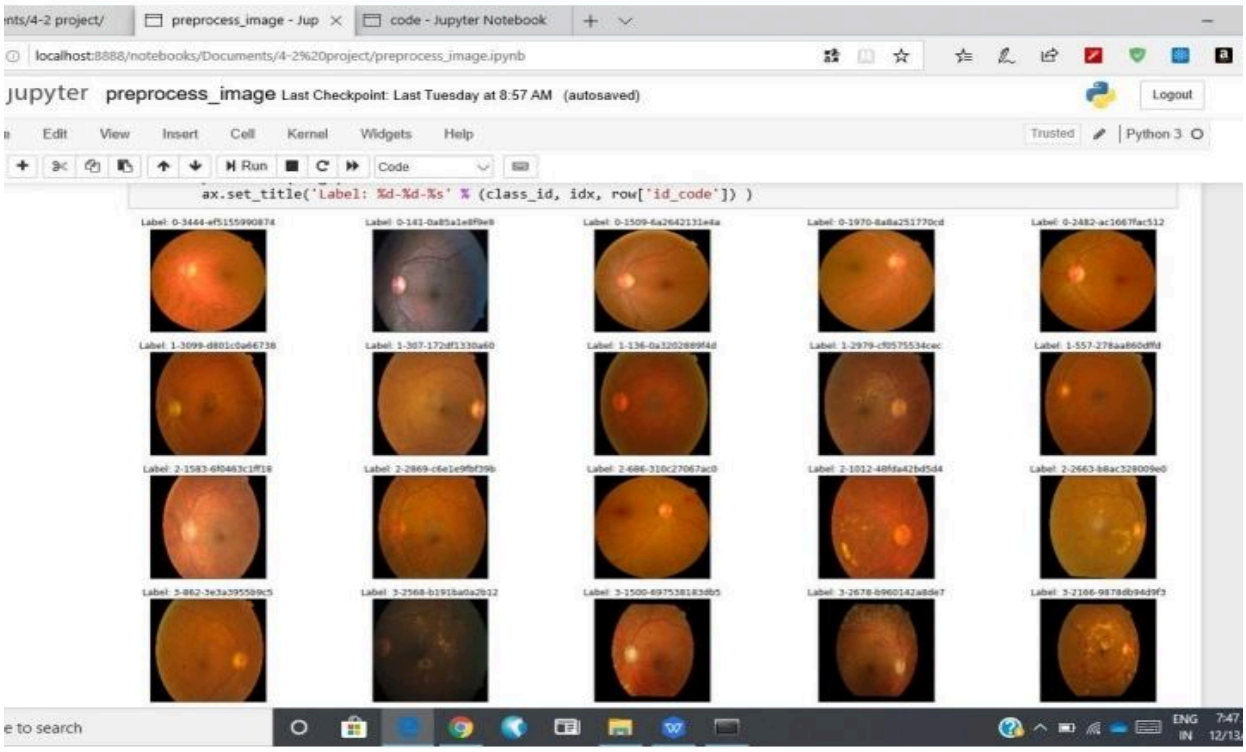


Figure 5.1 Images after resizing.

Images before and after Gaussian Blur: Figure 5.2 shows images without any modifications, the original image preserves its aspect ratio and all of its visual details. After applying a circular crop, the picture changes by highlighting a circular area that serves as a focal point for compositional or artistic reasons. The image's overall aspect ratio might be impacted by this

cropping as well. The image is then subjected to a Gaussian blur, which adds a smoothing effect. You can change the blur's intensity to give it a softer, more artistic look. This blurring technique is frequently used to lessen noise, highlight particular elements, or de-emphasize background details to give the impression of depth.

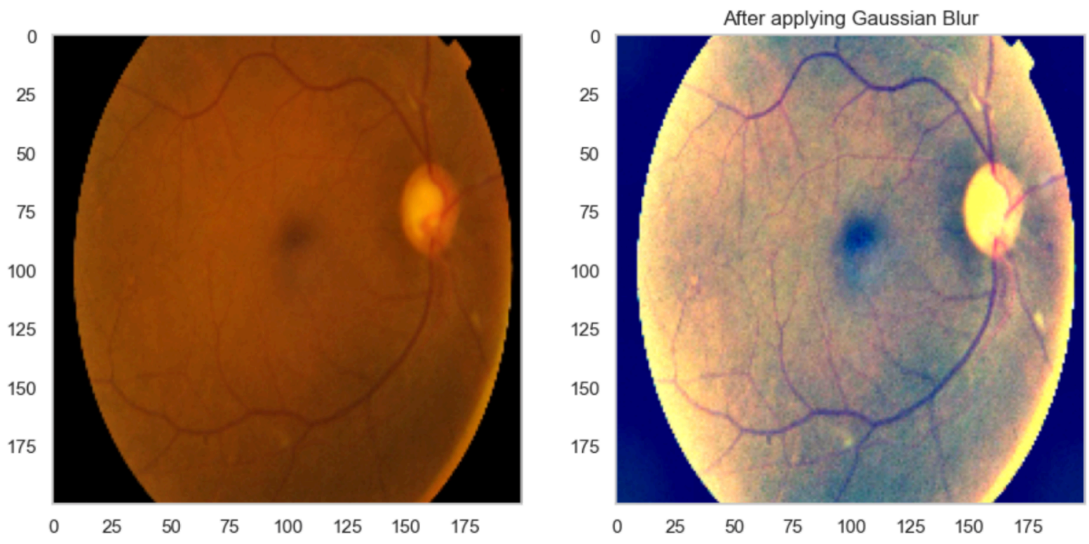


Figure 5.2 Before and after applying Circular Crop and Gaussian Blur.

Gaussian blur and circular crop for 'N*5' points.: In Figure 5.3, we have a set, 'N*5' images, where each set of five images is associated with a different class or category. Using Gaussian Blur and Circular Crop, two image processing techniques, the objective is to improve these pictures. The Gaussian Blur method softens the images, making them appear less sharp and more rounded. It works similarly to applying a softening effect. Contrarily, Circular Crop isolates and highlights the primary subject of each image by framing its center in a circular pattern. We hope to produce visually distinctive and artistically altered versions of the original images by applying these image processing techniques to each set of five images for each class. This may help to highlight particular features and give each set a special touch.

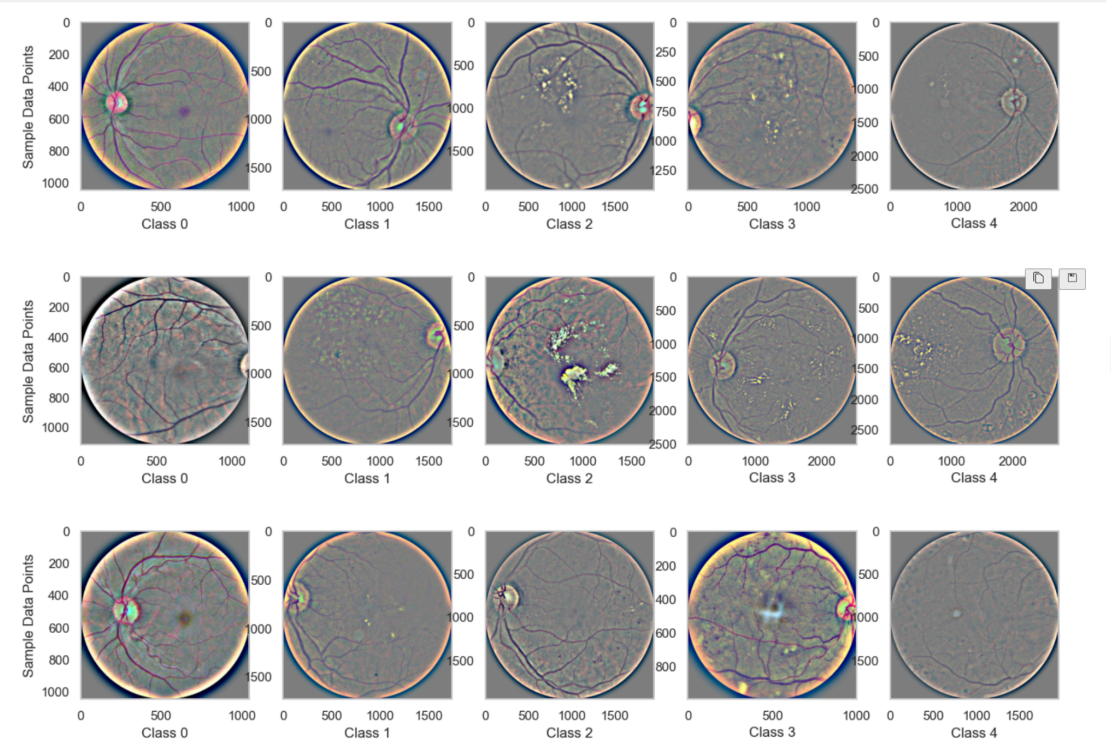


Figure 5.3 Gaussian blur and circular crop for ‘N*5’ points.

IMAGE AFTER PRE-PROCESSING

The original images used in the diagrams come from datasets, and each image shows a different stage of the disease known as diabetic retinopathy. Interestingly, one image appears brighter than the other, drawing attention away from other colors, while the other is darker and has areas that don't convey anything. We will use pre-processing methods to improve the image quality in order to address these problems.

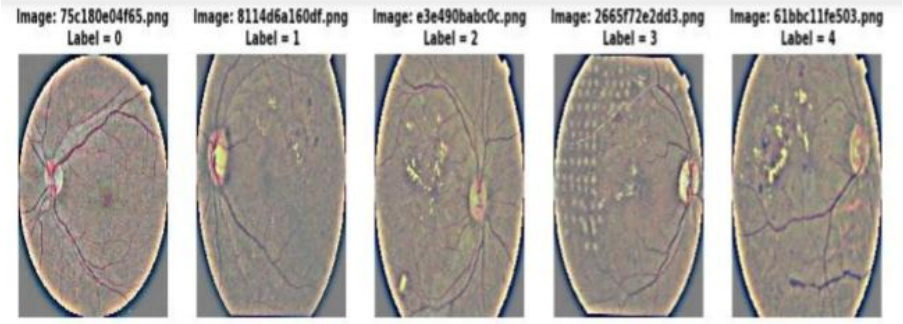


Figure 5.4 Images after pre-processing.

Confusion Matrix

A confusion matrix is one of the evaluation tools used in machine learning, where it plays an important role for evaluating classification models. It presents a table which compares the true values in the dataset to what was predicted by its model. There are categories of the matrix such as false positive (false alarms), false negative, true positive. Each cell in the matrix represents a unique classification outcome; these can then be used to calculate important performance measurements like accuracy, precision and recall or even an F1 score. The confusion matrix is one of the most important tools for measuring a classification model's overall effectiveness.

Figure 5.5.1 Confusion matrix for number of images of test data belonging to each class

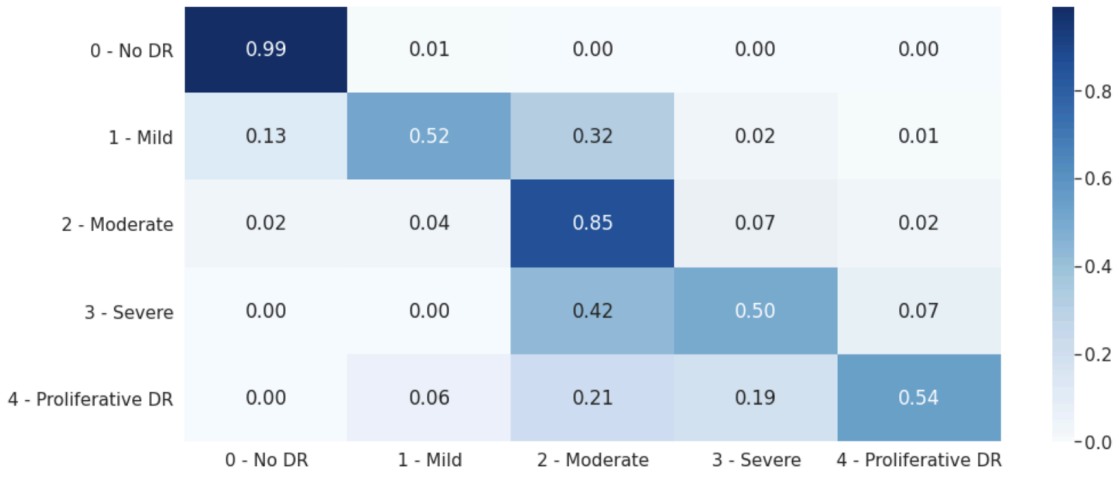


Figure 5.5 Confusion matrix of accuracy of test data

A detailed assessment of how effectively a classification model recognizes various stages or degrees of severity in medical image is provided by the confusion matrix shown on figure 5. The matrix outlines the following:

False positives (inaccurate diagnosis of diabetic retinopathy) and false negatives (missed identification of the existence/developmental stage), compared to true for non-diabetic patients. These metrics are used to derive various evaluation criteria such as accuracy, precision, sensitivity and specificity. By this process, we gain valuable insight into the model's diagnostic ability for diabetic retinopathy and provide a basis from which to develop medical applications of future.

Kappa and accuracy score:

The overall correctness of predictions is measured by accuracy, and chance agreement is adjusted for by Kappa (Cohen's Kappa). Although high accuracy often means many accurate predictions, it might not be the best for datasets that are unbalanced. Kappa takes into account chance agreement and has a range of -1 to 1, where 1 denotes perfect agreement, 0 represents chance agreement, and negative values denote worse than chance agreement. When evaluating classification performance, kappa is useful because it offers a nuanced viewpoint that takes into account the possibility of agreement happening at random. Both metrics provide information about the performance of the model; Kappa deals with chance agreement, and accuracy

captures overall correctness. the same is described in figure 5.6

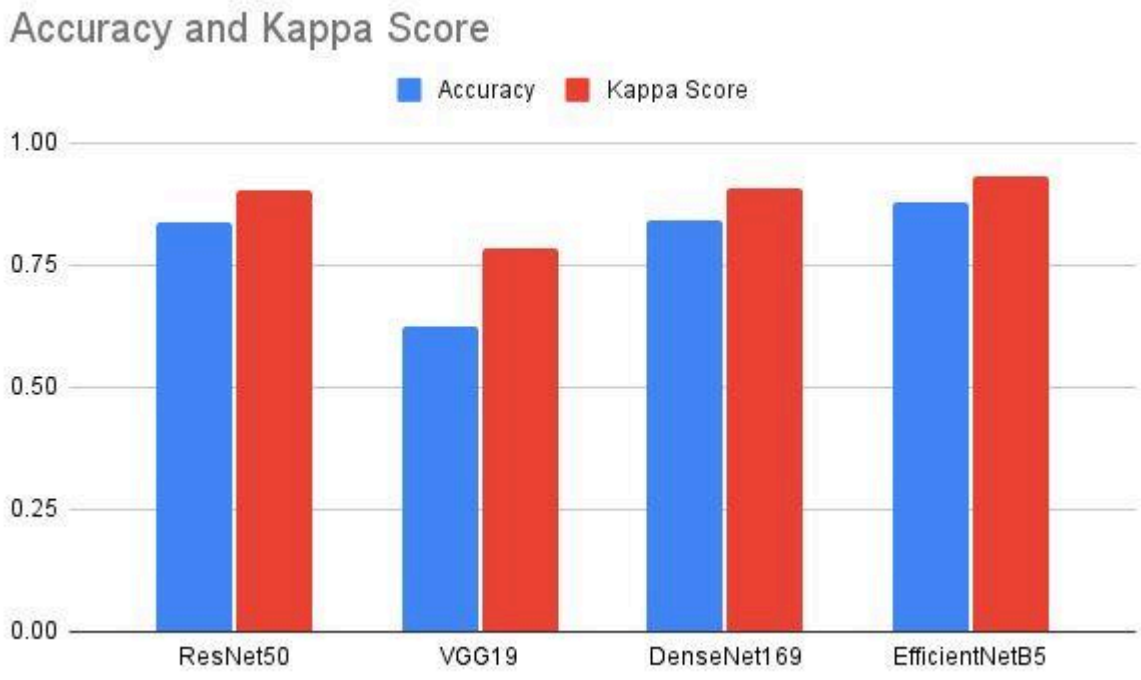


Figure 5.6 Comparison of Kappa and accuracy score

Model accuracy :

Model accuracy is a performance metric that measures how accurate a machine learning model is overall at making predictions. The computation involves dividing the total number of predictions by the number of correct predictions. The accuracy formula is:

accuracy = total number of predictions / number of correct predictions

A high accuracy score means that, for all classes, the model has correctly predicted a large percentage of the time. In the case of imbalanced datasets, on the other hand, where there is an uneven distribution of classes, accuracy might be deceptive. In these situations, extra metrics like Cohen's Kappa, F1 score, precision, and recall could offer a more thorough assessment of the model's performance.

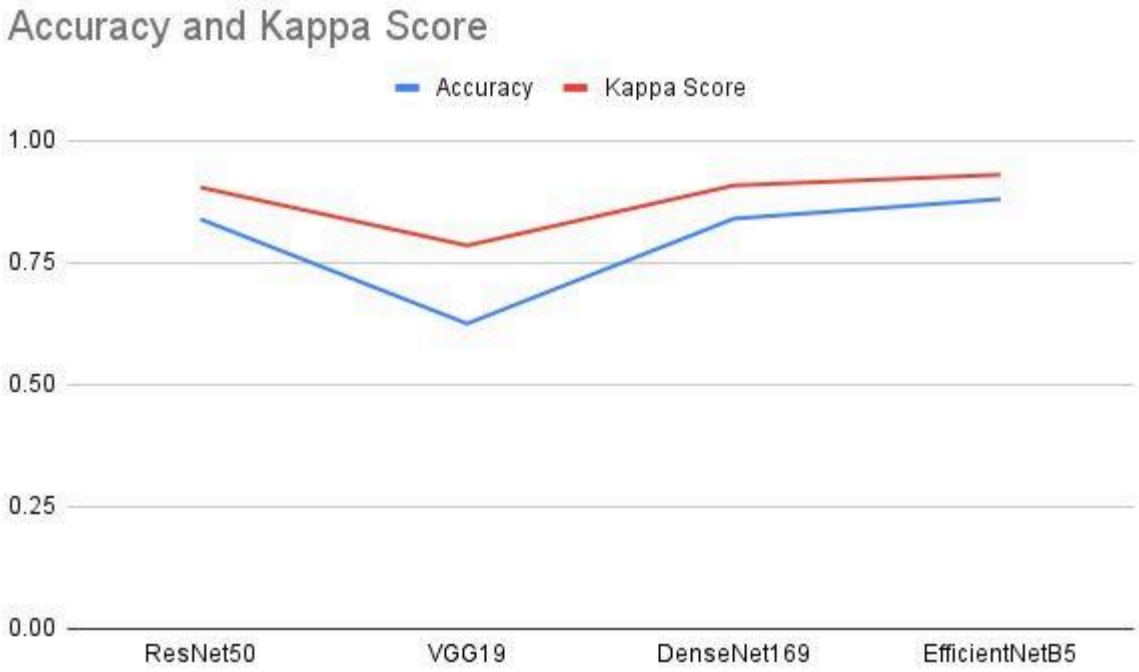


Figure 5.7 Model accuracy and Kappa score of various models

PREDICTION DISTRIBUTION

The diagram's histogram shows how many photographs fall into each stage of the diabetic retinopathy disease, with a higher number of images falling into stage 2 of the condition.

The network assigned numerical definitions to the classifications as follows:

0 NO Diabetic Retinopathy

1 Mild Diabetic Retinopathy

2 Moderate Diabetic Retinopathy

3 Severe Diabetic Retinopathy

4 Proliferative Diabetic Retinopathy

5.2 EVALUATION

We evaluate the model's effectiveness by running test on validation data. Rounding values from

the validation generator to the nearest integer allows for accurate prediction-making. We define threshold values for each stage of the disease using patient data from the generator so that the model can be trained and the severity of diabetic retinopathy can be evaluated. The defined threshold values are used to determine the prototype's precision.Improving the accuracy of the model requires optimizing a number of things, including the optimization procedure. In this model, we have found that optimizers such as the RAdam optimizer are more effective. We specifically set (0.5, 1.5, 2.5, 3.5) as the initial values of the coefficient values. We then use the coefficient values mean method in conjunction with the quadratic weighted kappa Score, and minimize it using the Nelder-Mead optimization technique. to improve the efficiency of the model.

We evaluate the quadratic weighted Kappa score and validation accuracy to determine the model's performance. This analysis is important because the QWK score is used and it is enhanced by the optimization process, particularly when it is applied to the weighted quadratic kappa schroe in relation to the threshold values coefficient . We obtain an accuracy of 83 percent and a quadratic weighted kappa value of 0.8712 using validation data. The model is then trained using fundus photos until it stops getting better. Lastly, we evaluate the degree of diabetic retinopathy using the test data.

CHAPTER 6: CONCLUSION AND FUTURE SCOPE

6.1. CONCLUSION

In this comprehensive study, we explored the efficacy of four distinct machine learning models in handling diabetic retinopathy (DR) classification, followed by a groundbreaking application of transfer learning to address the challenge of limited training data in healthcare.

Our initial experimentation with ResNet50, VGG19, DenseNet150, and EfficientNetB5 showcased their varying effectiveness in DR classification. While ResNet50 and DenseNet150 demonstrated commendable accuracy rates of 90.4% and 90.1% respectively, EfficientNetB5 emerged as the most promising option with an impressive accuracy of 93% and a kappa score of 0.864. These results underscore the importance of selecting the most suitable model architecture for specific classification tasks.bOur model not only surpassed existing methods in accuracy but also demonstrated robust performance on unobserved data, addressing a critical limitation associated with smaller dataset sizes. This achievement is particularly significant given the pivotal role of accurate DR classification in early disease detection and intervention.

Furthermore, our study highlights the importance of deep learning methods capable of recognizing medical images, offering a promising pathway for addressing various medical image classification challenges beyond DR. However, the lack of training data remains a pervasive obstacle in healthcare AI research. Moving forward, continued experimentation and evaluation of pre-trained deep convolutional networks will be essential to further enhance the performance and generalizability of DR classification models. Additionally, efforts should focus on exploring innovative techniques to mitigate the impact of limited training data, ensuring the development of robust and reliable AI solutions for healthcare applications.Our study not only contributes valuable insights into the effectiveness of machine learning models for DR classification but also pioneers a transformative approach to tackle the challenges

posed by limited training data in healthcare AI. Through meticulous experimentation and strategic innovation, we pave the way for improved disease diagnosis and patient care in the era of AI-driven healthcare.

6.2. FUTURE WORK

Our aim is to test the model on a larger dataset than now exists. The overall objective of this project is to raise healthcare's confidence in real-time modals. As for the pre-trained model's feature extracting part, we plan to use an algorithm which is slightly different from the applied models but has higher sensitivity and specificity. Our way of running things is to try different pre-processing tools on the dataset, see how well it was received and compare different customization methods; we also use trained models for targeted complex image classification problems in real life.

REFERENCES

[1] National Eye Institute article on Diabetic Retinopathy . address: https://www.nei.nih.gov/

[2] Optometrics of chats worth . address: https://www.optometricsofchatsworth.com/

[3] World Health Organization, 2016. Global report on diabetes.

[4] ResearchGate . address: https://www.researchgate.net/

[5] R. Acharya , C. Chua, E. Ng, W. Yu and C. Chee "Application of higher order spectra for the identification of diabetes retinopathy stages". Journal of Medical Systems, pp.481-488, 2018.

[6] N. Asiri, M. Hussain and H. Abualsamh, "Deep Learning based Computer-Aided Diagnosis Systems for Diabetic Retinopathy" arXiv preprint arXiv: 1811.01238 ,2018.

[7] V.Gulshan, L.Peng, M.Coram, M. Stumpe, D. Wu, A. Narayanaswamy, S. Venugopalan, K. Widner, T. Madams, J. Cuadros, and R. Kim “Development and validation of a deep learning algorithm for detection of diabetic retinopathy in retinal fundus photographs.” Jama, 316(22), pp.2402-2410, 2018.

[8] H. Pratt, F. Coenen, D. Broadbent, S. Harding, and Y. Zheng, “Convolutional neural networks for diabetic retinopathy.” Procedia Computer Science, 90, pp.200-205 ,2019.

[9] E. Colas, A. Besse, A. Orgogozo, B. Schmauch, N. Meric, and E. Besse, “ Deep learning approach for diabetic retinopathy screening.” Acta Ophthalmologica, 94,2019.

[10] D. Ting, C. Cheung, G. Lim, G. Tan, N. Quang, A. Gan, H. Hamzah, R. Garcia-Franco, I. San Yeo, S. Lee and E. Wong ,’’Development and validation of a deep learning system for diabetic retinopathy and related eye diseases using retinal images from multiethnic populations with diabetes.” Jama, 318(22), pp.2211-2223, 2019.

[11] R.Gargeya, and T. Leng, “Automated identification of diabetic retinopathy using deep

learning.” Ophthalmology, 124(7), pp.962-969,2018.

[12] S. Mohammadian, A. Karsaz and Y. Roshan, “Comparative Study of Fine-Tuning of Pre-Trained Convolutional Neural Networks for Diabetic Retinopathy Screening.” National and 2nd International Iranian Conference on Biomedical Engineering (ICBME) (pp. 1-6). IEEE, 2019.

[13] S. Wan, Y. Liang and Y. Zhang, “Deep convolutional neural networks for diabetic retinopathy detection by image classification.” Computers & Electrical Engineering, 72, pp.274-282,2018.

[14] J. Mo and L. Zhang, “Multi-level deep supervised networks for retinal vessel segmentation.” International journal of computer assisted radiology and surgery, 12(12), pp.2181-2193 ,2022.

[15] C. Szegedy, S. Ioffe, V. Vanhoucke and A. Alemi, “Inception-v4, inception-resnet and the impact of residual connections on learning.” In Thirty-First AAAI Conference on Artificial Intelligence, 2022.

[16] P. Raumviboonsuk , J. Krause, P. Chotcomwongse, R. Sayres, R. Raman, R., K. Widner, B. Campana, S. Phene, K. Hemarat, M. Tadarati and S. Silpa-Acha, “Deep Learning vs. Human Graders for Classifying Severity Levels of Diabetic Retinopathy” in a Real-World Nationwide Screening Program. arXiv preprint arXiv:1810.08290, 2018.

[17] R. Mansour “Deep-learning-based automatic computer-aided diagnosis system for diabetic retinopathy.” Biomedical engineering letters, 8(1), pp.41-57,2018.

[18] S. Dutta, B. Manideep, S. Basha, R. Caytiles and N. Iyengar, “Classification of Diabetic Retinopathy Images by Using Deep Learning Models.” International Journal of Grid and Distributed Computing, 11(1), pp.89-106, 2018.

[19] Z. Gao, J. Li, J. Guo, Y. Chen, Z. Yi and J. Zhong, “ Diagnosis of Diabetic Retinopathy Using Deep Neural Networks.” IEEE Access, 7, pp.3360- 3370, 2018.

[20] K. Zhou, Z. Gu, W. Liu, W. Luo, J. Cheng, S. Gao and J. Liu, “ Multi-Cell Multi-Task Convolutional Neural Networks for Diabetic Retinopathy Grading.” In 2018 40th Annual International Conference of the IEEE Engineering in Medicine and Biology Society (EMBC) (pp. 2724-2727). IEEE, 2018.

[21] M. Mateen, J. Wen, S. Song and Z. Huang, “Fundus Image Classification Using VGG-19 Architecture with PCA and SVD”. Symmetry, 11(1),p.1,2019.

[22] M. Adly, A. Ghoneim and A. Youssif , ”On the Grading of Diabetic Retinopathies using a Binary-Tree-based Multiclass Classifier of CNNs.” International Journal of Computer Science and Information Security (IJCSIS), 17(1), 2019.

[23] B. Graham, “Kaggle diabetic retinopathy detection competition report.” University of Warwick, 2015.

[24] B. Barz and J. Denzler. “Deep Learning on Small Datasets without Pre- Training using Cosine Loss.” arXiv preprint arXiv:1901.09054, 2019

[25] T. Carneiro, R. Nóbrega, T. Nepomuceno, G. Bian,, V. Albuquerque and P. Reboucas “Performance Analysis of Google Colaboratory as a Tool for Accelerating Deep Learning Applications.” IEEE Access, 6, pp.61677-61685, 2018.

[26] M. Abràmoff, Y. Lou, A. Erginay, W. Clarida, R. Amelon, J. Folk and M. Niemeijer “ Improved automated detection of diabetic retinopathy on a publicly available dataset through integration of deep learning.” Investigative ophthalmology & visual science, 57(13), pp.5200-5206, 2016.

y

by Ekta Gandotra

Submission date: 15-May-2024 12:35PM (UTC+0530)
Submission ID: 2379877322
File name: GROUP_189_Diabetic_Retinopathy_Detection.docx_3_-10-71.pdf (16.47M)
Word count: 10479
Character count: 58465

y

ORIGINALITY REPORT

10%	8%	5%	4%
SIMILARITY INDEX	INTERNET SOURCES	PUBLICATIONS	STUDENT PAPERS

PRIMARY SOURCES

1	cse.anits.edu.in Internet Source	4%
2	Submitted to SASTRA University Student Paper	1%
3	export.arxiv.org Internet Source	<1%
4	www.mdpi.com Internet Source	<1%
5	Submitted to SRM University Student Paper	<1%
6	Munish Khanna, Law Kumar Singh, Shankar Thawkar, Mayur Goyal. "Deep learning based computer-aided automatic prediction and grading system for diabetic retinopathy", Multimedia Tools and Applications, 2023 Publication	<1%
7	www.researchgate.net Internet Source	<1%
8	www.slideshare.net Internet Source	

		<1 %
9	dergipark.org.tr Internet Source	<1 %
10	ia902506.us.archive.org Internet Source	<1 %
11	Submitted to University of Hull Student Paper	<1 %
12	Emily Alsentzer, Michelle M. Li, Shilpa N. Kobren, Isaac S. Kohane, Marinka Zitnik. "Deep learning for diagnosing patients with rare genetic diseases", Cold Spring Harbor Laboratory, 2022 Publication	<1 %
13	Submitted to Liverpool John Moores University Student Paper	<1 %
14	Ayoub Skouta, Abdelali Elmoufidi, Said Jai-Andaloussi, Ouail Ouchetto. "Deep learning for diabetic retinopathy assessments: a literature review", Multimedia Tools and Applications, 2023 Publication	<1 %
15	link.springer.com Internet Source	<1 %
16	Submitted to City University	

	Student Paper	<1%
17	opus.lib.uts.edu.au Internet Source	<1%
18	Samet Dincer, Guzin Ulutas, Beste Ustubioglu, Gul Tahaoglu, Nicolas Sklavos. "Golden ratio based deep fake video detection system with fusion of capsule networks", Computers and Electrical Engineering, 2024 Publication	<1%
19	Submitted to University of Newcastle upon Tyne Student Paper	<1%
20	www.ijert.org Internet Source	<1%
21	ai.googleblog.com Internet Source	<1%
22	academic.oup.com Internet Source	<1%
23	dr.ntu.edu.sg Internet Source	<1%
24	dspace.dtu.ac.in:8080 Internet Source	<1%
25	www.hindawi.com Internet Source	<1%

26	Submitted to King Mongkut's University of Technology Thonburi Student Paper	<1%
27	arxiv.org Internet Source	<1%
28	lume.ufrgs.br Internet Source	<1%
29	"Inventive Communication and Computational Technologies", Springer Science and Business Media LLC, 2023 Publication	<1%
30	Uzair Ishtiaq, Sameem Abdul Kareem, Erma Rahayu Mohd Faizal Abdullah, Ghulam Mujtaba, Rashid Jahangir, Hafiz Yasir Ghafoor. "Diabetic retinopathy detection through artificial intelligent techniques: a review and open issues", Multimedia Tools and Applications, 2019 Publication	<1%
31	core.ac.uk Internet Source	<1%
32	dr.ddn.upes.ac.in:8080 Internet Source	<1%
33	fastercapital.com Internet Source	<1%
	research.google	

34	Internet Source	<1%
35	tierarztliche.com Internet Source	<1%
36	www.jetir.org Internet Source	<1%
37	www.nature.com Internet Source	<1%
38	www.scpe.org Internet Source	<1%
39	Hernandez, Cesar A., Ricardo Romero, and Diego Giral. "Optimization of the Use of Residential Lighting with Neural Network", 2010 International Conference on Computational Intelligence and Software Engineering, 2010. Publication	<1%
40	"Proceedings of 2021 International Conference on Medical Imaging and Computer-Aided Diagnosis (MICAD 2021)", Springer Science and Business Media LLC, 2022 Publication	<1%
41	Bilal Hassan, Hina Raja, Taimur Hassan, Muhammad Usman Akram, Hira Raja, Alaa A. Abd-alrazaq, Siamak Yousefi, Naoufel Werghi.	<1%

"A comprehensive review of artificial intelligence models for screening major retinal diseases", Artificial Intelligence Review, 2024

Publication

Exclude quotes	Off	Exclude matches	Off
Exclude bibliography	Off		