DIABETIC RETINAL IMAGE CLASSIFICATION USING DEEP LEARNING

ABSTRACT

Diabetic retinopathy is a serious complication of diabetes and a leading cause of blindness in adults. Early detection and diagnosis of this condition through retinal image analysis can significantly improve patient outcomes. In this study, we present a deep learning-based approach for the automated classification of retinal images as either "Diseased" or "Normal" based on the presence or absence of diabetic retinopathy.

The core of our approach lies in leveraging a convolutional neural network (CNN) architecture trained on a large dataset of labeled retinal images. We utilize transfer learning techniques, fine-tuning a pre-trained CNN model to adapt to the specific features relevant to diabetic retinopathy detection. The model is trained using a combination of retinal images annotated by medical professionals.

Once trained, the model is capable of making predictions on unseen retinal images. For each input image, the model outputs a probability score indicating the likelihood of diabetic retinopathy being present. We threshold this probability to classify images as either "Diseased" or "Normal".

To demonstrate the effectiveness of our approach, we implemented the model into a Python script. The script loads a pre-trained model, processes a single retinal image, and makes predictions regarding its status. Visual feedback is provided to the user, including the predicted classification and a confidence score.

The presented approach offers a promising tool for assisting healthcare professionals in the early detection and management of diabetic retinopathy. Future work may involve further refinement of the model architecture, integration into clinical workflows, and large-scale validation studies to assess its performance in real-world settings.

1. PROBLEM DESCRIPTION

1.1 INTRODUCTION

Data Collection

Acquiring a comprehensive dataset of retinal images is critical for training and evaluating the classification model. This dataset should encompass a wide range of retinal conditions, including different stages of diabetic retinopathy, as well as images of healthy retinas for comparison. The data collection process involves several steps:

Source Selection: Identifying sources for obtaining retinal images, such as medical institutions, research databases, or publicly available datasets like Kaggle's Diabetic Retinopathy Detection dataset.

Annotation: Retinal images need to be annotated by medical professionals to indicate the presence and severity of diabetic retinopathy. This annotation process may involve grading the images according to established classification systems like the Early Treatment Diabetic Retinopathy Study (ETDRS) scale.

Data Augmentation: To improve model generalization and robustness, data augmentation techniques such as rotation, flipping, scaling, and brightness adjustment can be applied to artificially increase the diversity of the dataset.

Data Quality Assurance: Ensuring the quality and consistency of the collected data through rigorous quality control measures, including image preprocessing to correct for artifacts or inconsistencies.

Model Development

Designing and training an effective deep learning model for retinal image classification requires careful consideration of various factors:

Model Architecture: Choosing an appropriate CNN architecture, such as VGG, Res Net, or Inception, and possibly modifying it to suit the specific characteristics of retinal images.

Transfer Learning: Leveraging pre-trained models trained on large datasets (e.g., ImageNet) and fine-tuning them on the retinal image dataset to accelerate training and improve performance.

Hyper parameter Tuning: Optimizing hyper parameters, including learning rate, batch size, and regularization techniques, through systematic experimentation to achieve optimal model performance.

Regularization Techniques: Implementing regularization techniques like dropout, batch normalization, and L2 regularization to prevent overfitting and improve model generalization.

Model Evaluation

Assessing the performance of the trained model involves several quantitative and qualitative metrics:

Accuracy: The overall proportion of correctly classified images out of the total number of images in the test dataset.

Sensitivity and Specificity: Sensitivity measures the proportion of true positive predictions (correctly identified diseased retinas) out of all actual diseased retinas, while specificity measures the proportion of true negative predictions (correctly identified normal retinas) out of all actual normal retinas.

Receiver Operating Characteristic (ROC) Curve and Area Under the Curve (AUC): Plotting the true positive rate (sensitivity) against the false positive rate (1 - specificity) and calculating the area under this curve to assess the classifier's discrimination ability across different threshold values.

Confusion Matrix: A table summarizing the classification results, including true positives, false positives, true negatives, and false negatives, to gain insights into the model's performance.

Deployment

Once the model has been trained and evaluated, deploying it into a practical tool or application involves several steps:

Integration with Clinical Workflow: Integrating the model into existing clinical workflows to streamline the process of retinal image analysis, potentially as part of electronic medical record systems or standalone diagnostic tools.

User Interface Design: Designing a user-friendly interface for healthcare professionals to interact with the model, allowing for easy input of retinal images and interpretation of classification results.

Performance Monitoring: Implementing mechanisms to monitor the model's performance in real-world settings, including tracking accuracy, false positive rate, and false negative rate over time, and updating the model periodically as new data becomes available.

Regulatory Compliance: Ensuring compliance with relevant regulations and standards governing medical device software, such as the FDA's guidelines for software as a medical device (SaMD) or the European Union's Medical Device Regulation (MDR).

Challenges and Considerations:

Data Imbalance: The imbalance between diseased and normal retinal images in the dataset can pose challenges for model training and evaluation, requiring techniques such as oversampling, under sampling, or class weighting to address.

Interpretability and Explain ability: Deep learning models are often perceived as black boxes, making it challenging to interpret their decisions. Developing methods for explaining model predictions, such as attention mechanisms or saliency maps, can enhance trust and adoption by healthcare professionals.

Ethical and Privacy Concerns: Ensuring patient privacy and confidentiality throughout the data collection, annotation, and deployment processes, including obtaining informed consent and adhering to data protection regulations like HIPAA or GDPR.

Clinical Validation: Conducting rigorous clinical validation studies to assess the real-world performance of the deployed model and its impact on patient outcomes, potentially involving collaboration with medical institutions and regulatory agencies.

2. SYSTEM STUDY

2.1 EXISTING SYSTEM

The existing system for diabetic retinal image classification may involve manual inspection of retinal images by ophthalmologists or trained medical professionals. This process typically requires the following steps:

Manual Examination: Ophthalmologists visually inspect retinal images to identify signs of diabetic retinopathy, such as micro aneurysms, hemorrhages, exudates, and neovascularization.

Grading and Diagnosis: Based on the observed abnormalities, the retinal images are graded according to established classification systems, such as the Early Treatment Diabetic Retinopathy Study (ETDRS) scale, which categorizes the severity of diabetic retinopathy into stages ranging from mild non proliferative retinopathy to severe proliferative retinopathy.

Decision Making: Ophthalmologists make diagnostic decisions and recommendations for further treatment or monitoring based on the severity of diabetic retinopathy and the presence of other risk factors.

Advantages of the Existing System

Expertise: The manual examination of retinal images by trained ophthalmologists leverages their expertise and clinical judgment, allowing for nuanced interpretation and diagnosis.

Accuracy: Ophthalmologists can provide accurate diagnoses based on a comprehensive evaluation of retinal images, taking into account various clinical factors and patient history.

Disadvantages of the Existing System

Subjectivity: Interpretation of retinal images can be subjective and prone to inter-observer variability, leading to inconsistent diagnoses and treatment recommendations.

Time-Consuming: Manual examination of retinal images is time-consuming, limiting the scalability and efficiency of diabetic retinopathy screening programs, particularly in regions with limited access to healthcare resources.

Resource Intensive: The reliance on trained ophthalmologists for retinal image analysis requires significant healthcare resources, including specialized equipment and personnel.

2.2 PROPOSED SYSTEM

The proposed system aims to address the limitations of the existing system by introducing an automated approach to diabetic retinal image classification using deep learning techniques:

Automated Image Analysis: Deep learning models, such as convolutional neural networks (CNNs), are trained on large datasets of annotated retinal images to automatically identify and classify features indicative of diabetic retinopathy.

Real-Time Classification: The trained model can analyze retinal images in real-time, providing instant classification results without the need for manual inspection by healthcare professionals.

Scalability: Automated image classification allows for scalable diabetic retinopathy screening programs, enabling broader access to screening services and reducing the burden on healthcare resources.

Advantages of the Proposed System

Efficiency: Automated image analysis streamlines the process of diabetic retinopathy screening, reducing the time and resources required for image interpretation and diagnosis.

consistency: Deep learning models offer consistent and reproducible classification results, mitigating the variability associated with manual image interpretation.

Accessibility: The automated system can be deployed in various healthcare settings, including primary care clinics, mobile health units, and telemedicine platforms, expanding access to diabetic retinopathy screening for underserved populations.

Disadvantages of the Proposed System

Algorithmic Bias: Deep learning models may exhibit biases or limitations inherent in the training data, leading to disparities in classification accuracy across demographic groups or retinal imaging modalities.

Interpretability: The black-box nature of deep learning models makes it challenging to interpret their decisions, potentially undermining trust and acceptance by healthcare professionals and patients.

Validation and Regulation: Ensuring the safety, efficacy, and regulatory compliance of automated diagnostic systems requires rigorous validation studies and adherence to medical device regulations, such as the FDA's guidelines for software as a medical device (SaMD).

3. SYSTEM CONFIGURATION

3.1 HARDWARE CONFIGURAION

Processor - I5

Speed - 3 GHz

RAM - 8 GB (min)

Hard Disk - 500 GB

Key Board - Standard Windows Keyboard

Mouse - Two or Three Button Mouse

Monitor - LCD,LED

3.2 SOFTWARE CONFIGURATION

Operating System - Windows/7/10

Server - Anaconda, Jupyter

Front End - tkinter GUI toolkit

Server side Script - Python , AIML

4.SOFTWARE DESCRIPTION

4.1 PYTHON

What is Python :-

Below are some facts about Python:

Python is currently the most widely used multi-purpose, high-level programming language. Python allows programming in Object-Oriented and Procedural paradigms. Python programs generally are smaller than other programming languages like Java. Programmers have to type relatively less and indentation requirement of the language, makes them readable all the time.

Python language is being used by almost all tech-giant companies like – Google, Amazon, Facebook, Instagram, Dropbox, Uber… etc.The biggest strength of Python is huge collection of standard library which can be used for the following –

- Machine Learning
- GUI Applications (like Kivy, Tkinter, PyQt etc.)
- Web frameworks like Django (used by YouTube, Instagram, Dropbox)
- Image processing (like Opencv, Pillow)
- Web scraping (like Scrapy, BeautifulSoup, Selenium)
- Test frameworks
- Multimedia

Advantages of Python

Let’s see how Python dominates over other languages.

Extensive Libraries

Python downloads with an extensive library and it contain code for various purposes like regular expressions, documentation-generation, unit-testing, web browsers, threading, databases, CGI, email, image manipulation, and more. So, we don’t have to write the complete code for that manually.

Extensible

As we have seen earlier, Python can be **extended to other languages**. You can write some of your code in languages like C++ or C. This comes in handy, especially in projects.

Embeddable

Complimentary to extensibility, Python is embeddable as well. You can put your Python code in your source code of a different language, like C++. This lets us add **scripting capabilities** to our code in the other language.

Improved Productivity

The language's simplicity and extensive libraries render programmers **more productive** than languages like Java and C++ do. Also, the fact that you need to write less and get more things done.

IOT Opportunities

Since Python forms the basis of new platforms like Raspberry Pi, it finds the future bright for the Internet Of Things. This is a way to connect the language with the real world.

Simple and Easy

When working with Java, you may have to create a class to print **'Hello World'**. But in Python, just a print statement will do. It is also quite **easy to learn, understand,** and **code.** This is why when people pick up Python, they have a hard time adjusting to other more verbose languages like Java.

Readable

Because it is not such a verbose language, reading Python is much like reading English. This is the reason why it is so easy to learn, understand, and code. It also does not need curly braces to define blocks, and **indentation is mandatory.** This further aids the readability of the code.

Object-Oriented

This language supports both the **procedural and object-oriented** programming paradigms. While functions help us with code reusability, classes and objects let us model the real world. A class allows the **encapsulation of data** and functions into one.

Free and Open-Source

Like we said earlier, Python is **freely available.** But not only can you **download Python** for free, but you can also download its source code, make changes to it, and even distribute it. It downloads with an extensive collection of libraries to help you with your tasks.

Portable

When you code your project in a language like C++, you may need to make some changes to it if you want to run it on another platform. But it isn't the same with Python. Here, you need to **code only once**, and you can run it anywhere. This is called **Write Once Run Anywhere (WORA)**. However, you need to be careful enough not to include any system-dependent features.

Interpreter

Lastly, we will say that it is an interpreted language. Since statements are executed one by one, **debugging is easier** than in compiled languages.

Any doubts till now in the advantages of Python? Mention in the comment section.

Advantages of Python Over Other Languages

Less Coding

Almost all of the tasks done in Python requires less coding when the same task is done in other languages. Python also has an awesome standard library support, so you don't have to search for any third-party libraries to get your job done. This is the reason that many people suggest learning Python to beginners.

Affordable

Python is free therefore individuals, small companies or big organizations can leverage the free available resources to build applications. Python is popular and widely used so it gives you better community support.

Python is for Everyone

Python code can run on any machine whether it is Linux, Mac or Windows. Programmers need to learn different languages for different jobs but with Python, you can professionally build web apps, perform data analysis and **machine learning**, automate things, do web scraping and also build games and powerful visualizations. It is an all-rounder programming language.

Disadvantages of Python

So far, we've seen why Python is a great choice for your project. But if you choose it, you should be aware of its consequences as well. Let's now see the downsides of choosing Python over another language.

Speed Limitations

We have seen that Python code is executed line by line. But since Python is interpreted, it often results in **slow execution**. This, however, isn't a problem unless speed is a focal point for the project. In other words, unless high speed is a requirement, the benefits offered by Python are enough to distract us from its speed limitations.

Weak in Mobile Computing and Browsers

While it serves as an excellent server-side language, Python is much rarely seen on the **client-side**. Besides that, it is rarely ever used to implement smartphone-based applications. One such application is called **Carbonnelle**.

Design Restrictions

As you know, Python is **dynamically-typed**. This means that you don't need to declare the type of variable while writing the code. It uses **duck-typing**. But wait, what's that? Well, it just means that if it looks like a duck, it must be a duck. While this is easy on the programmers during coding, it can **raise run-time errors**.

Underdeveloped Database Access Layers

Compared to more widely used technologies like **JDBC (Java DataBase Connectivity)** and **ODBC (Open DataBase Connectivity)**, Python's database access layers are a bit underdeveloped. Consequently, it is less often applied in huge enterprises.

Simple

No, we're not kidding. Python's simplicity can indeed be a problem. Take my example. I don't do Java, I'm more of a Python person. To me, its syntax is so simple that the verbosity of Java code seems unnecessary.

This was all about the Advantages and Disadvantages of Python Programming Language.

What is Machine Learning ?

Before we take a look at the details of various machine learning methods, let's start by looking at what machine learning is, and what it isn't. Machine learning is often categorized as a subfield of artificial intelligence, but I find that categorization can often be misleading at first brush. The study of machine learning certainly arose from research in this context, but in the data science application of machine learning methods, it's more helpful to think of machine learning as a means of building models of data.

Fundamentally, machine learning involves building mathematical models to help understand data. "Learning" enters the fray when we give these models tunable parameters that can be adapted to observed data; in this way the program can be considered to be "learning" from the data. Once these models have been fit to previously seen data, they can be used to predict and understand aspects of newly observed data. I'll leave to the reader the more philosophical digression regarding the extent to which this type of mathematical, model-based "learning" is similar to the "learning" exhibited by the human brain.Understanding the problem setting in machine learning is essential to using these tools effectively, and so we will start with some broad categorizations of the types of approaches we'll discuss here.

Categories Of Machine Leaning

At the most fundamental level, machine learning can be categorized into two main types: supervised learning and unsupervised learning.

Supervised learning involves somehow modeling the relationship between measured features of data and some label associated with the data; once this model is determined, it can be used to apply labels to new, unknown data. This is further subdivided into classification tasks and regression tasks: in classification, the labels are discrete categories, while in regression, the labels are continuous quantities. We will see examples of both types of supervised learning in the following section.

Unsupervised learning involves modeling the features of a dataset without reference to any label, and is often described as "letting the dataset speak for itself." These models include tasks such as clustering and dimensionality reduction. Clustering algorithms identify distinct groups of data, while dimensionality reduction algorithms search for more succinct representations of the data. We will see examples of both types of unsupervised learning in the following section.

Need for Machine Learning

Human beings, at this moment, are the most intelligent and advanced species on earth because they can think, evaluate and solve complex problems. On the other side, AI is still in its initial stage and haven’t surpassed human intelligence in many aspects. Then the question is that what is the need to make machine learn? The most suitable reason for doing this is, “to make decisions, based on data, with efficiency and scale”.

Lately, organizations are investing heavily in newer technologies like Artificial Intelligence, Machine Learning and Deep Learning to get the key information from data to perform several real-world tasks and solve problems. We can call it data-driven decisions taken by machines, particularly to automate the process. These data-driven decisions can be used, instead of using programing logic, in the problems that cannot be programmed inherently. The fact is that we can’t do without human intelligence, but other aspect is that we all need to solve real-world problems with efficiency at a huge scale. That is why the need for machine learning arises.

Challenges in Machine Learning

While Machine Learning is rapidly evolving, making significant strides with cybersecurity and autonomous cars, this segment of AI as whole still has a long way to go. The reason behind is that ML has not been able to overcome number of challenges. The challenges that ML is facing currently are −

Quality of data − Having good-quality data for ML algorithms is one of the biggest challenges. Use of low-quality data leads to the problems related to data preprocessing and feature extraction.

Time-Consuming task − Another challenge faced by ML models is the consumption of time especially for data acquisition, feature extraction and retrieval.

Lack of specialist persons − As ML technology is still in its infancy stage, availability of expert resources is a tough job.

No clear objective for formulating business problems − Having no clear objective and well-defined goal for business problems is another key challenge for ML because this technology is not that mature yet.

Issue of overfitting & underfitting − If the model is overfitting or underfitting, it cannot be represented well for the problem.

Curse of dimensionality − Another challenge ML model faces is too many features of data points. This can be a real hindrance.

Difficulty in deployment − Complexity of the ML model makes it quite difficult to be deployed in real life.

Applications of Machines Learning

Machine Learning is the most rapidly growing technology and according to researchers we are in the golden year of AI and ML. It is used to solve many real-world complex problems which cannot be solved with traditional approach. Following are some real-world applications of ML –

- Emotion analysis
- Sentiment analysis
- Error detection and prevention
- Weather forecasting and prediction

- Stock market analysis and forecasting
- Speech synthesis
- Speech recognition
- Customer segmentation
- Object recognition
- Fraud detection
- Fraud prevention
- Recommendation of products to customer in online shopping

How to start learning ML?

This is a rough roadmap you can follow on your way to becoming an insanely talented Machine Learning Engineer. Of course, you can always modify the steps according to your needs to reach your desired end-goal!

Step 1 – Understand the Prerequisites

In case you are a genius, you could start ML directly but normally, there are some prerequisites that you need to know which include Linear Algebra, Multivariate Calculus, Statistics, and Python. And if you don't know these, never fear! You don't need a Ph.D. degree in these topics to get started but you do need a basic understanding.

(a) Learn Linear Algebra and Multivariate Calculus

Both Linear Algebra and Multivariate Calculus are important in Machine Learning. However, the extent to which you need them depends on your role as a data scientist. If you are more focused on application heavy machine learning, then you will not be that heavily focused on maths as there are many common libraries available. But if you want to focus on R&D in Machine Learning, then mastery of Linear Algebra and Multivariate Calculus is very important as you will have to implement many ML algorithms from scratch.

(b) Learn Statistics

Data plays a huge role in Machine Learning. In fact, around 80% of your time as an ML expert will be spent collecting and cleaning data. And statistics is a field that handles the collection, analysis, and presentation of data. So it is no surprise that you need to learn it!!! Some of the key concepts in statistics that are important are Statistical Significance, Probability Distributions, Hypothesis Testing, Regression, etc. Also, Bayesian Thinking is also a very important part of ML which deals with various concepts like Conditional Probability, Priors, and Posteriors, Maximum Likelihood, etc.

(c) Learn Python

Some people prefer to skip Linear Algebra, Multivariate Calculus and Statistics and learn them as they go along with trial and error. But the one thing that you absolutely cannot skip is Python! While there are other languages you can use for Machine Learning like R, Scala, etc. Python is currently the most popular language for ML. In fact, there are many Python libraries that are specifically useful for Artificial Intelligence and Machine Learning such as Keras, TensorFlow, Scikit-learn, etc.
So if you want to learn ML, it's best if you learn Python! You can do that using various online resources and courses such as **Fork Python** available Free on Geeks for Geeks.

Step 2 – Learn Various ML Concepts

Now that you are done with the prerequisites, you can move on to actually learning ML (Which is the fun part!!!) It's best to start with the basics and then move on to the more complicated stuff. Some of the basic concepts in ML are:

(a) Terminologies of Machine Learning

- **Model –** A model is a specific representation learned from data by applying some machine learning algorithm. A model is also called a hypothesis.
- **Feature –** A feature is an individual measurable property of the data. A set of numeric features can be conveniently described by a feature vector. Feature vectors are fed as input to the model. For example, in order to predict a fruit, there may be features like color, smell, taste, etc.

- **Target (Label) –** A target variable or label is the value to be predicted by our model. For the fruit example discussed in the feature section, the label with each set of input would be the name of the fruit like apple, orange, banana, etc.
- **Training –** The idea is to give a set of inputs(features) and it's expected outputs(labels), so after training, we will have a model (hypothesis) that will then map new data to one of the categories trained on.
- **Prediction -** Once our model is ready, it can be fed a set of inputs to which it will provide a predicted output (label).

(b) Types of Machine Learning

- **Supervised Learning –** This involves learning from a training dataset with labeled data using classification and regression models. This learning process continues until the required level of performance is achieved.
- **Unsupervised Learning –** This involves using unlabelled data and then finding the underlying structure in the data in order to learn more and more about the data itself using factor and cluster analysis models.
- **Semi-supervised Learning –** This involves using unlabelled data like Unsupervised Learning with a small amount of labeled data. Using labeled data vastly increases the learning accuracy and is also more cost-effective than Supervised Learning.
- **Reinforcement Learning –** This involves learning optimal actions through trial and error. So the next action is decided by learning behaviors that are based on the current state and that will maximize the reward in the future.

Advantages of Machine learning

Easily identifies trends and patterns

Machine Learning can review large volumes of data and discover specific trends and patterns that would not be apparent to humans. For instance, for an e-commerce website like Amazon, it serves to understand the browsing behaviors and purchase histories of its users to help cater to the right products, deals, and reminders relevant to them. It uses the results to reveal relevant advertisements to them.

No human intervention needed (automation)

With ML, you don't need to babysit your project every step of the way. Since it means giving machines the ability to learn, it lets them make predictions and also improve the algorithms on their own. A common example of this is anti-virus softwares; they learn to filter new threats as they are recognized. ML is also good at recognizing spam.

Continuous Improvement

As **ML algorithms** gain experience, they keep improving in accuracy and efficiency. This lets them make better decisions. Say you need to make a weather forecast model. As the amount of data you have keeps growing, your algorithms learn to make more accurate predictions faster.

Handling multi-dimensional and multi-variety data

Machine Learning algorithms are good at handling data that are multi-dimensional and multi-variety, and they can do this in dynamic or uncertain environments.

Wide Applications

You could be an e-tailer or a healthcare provider and make ML work for you. Where it does apply, it holds the capability to help deliver a much more personal experience to customers while also targeting the right customers.

Disadvantages of Machine Learning

Data Acquisition

Machine Learning requires massive data sets to train on, and these should be inclusive/unbiased, and of good quality. There can also be times where they must wait for new data to be generated.

Time and Resources

ML needs enough time to let the algorithms learn and develop enough to fulfill their purpose with a considerable amount of accuracy and relevancy. It also needs massive resources to function. This can mean additional requirements of computer power for you.

Interpretation of Results

Another major challenge is the ability to accurately interpret results generated by the algorithms. You must also carefully choose the algorithms for your purpose.

High error-susceptibility

Machine Learning is autonomous but highly susceptible to errors. Suppose you train an algorithm with data sets small enough to not be inclusive. You end up with biased predictions coming from a biased training set. This leads to irrelevant advertisements being displayed to customers. In the case of ML, such blunders can set off a chain of errors that can go undetected for long periods of time. And when they do get noticed, it takes quite some time to recognize the source of the issue, and even longer to correct it.

- Views and iterators instead of lists
- The rules for ordering comparisons have been simplified. E.g. a heterogeneous list cannot be sorted, because all the elements of a list must be comparable to each other.
- There is only one integer type left, i.e. int. long is int as well.
- The division of two integers returns a float instead of an integer. "//" can be used to have the "old" behaviour.
- Text Vs. Data Instead Of Unicode Vs. 8-bit

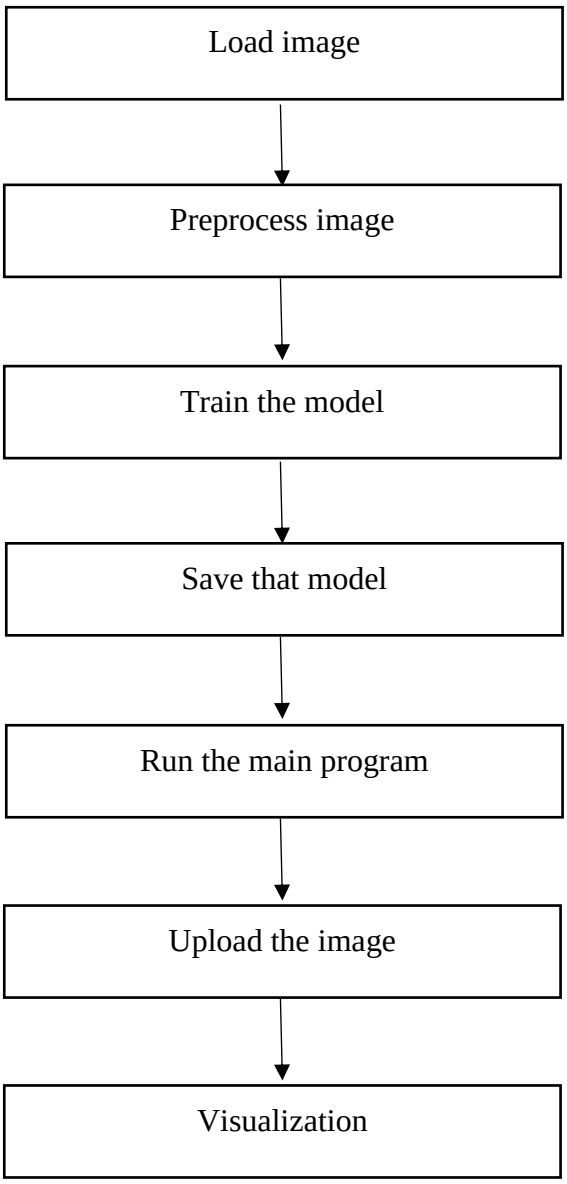
4.2SUPPORTED PLATFORMS

Requirement	**Minimum**	**Recommended**
RAM	4 GB of free RAM	8 GB of total system RAM
CPU	Any modern CPU	Multi-core CPU. PyCharm supports multithreading for different operations and processes making it faster the more CPU cores it can use.
Disk space	2.5 GB and another 1 GB for caches	SSD drive with at least 5 GBof free space
Monitor resolution	1024x768	1920×1080
Operating system	Officially released 64-bit versions of thefollowing: Microsoft Windows 8 or latermacOS 10.13 or later Any Linux distribution that supports Gnome, KDE, or Unity DE. PyCharm is not available for some Linux distributions, suchas RHEL6 or CentOS6, that do not include GLIBC 2.14 or later. Pre-release versions are not supported.	Latest 64-bit version of Windows, macOS, or Linux(for example, Debian, Ubuntu, or RHEL)

5.SYSTEM DESIGN

DATA FLOW DIAGRAM

`



6. SOURCE CODE

Main.py

```
import numpy as np
import matplotlib.pyplot as plt
import tensorflow as tf
from tensorflow.keras import layers, models
from tensorflow.keras.preprocessing.image import ImageDataGenerator
# Load and preprocess your retinal image dataset
train_datagen = ImageDataGenerator(rescale=1./255)
test_datagen = ImageDataGenerator(rescale=1./255)
train_generator = train_datagen.flow_from_directory(
        'sample',
        target_size=(150, 150),
        batch_size=32,
        class_mode='binary')
validation_generator = test_datagen.flow_from_directory(
        'sample',
        target_size=(150, 150),
        batch_size=32,
        class_mode='binary')

# Define your CNN model
def create_model(input_shape):
    model = models.Sequential([
        layers.Conv2D(32, (3, 3), activation='relu', input_shape=input_shape),
        layers.MaxPooling2D((2, 2)),
        layers.Conv2D(64, (3, 3), activation='relu'),
        layers.MaxPooling2D((2, 2)),
         layers.Conv2D(64, (3, 3), activation='relu'),
```

```
        layers.Flatten(),
        layers.Dense(64, activation='relu'),
        layers.Dense(1, activation='sigmoid') ])
    return model
# Create and compile the model
input_shape = (150, 150, 3)
model = create_model(input_shape)
model.compile(optimizer='adam',
              loss='binary_crossentropy',
              metrics=['accuracy'])


# Train the model
# Train the model
history = model.fit(
      train_generator,
      steps_per_epoch=int(train_generator.samples/train_generator.batch_size),
      epochs=10,
      validation_data=validation_generator,
      validation_steps=int(validation_generator.samples/validation_generator.batch_size))
# Load and preprocess your retinal image test dataset
test_generator = test_datagen.flow_from_directory(
        'sample',
        target_size=(150, 150),
        batch_size=32,
        class_mode='binary’
# Evaluate the model
test_loss, test_accuracy = model.evaluate(test_generator,
steps=test_generator.samples/test_generator.batch_size)
print("Test Loss:", test_loss)
print("Test Accuracy:", test_accuracy)
```

```
test_steps_per_epoch = test_generator.samples // test_generator.batch_size
# Evaluate the model
test_loss, test_accuracy = model.evaluate(test_generator, steps=test_steps_per_epoch)
print("Test Loss:", test_loss)
print("Test Accuracy:", test_accuracy)
# Plot training history
plt.plot(history.history['accuracy'], label='accuracy')
plt.plot(history.history['val_accuracy'], label = 'val_accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.ylim([0, 1])
plt.legend(loc='lower right')
plt.show()
```

train.py

```
import numpy as np
import matplotlib.pyplot as plt
import tensorflow as tf
from tensorflow.keras import layers, models
from tensorflow.keras.preprocessing.image import ImageDataGenerator
# Load and preprocess your retinal image dataset
train_datagen = ImageDataGenerator(rescale=1./255)
train_generator = train_datagen.flow_from_directory(
        'sample',
        target_size=(150, 150),
        batch_size=32,
        class_mode='binary')
# Define your CNN model
def create_model(input_shape):
    model = models.Sequential([
```

```
        layers.Conv2D(32, (3, 3), activation='relu', input_shape=input_shape),
        layers.MaxPooling2D((2, 2)),
        layers.Conv2D(64, (3, 3), activation='relu'),
        layers.MaxPooling2D((2, 2)),
        layers.Conv2D(64, (3, 3), activation='relu'),
        layers.Flatten(),
        layers.Dense(64, activation='relu'),
        layers.Dense(1, activation='sigmoid')])
    return model
# Create and compile the model
input_shape = (150, 150, 3)
model = create_model(input_shape)
model.compile(optimizer='adam',
              loss='binary_crossentropy',
              metrics=['accuracy'])
# Train the model
history = model.fit(
      train_generator,
      steps_per_epoch=int(train_generator.samples/train_generator.batch_size),
      epochs=30)
# Save the model to a file
model.save("retinal_model.h5")
```

model.py

```
# Importing all packages
import numpy as np
import matplotlib.pyplot as plt
from torch.utils import data
import torch
from torch import nn
```

```
from torch import optim
import torchvision
import torch.nn.functional as F
from torchvision import datasets, transforms, models
import torchvision.models as models
from PIL import Image, ImageFile
import json
from torch.optim import lr_scheduler
import random
import os
import system
print('Imported packages')
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
model = models.resnet152(pretrained=False)
num_ftrs = model.fc.in_features
out_ftrs = 5
model.fc = nn.Sequential(nn.Linear(num_ftrs,
512),nn.ReLU(),nn.Linear(512,out_ftrs),nn.LogSoftmax(dim=1))
criterion = nn.NLLLoss()
optimizer = torch.optim.Adam(filter(lambda p:p.requires_grad,model.parameters()) , lr = 0.00001)

scheduler = lr_scheduler.StepLR(optimizer, step_size=5, gamma=0.1)
model.to(device);
# to unfreeze more layers
for name,child in model.named_children():
    if name in ['layer2','layer3','layer4','fc']:
        #print(name + 'is unfrozen')
        for param in child.parameters():
            param.requires_grad = True
    else:
        #print(name + 'is frozen')
```

```
        for param in child.parameters():
            param.requires_grad = False
optimizer = torch.optim.Adam(filter(lambda p:p.requires_grad,model.parameters()) , lr = 0.000001)
scheduler = lr_scheduler.StepLR(optimizer, step_size=5, gamma=0.1)
def load_model(path):
    checkpoint = torch.load(path,map_location='cpu')
    model.load_state_dict(checkpoint['model_state_dict'])
    optimizer.load_state_dict(checkpoint['optimizer_state_dict'])
    return model
def inference(model, file, transform, classes):
    file = Image.open(file).convert('RGB')
    img = transform(file).unsqueeze(0)
    print('Transforming your image...')
    device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
    model.eval()
    with torch.no_grad():
        print('Passing your image to the model....')
        out = model(img.to(device))
        ps = torch.exp(out)
        top_p, top_class = ps.topk(1, dim=1)
        value = top_class.item()
        print("Predicted Severity Value: ", value)
        print("class is: ", classes[value])
        print('Your image is printed:')
        return value, classes[value]
        # plt.imshow(np.array(file))
        # plt.show()
model = load_model('../Desktop/classifier.pt')
print("Model loaded Succesfully")
classes = ['No DR', 'Mild', 'Moderate', 'Severe', 'Proliferative DR']
test_transforms = torchvision.transforms.Compose([
```

```
    torchvision.transforms.Resize((224, 224)),
    torchvision.transforms.RandomHorizontalFlip(p=0.5),
    torchvision.transforms.ToTensor(),
    torchvision.transforms.Normalize(mean=(0.485, 0.456, 0.406), std=(0.229, 0.224, 0.225))])
def main(path):
    x, y = inference(model, path, test_transforms, classes)
    return x, y
# if __name__ == '__model__':
#     # test_dir = '../Desktop/eye'
#     # folders = os.listdir(test_dir)
#     # for num in range(len(folders)):
#     #     path = test_dir+"/"+folders[num]
#     #     print(path)
#     #     inference(model, path, test_transforms, classes)
#     l = sys.argv
#     if(len(l)>1):
#         for i in range(1, len(l)):
#             print(l[i])
#             path = l[i]
#             inference(model, path, test_transforms, classes)
#     else:
#         print('please provide the exact path of image !')

_API_MODULE = _sys.modules[__name__].bitwise
_tf_api_dir = _os.path.dirname(_os.path.dirname(_API_MODULE.__file__))
_current_module = _sys.modules[__name__]

if not hasattr(_current_module, "__path__"):
  __path__ = [_tf_api_dir]
elif _tf_api_dir not in __path__:
  __path__.append(_tf_api_dir)
```

```
# Hook external TensorFlow modules.
# Import compat before trying to import summary from tensorboard, so that
# reexport_tf_summary can get compat from sys.modules. Only needed if using
# lazy loading.
_current_module.compat.v2  # pylint: disable=pointless-statement
try:
  from tensorboard.summary._tf import summary
  _current_module.__path__ = (
      [_module_util.get_parent_dir(summary)] + _current_module.__path__)
  setattr(_current_module, "summary", summary)
except ImportError:
  _logging.warning(
      "Limited tf.summary API due to missing TensorBoard installation.")
# Load tensorflow-io-gcs-filesystem if enabled
if (_os.getenv("TF_USE_MODULAR_FILESYSTEM", "0") == "true" or
    _os.getenv("TF_USE_MODULAR_FILESYSTEM", "0") == "1"):
  import tensorflow_io_gcs_filesystem as _tensorflow_io_gcs_filesystem

# Lazy-load estimator.
_estimator_module = "tensorflow_estimator.python.estimator.api._v2.estimator"
estimator = _LazyLoader("estimator", globals(), _estimator_module)
_module_dir = _module_util.get_parent_dir_for_name(_estimator_module)
if _module_dir:
  _current_module.__path__ = [_module_dir] + _current_module.__path__
setattr(_current_module, "estimator", estimator)

# Lazy-load Keras v2/3.
_tf_uses_legacy_keras = (
    _os.environ.get("TF_USE_LEGACY_KERAS", None) in ("true", "True", "1"))
setattr(_current_module, "keras", _KerasLazyLoader(globals()))
```

```
_module_dir = _module_util.get_parent_dir_for_name("keras._tf_keras.keras")
_current_module.__path__ = [_module_dir] + _current_module.__path__
if _tf_uses_legacy_keras:
  _module_dir = _module_util.get_parent_dir_for_name("tf_keras.api._v2.keras")
else:
  _module_dir = _module_util.get_parent_dir_for_name("keras.api._v2.keras")
_current_module.__path__ = [_module_dir] + _current_module.__path__


# Enable TF2 behaviors
from tensorflow.python.compat import v2_compat as _compat
_compat.enable_v2_behavior()
_major_api_version = 2


# Load all plugin libraries from site-packages/tensorflow-plugins if we are
# running under pip.
# TODO(gunan): Find a better location for this code snippet.
from tensorflow.python.framework import load_library as _ll
from tensorflow.python.lib.io import file_io as _fi

# Get sitepackages directories for the python installation.
_site_packages_dirs = []
if _site.ENABLE_USER_SITE and _site.USER_SITE is not None:
  _site_packages_dirs += [_site.USER_SITE]
_site_packages_dirs += [p for p in _sys.path if "site-packages" in p]
if "getsitepackages" in dir(_site):
  _site_packages_dirs += _site.getsitepackages()

if "sysconfig" in dir(_distutils):
  _site_packages_dirs += [_distutils.sysconfig.get_python_lib()]
```

```
_site_packages_dirs = list(set(_site_packages_dirs))

# Find the location of this exact file.
_current_file_location = _inspect.getfile(_inspect.currentframe())

def _running_from_pip_package():
  return any(
      _current_file_location.startswith(dir_) for dir_ in _site_packages_dirs)

if _running_from_pip_package():
  # TODO(gunan): Add sanity checks to loaded modules here.

  # Load first party dynamic kernels.
  _tf_dir = _os.path.dirname(_current_file_location)
  _kernel_dir = _os.path.join(_tf_dir, "core", "kernels")
  if _os.path.exists(_kernel_dir):
    _ll.load_library(_kernel_dir)

  # Load third party dynamic kernels.
  for _s in _site_packages_dirs:
    _plugin_dir = _os.path.join(_s, "tensorflow-plugins")
    if _os.path.exists(_plugin_dir):
      _ll.load_library(_plugin_dir)
      # Load Pluggable Device Library
      _ll.load_pluggable_device_library(_plugin_dir)

if _os.getenv("TF_PLUGGABLE_DEVICE_LIBRARY_PATH", ""):
  _ll.load_pluggable_device_library(
      _os.getenv("TF_PLUGGABLE_DEVICE_LIBRARY_PATH"))
```

```
# Add Keras module aliases
_losses = _KerasLazyLoader(globals(), submodule="losses", name="losses")
_metrics = _KerasLazyLoader(globals(), submodule="metrics", name="metrics")
_optimizers = _KerasLazyLoader(
    globals(), submodule="optimizers", name="optimizers")
_initializers = _KerasLazyLoader(
    globals(), submodule="initializers", name="initializers")
setattr(_current_module, "losses", _losses)
setattr(_current_module, "metrics", _metrics)
setattr(_current_module, "optimizers", _optimizers)
setattr(_current_module, "initializers", _initializers)

# Do an eager load for Keras' code so that any function/method that needs to
# happen at load time will trigger, eg registration of optimizers in the
# SavedModel registry.
# See b/196254385 for more details.
try:
  if _tf_uses_legacy_keras:
    importlib.import_module("tf_keras.src.optimizers")
  else:
    importlib.import_module("keras.src.optimizers")
except (ImportError, AttributeError):
  pass

del importlib

# Explicitly import lazy-loaded modules to support autocompletion.
if _typing.TYPE_CHECKING:
  from tensorflow_estimator.python.estimator.api._v2 import estimator as estimator

# pylint: enable=undefined-variable
```

```
# Delete modules that should be hidden from dir().
# Don't fail if these modules are not available.
# For e.g. this file will be originally placed under tensorflow/_api/v1 which
# does not have "python", "core" directories. Then, it will be copied
# to tensorflow/ which does have these two directories.
# pylint: disable=undefined-variable
try:
  del python
except NameError:
  pass
try:
  del core
except NameError:
  pass
try:
  del compiler
except NameError:
  pass
_names_with_underscore = ['__compiler_version__', '__cxx11_abi_flag__', '__cxx_version__',
'__git_version__', '__internal__', '__monolithic_build__', '__operators__', '__version__']
__all__ = [_s for _s in dir() if not _s.startswith('_')]
__all__.extend([_s for _s in _names_with_underscore])
```

7. SYSTEM TESTING

Testing is vital to the success of the system. System testing makes a logical assumption that if all parts of the system are correct, the goal will be successfully achieved. In the testing process we test the actual system in an organization and gather errors from the new system operates in full efficiency as stated. System testing is the stage of implementation, which is aimed to ensuring that the system works accurately and efficiently.

In the testing process we test the actual system in an organization and gather errors from the new system and take initiatives to correct the same. All the front-end and back-end connectivity are tested to be sure that the new system operates in full efficiency as stated. System testing is the stage of implementation, which is aimed at ensuring that the system works accurately and efficiently.

The main objective of testing is to uncover errors from the system. For the uncovering process we have to give proper input data to the system. So we should have more conscious to give input data. It is important to give correct inputs to efficient testing.

Testing is done for each module. After testing all the modules, the modules are integrated and testing of the final system is done with the test data, specially designed to show that the system will operate successfully in all its aspects conditions. Thus the system testing is a confirmation that all is correct and an opportunity to show the user that the system works. Inadequate testing or non- testing leads to errors that may appear few months later.

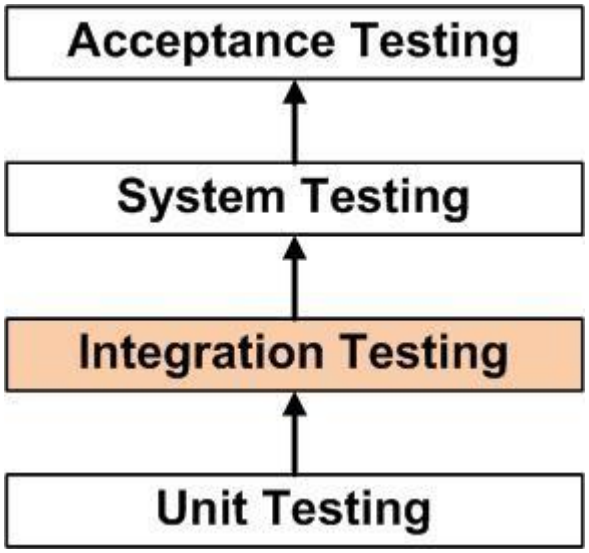
This will create two problems, Time delay between the cause and appearanceof the problem. The effect of the system errors on files and records within the system. The purpose of the system testing is to consider all the likely variations to which it will be suggested and push the system to its limits.

The testing process focuses on logical intervals of the software ensuring thatall the statements have been tested and on the function intervals (i.e.,) conducting tests to uncover errors and ensure that defined inputs will produce actual results that agree with the required results. Testing has to be done using the two common steps Unit testing and Integration testing. In the project system testing is made as follows:

The procedure level testing is made first. By giving improper inputs, the errorsoccurred are noted and eliminated. This is the final step in system life cycle. Here weimplement the tested error-free system into real-life environment and make necessary changes, which runs in an online fashion. Here system maintenance is done every months or year based on company

policies, and is checked for errors likeruntime errors, long run errors and other maintenances like table verification and reports.

Integration Testing is a level of software testing where individual units are combinedand tested as a group.



The purpose of this level is to expose faults in the interaction between integrated units. Test drivers and test stubs are used to assit in integration testing.

7.1 METHOD

Any of Black Box Testing, White Box Testing, and Gray Box Testing methods can be used. Normally, the method depends on your definition of ‘unit’.

7.2 TASKS

- Integration Test Plan
  - Prepare
  - Review
  - Rework
  - Baseline
- Integration Test Cases/Scripts
  - Prepare
  - Review
  - Rework

- Baseline

➢ Integration Test

- Perform

7.3 UNIT TESTING

Unit testing verification efforts on the smallest unit of software design, module. This is known as “Module Testing”. The modules are tested separately. Thistesting is carried out during programming stage itself. In these testing steps, each module is found to be working satisfactorily as regard to the expected output from the module.

7.4 BLACK BOX TESTING

Black box testing, also known as Behavioral Testing, is a software testing method in which the internal structure/ design/ implementation of the item being tested is not known to the tester. These tests can be functional or non-functional, though usuallyfunctional.

7.5 WHITE-BOX TESTING

White-box testing (also known as clear box testing, glass box testing, transparent box testing, and structural testing) is a method of testing software that tests internal structures or workings of an application, as opposed to its functionality (i.e. black-box testing).

7.6 GREY BOX TESTING

Grey box testing is a technique to test the application with having a limited knowledge of the internal workings of an application. To test the Web Servicesapplication usually the Grey box testing is used. Grey box testing is performed by end-users and also by testers and developers.

7.7 INTEGRATION TESTING:

Integration testing is a systematic technique for constructing tests to uncover error associated within the interface. In the project, all the modules are combined and then the entire programmer is tested as a whole.

In the integration-testing step, all the error uncovered is corrected for the nexttesting steps.

Software integration testing is the incremental integration testing of two or more integrated software components on a single platform to produce failurescaused by interface defects.

The task of the integration test is to check that components orsoftware applications, e.g. components in a software system or – one step up – software applications at the company level – interact without error.

7.8 ACCEPTANCE TESTING

User Acceptance Testing is a critical phase of any project and requires significant participation by the end user. It also ensures that the system meets the functional requirements.

Acceptance testing for Data Synchronization:

- The Acknowledgements will be received by the Sender Node after the Packets are received by the Destination Node
- The Route add operation is done only when there is a Route request in need
- The Status of Nodes information is done automatically in the Cache Updating process

7.9 BUILD THE TEST PLAN:

Any project can be divided into units that can be further performed for detailed processing. Then a testing strategy for each of this unit is carried out. Unit testing helps to identity the possible bugs in the individual component, so the component that has bugs can be identified and can be rectified from error.

8. SYSTEM IMPLEMENTATION

8.1 Data Collection and Preprocessing

Data Acquisition: Obtain a diverse dataset of retinal images, including both diseased (diabetic retinopathy) and normal retinas. This dataset should be large enough to train a robust deep learning model.

Data Annotation: Annotate the retinal images to indicate the presence and severity of diabetic retinopathy. This step may involve expert ophthalmologists grading the images according to established classification systems.

Data Preprocessing: Preprocess the retinal images to standardize their size, format, and quality. Common preprocessing steps include resizing, normalization, and augmentation to enhance the diversity of the dataset.

8.2 Model Development

Model Selection: Choose a suitable deep learning architecture for retinal image classification, such as convolutional neural networks (CNNs). Consider pre-trained models like VGG, ResNet, or Inception for transfer learning.

Transfer Learning: Fine-tune the pre-trained CNN models on the retinal image dataset to adapt them to the specific features of diabetic retinopathy. This step helps accelerate training and improve model performance.

Hyper parameter Tuning: Experiment with different hyper parameters, including learning rate, optimizer, batch size, and regularization techniques, to optimize the model's performance on the validation set.

8.3 Model Training and Evaluation

Training: Train the deep learning model using the annotated retinal image dataset. Monitor training metrics such as loss and accuracy to assess model convergence and performance.

Validation: Evaluate the trained model on a separate validation dataset to assess its generalization ability and identify potential overfitting issues.

Performance Metrics: Calculate evaluation metrics such as accuracy, sensitivity, specificity, and area under the ROC curve (AUC-ROC) to quantify the model's performance in classifying diseased and normal retinas.

8.4 Deployment

Model Serialization: Save the trained model and its associated weights to disk for later deployment. Common formats for model serialization include HDF5 (.h5) or Tensor Flow Saved Model.

Integration: Integrate the trained model into a software application or platform suitable for deployment in clinical settings. This may involve developing a graphical user interface (GUI) for easy interaction with healthcare professionals.

Testing: Perform thorough testing of the deployed system to ensure its functionality, reliability, and performance. Test various scenarios and edge cases to validate the system's robustness.

Validation Studies: Conduct clinical validation studies to assess the real-world performance of the deployed system, including its impact on diagnostic accuracy, efficiency, and patient outcomes.

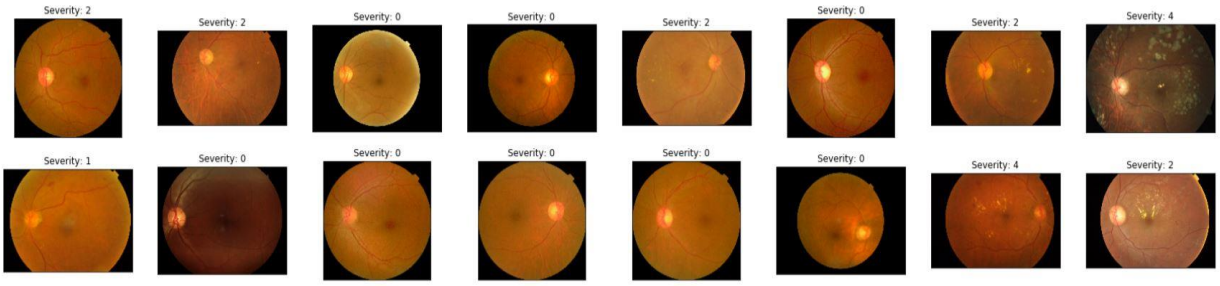
8.5 Continuous Improvement

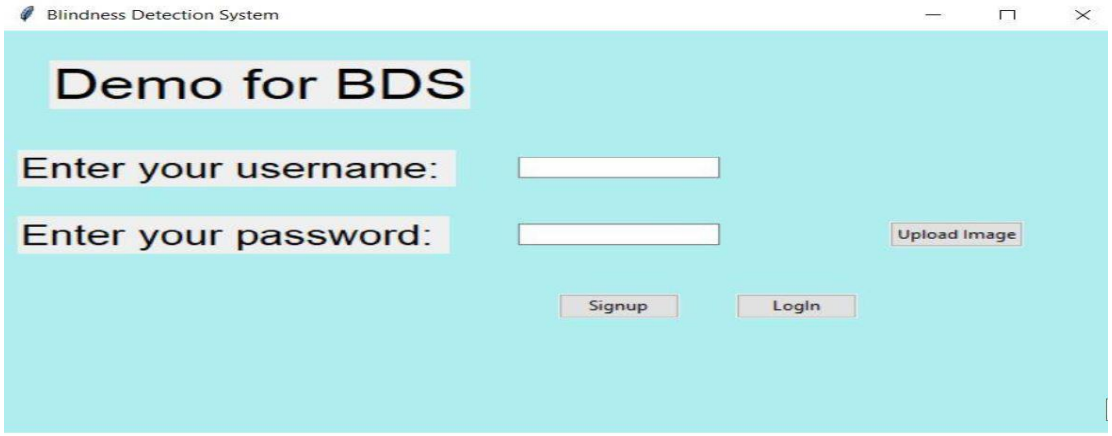
Monitoring and Maintenance: Monitor the deployed system's performance over time and implement regular updates and maintenance to address any issues or improvements identified.

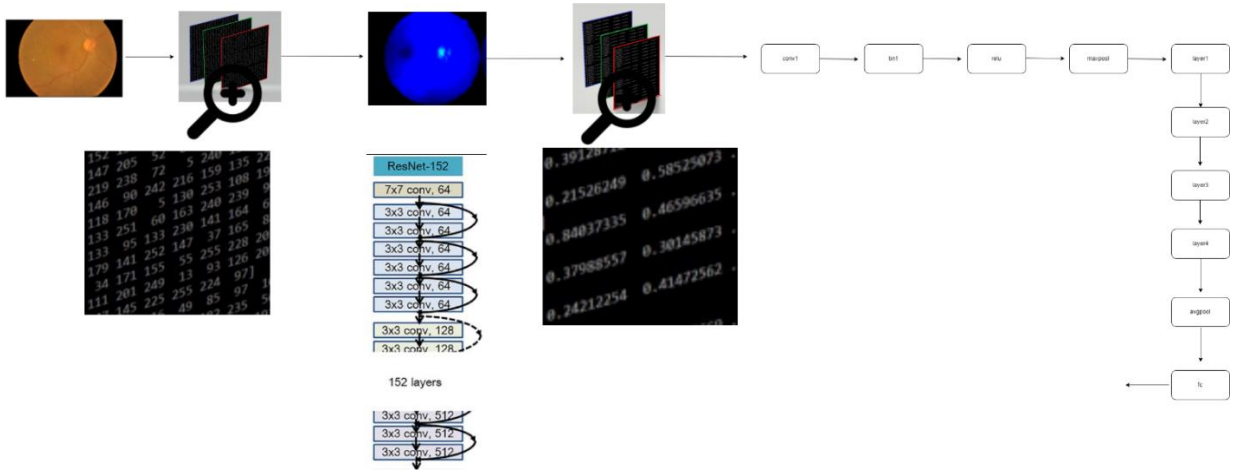
Feedback Loop: Establish a feedback loop with end-users, including healthcare professionals and patients, to gather feedback and incorporate suggestions for system enhancement.

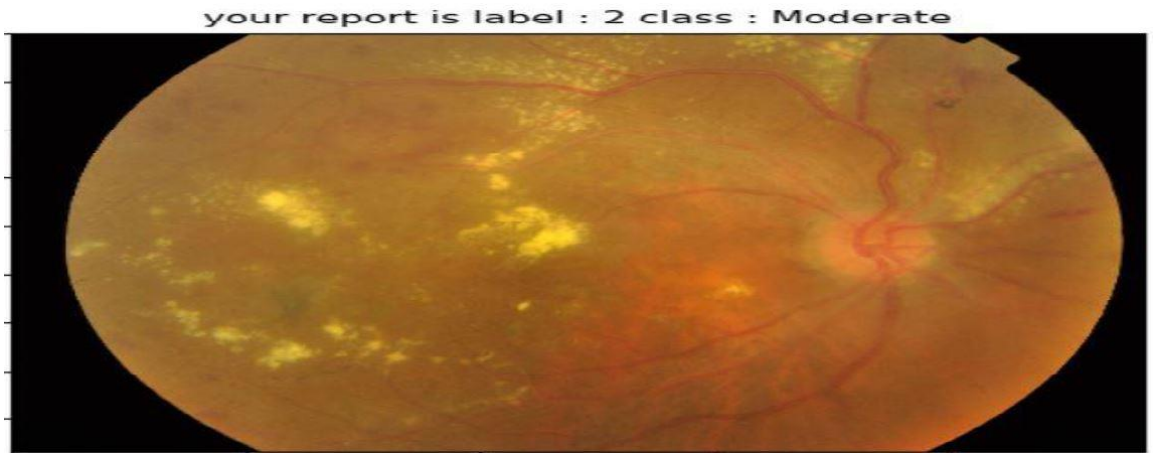
Research and Innovation: Stay informed about the latest advancements in deep learning, medical imaging, and diabetic retinopathy detection to drive ongoing research and innovation in the field

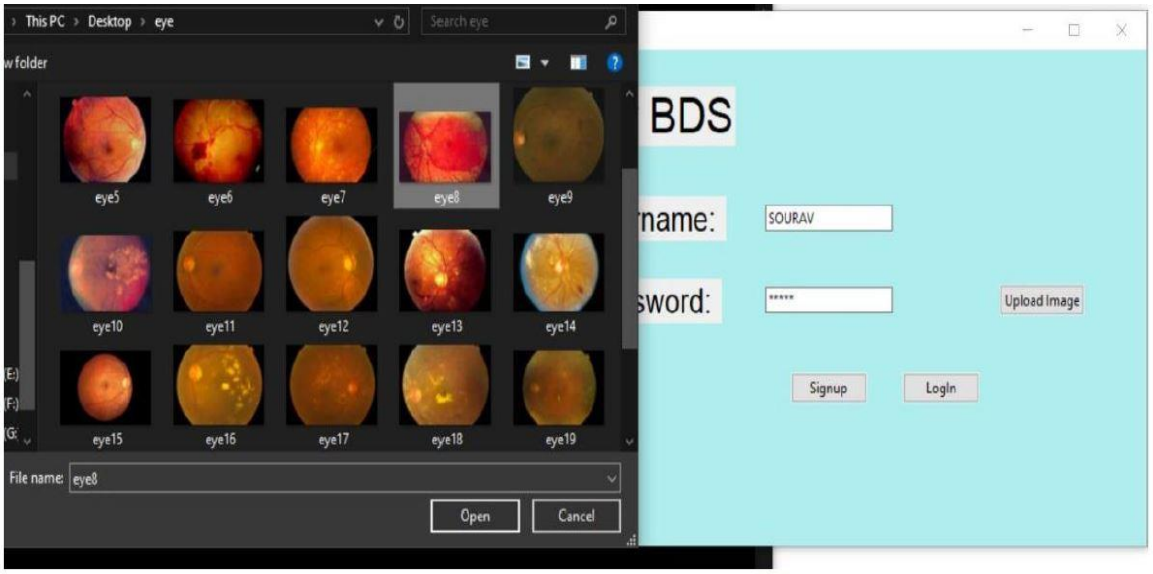
9. SCREEN LAYOUT






ResNet-152
7x7 conv, 64
3x3 conv, 64
3x3 conv, 64
3x3 conv, 64
3x3 conv, 64
3x3 conv, 64
3x3 conv, 64
3x3 conv, 128
152 layers
3x3 conv, 512
3x3 conv, 512
3x3 conv, 512
conv1
bn1
relu
maxpool
layer1
layer2
layer3
layer4
avgpool
fc


your report is label : 2 class : Moderate


This PC
Desktop
eye
Search eye
folder
eye5
eye6
eye7
eye8
eye9
eye10
eye11
eye12
eye13
eye14
eye15
eye16
eye17
eye18
eye19
File name:
eye8
Open
Cancel
BDS
name:
SOURAV
sword:
Upload Image
Signup
LogIn

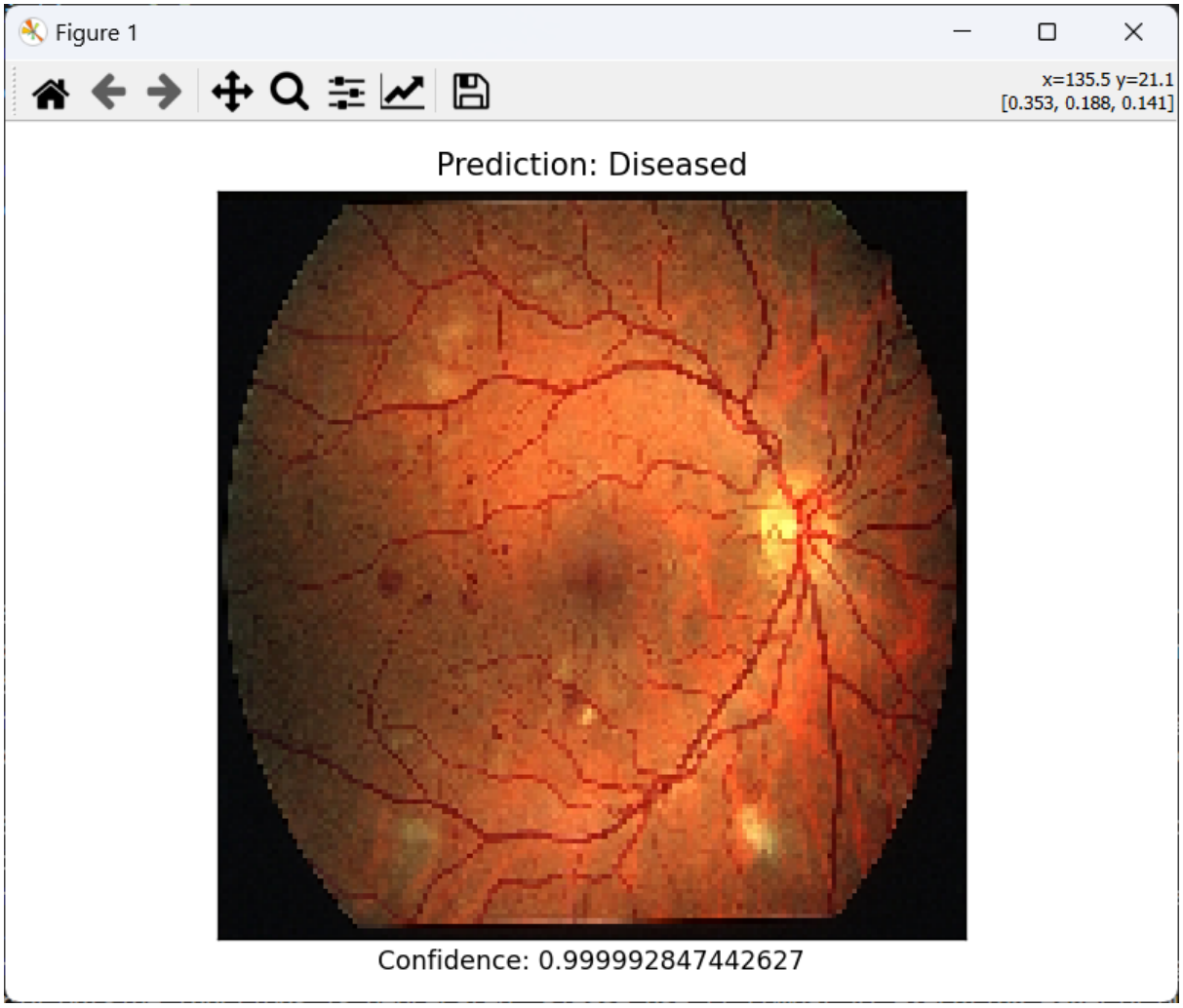


Figure 1
x=135.5 y=21.1
[0.353, 0.188, 0.141]
Prediction: Diseased
Confidence: 0.999992847442627

10.CONCLUSION

The imbalance between diseased and normal retinal images in the dataset can pose challenges for model training and evaluation, requiring techniques such as oversampling, under sampling, or class weighting to address. Deep learning models are often perceived as black boxes, making it challenging to interpret their decisions. Developing methods for explaining model predictions, such as attention mechanisms or saliency maps, can enhance trust and adoption by healthcare professionals.

Ensuring patient privacy and confidentiality throughout the data collection, annotation, and deployment processes, including obtaining informed consent and adhering to data protection regulations like HIPAA or GDPR. Conducting rigorous clinical validation studies to assess the real-world performance of the deployed model and its impact on patient outcomes, potentially involving collaboration with medical institutions and regulatory agencies.

BIBLIOGRAPHY

BOOK REFERENCE

[1] N. Abbas, T. Saba, D. Mohamad, A. Rehman, A. Almazyad, and J. Saleh Al-Ghamdi, "Machine aided malaria parasitemia detection in giemsastained thin blood smears," Neural Computing and Applications, vol. 29, no. 3, p. 803–818, 2016. [2]

A. Capone, I. Ricci, C. Damiani, M. Mosca, P. Rossi, P. Scuppa, E. Crotti, S. Epis, M. Angeletti, M. Valzano, L. Sacchi, C. Bandi, D. Daffonchio, M. Mandrioli, and G. Favia, "Interactions between asaia, plasmodium and anopheles: New insights into mosquito symbiosis and implications in malaria symbiotic control," Parasites & vectors, vol. 6, p. 182, 2013.

[3] I. Bates, V. Bekoe, and A. Asamoa-Adu, "Improving the accuracy of malaria-related laboratory tests in ghana," Malaria journal, 12 2004.

[4] F. T. Boray, G. D. Andrew, and K. Izzet, "Computer vision for microscopy diagnosisof malaria," Malaria Journal, pp. 1–14, 2009.

[5] S. Akanksha, S. Sini, and D. Shatendra, "Recent image enhancement techniques: A review," International Journal of Engineering and Advanced Technology (IJEAT), vol.4, no. 1, p. 40–45, 2014. [6] S. Jalari, E. B, K. Reddy, and C.-H. Lai, "An image processing approach for accurate determination of parasitemia in peripheral blood smear images," International Journal pf Computer Applications, vol. 1, pp. 23–28, 2011.

[7] O. Nobuyuki, "A threshold selection method from gray-level histograms," IEEE Transactions on Systems, Man, and Cybernetics, vol. 9, no. 1, pp. 62–66, 1979.

[8] S. N. Mark and S. A. Alberto, "Feature extraction and image processing," The ACM Digital Library, 2008.

[9] L. Zhaohui, P. Andrew, E. IIker, P. Mahdieh, S. Kamolrat, P. Kannappan, G. Peng, H. Md Amir, M. Richard, James, H. Jimmy, Xiangji, J. Stefan, and T. George, "Cnn-based image analysis for malaria diagnosis," in 2016 IEEE International Conference on Bioinformatics and Biomedicine (BIBM). IEEE, 2016.

[10] D. Yuhang, J. Zhuocheng, S. Hongda, and P. W. David, "Classification accuraciesof malaria infected cells using deep convolutional neural networks based on decompressed images," in SoutheastCon 2017. IEEE, 2017.

[11] H. Jane, R. Deepali, C. L. Stefanie, R. Gabriel, N. Francois, A. N. Odailton, M. Benoit, V. L. Marcus, U. F. Marcelo, R. Laurent, T. D. Manoj, T. C. Fabio, M. Matthias,and E. C. Anne, "Applying faster r-cnn for object detection on malaria images," in

2017 IEEE Conference on Computer Vision and Pattern Recognition Workshops(CVPRW). IEEE, 2017.

[12] M. Leila, A.-A. Karim, and B. Abdolamir, "Malaria parasite detection in giemsa-stained blood cell images," in 2013 8th Iranian Conference on Machine Vision and Image Processing (MVIP). IEEE, 2013.

[13] W. Sri and Wijiyanto, "Texture analysis to detect malaria tropica in blood smears image using support vector machine," International Journal of Innovative Research in Advanced Engineering (IJIRAE), vol. 1, no. 8, pp. 302–306, 2014.

WEB REFERENCE

- www://.w3schools.com/
- www://python.org
- www://docs.python.org
- www://devdocs10/python
- www://geeksforgeeks.org/python
- www://programiz.com/python