EARLY IDENTIFICATION OF DIABETIC RETINOPATHY THROUGH DEEP LEARNING-BASED FUNDUS IMAGE ANALYSIS

Prepared By,

S.Santhi

Assistant Professor

Department of IT,

National Engineering College, Kovilpatti

ABSTRACT

Diabetic Retinopathy is a disease caused by uncontrolled chronic diabetes and it can cause complete blindness if not timely treated. Therefore early medical diagnosis of diabetic retinopathy and it medical cure is essential to prevent the severe side effects of diabetic retinopathy. Manual detection of diabetic retinopathy by ophthalmologist take plenty of time and patients need to suffer a lot at this time. This project integrates the deep learning technique to generate a model that can help detect diabetic retinopathy quickly. By using Convolutional Neural Network (CNN), a deep learning algorithm that is well-suited for image recognition and processing tasks, the proposed deep learning model intends to autonomously investigate retinal images. This study focuses on employing deep learning methodologies to analyze fundus images for the timely identification of diabetic retinopathy at its nascent stages. The research aims to develop a robust system capable of accurately detecting subtle signs of retinal pathology through automated analysis of fundus images. Leveraging Convolutional Neural Networks and other deep learning techniques, this approach seeks to enhance the efficiency and accuracy of diabetic retinopathy diagnosis. The proposed system's performance will be evaluated against established benchmarks and clinical standards, aiming to contribute to the early identification and subsequent management of diabetic retinopathy, thereby potentially reducing vision-related complications in diabetic patients. By introducing automated systems capable of detecting diabetic retinopathy through image analysis can augment medical practices and leads to the development of more efficient, cost-effective, and accessible diagnostic tools for diabetic patient. The study aims to contribute to improved patient care by enabling early identification of diabetic retinopathy, leading to timely intervention and better management of the condition.

CHAPTER 1

INTRODUCTION

Diabetic Retinopathy (DR) is an eye disorder which is a consequence of Diabetes Mellitus (DM). DR causes inflammation and breach of retinal blood vessels for the formation of various irregular retinal lesions. It is frequently observed in patients having diabetes from a longer duration of time such as 10–15 years. Statistics have stated that 80% of diabetic patients, suffering from protracted diabetes suffer from diverse phases of DR. India has approximately 73 million individuals suffering from diabetes. The early consultation of the patient with diabetes, with experts for DR assessment and evaluation has become necessary. The process of early diagnosis can reduce the development of DR and susceptibility to severe blindness.

Diabetes mellitus is a common metabolic disease that is associated with a number of consequences, one of which is DR, which is a major risk factor for visual acuity. Due to the complex pathophysiology of DR, the retina experiences microvascular alterations that result in the creation of recognizable lesions. The present paradigm for diagnosis depends on subjective and variable manual evaluations performed by experienced ophthalmologists, which can delay the timely identification of DR. The shortcomings of the existing diagnosis strategy highlight how urgent it is to look into novel solutions as the prevalence of diabetes rises worldwide.

1.1 Diagnosis of Diabetic Retinopathy:

The diagnosis of DR with the traditional approach mostly depends on the skill of ophthalmologists who manually assess retinal fundus pictures to determine whether or not DR is present and how severe it is. Although there is no doubting the expertise of these medical professionals, the procedure is time-consuming, resource-intensive, and susceptible to inter-observer variability. There is a huge gap in the supply of qualified ophthalmologists, which causes delays in diagnosis and treatment.

1.2 Early detection of Diabetic Retinopathy:

Early detection of DR may be delayed due to the time and skill needed for manual examinations. The ailment may present with mild symptoms at first, but if treatment is delayed, it can worsen and reach later stages with fewer alternatives. Therefore, there's a chance that the current

diagnostic strategy will overlook important chances for early diagnosis and management, which could have an effect on patient outcomes.

1.3 Deep Learning in Medical Imaging:

Deep Learning has proven very effective in image processing applications, especially when using Convolutional Neural Networks (CNNs). It is a perfect fit for the detailed analysis needed in medical imaging because of its capacity to identify intricate patterns and features within big datasets. Deep learning has the potential to improve diagnostic precision and facilitate prompt intervention in the context of depression and anxiety (DR), when early indicators may be imperceptible. Deep learning can overcome the drawbacks of manual diagnosis by automating the processing of fundus images, offering a standardized and scalable method.

A subset of Artificial Intelligence (AI) called Deep Learning (DL) has become a game-changing paradigm in a number of industries, medical imaging being one of the most promising. The use of Deep Learning (DL) in medical imaging, such as the examination of retinal fundus pictures for diseases such as Diabetic Retinopathy (DR), has enormous potential to revolutionize current diagnostic methods.

The intelligent systems using DL are proposed as an early and potentially scalable alternative for DR detection. Conventional ML models and data analysis approaches are shallow in nature, and have shown poor performance in learning and training complex non-linear features from larger datasets and hence, are unable to exhibit better analysis and interpretation. Besides, DL based CNN models have overshadowed ML models in performance, through inbuilt preprocessing, convolutional operations, better learning and generalization with deeper networks, less overfitting, data imbalance mitigation, optimization etc. Therefore, various state-of-the-art ML-based models called DL models are proposed for deep feature extraction and image classification tasks.

1.4 Objective and Scope of the Project:

This project's main goal is to create a Deep Learning Fundus Image Analysis system that can recognize early symptoms of DR on its own. By utilizing a broad range of datasets that span different phases of the illness, the deep learning model seeks to identify complex patterns characteristic of drug resistance (DR), providing a more uniform and impartial method of diagnosis. This work aims to solve important problems like resource intensity and inter-observer variability related to manual assessments, in addition to pushing the boundaries of technology.

1.5 Impact and Significance of the Project:

This initiative is expected to have a large-scale impact. With timely therapies that can dramatically reduce the risk of visual loss, early identification by deep learning analysis has the potential to change the management of DR. Furthermore, integrating a strong deep learning model into clinical settings could simplify diagnostic procedures, increase productivity, and broaden access to DR screening—particularly in areas where there is a scarcity of eye care specialists.

CHAPTER 2

LITERATURE REVIEW

This chapter discusses about the various methodologies used to perform the image analysis of early detection of diabetic retinopathy.

2.1 EfficientNetB3

EfficientNetB3 is a convolutional neural network architecture that belongs to the EfficientNet family, which was proposed by Google Brain researchers in 2019. EfficientNet models are designed to achieve state-of-the-art performance while being computationally efficient, meaning they strive to maximize accuracy while minimizing the number of parameters and computational resources required.

EfficientNetB3 specifically is one of the variants in the EfficientNet series, characterized by its depth, width, and resolution. It is larger and more complex than EfficientNetB0 and EfficientNetB1 but smaller than EfficientNetB4, B5, and B7.

The 'B3' in EfficientNetB3 refers to its compound scaling parameters:

- Depth: The number of layers in the neural network.
- Width: The width of the network, which typically refers to the number of channels (or filters) in convolutional layers.
- Resolution: The input image resolution.

EfficientNetB3 strikes a balance between model size and performance, making it suitable for various computer vision tasks like image classification, object detection, and segmentation.

These models have been pre-trained on large-scale datasets like ImageNet and have demonstrated strong transfer learning capabilities.

2.2 Inception V3:

InceptionV3 is a convolutional neural network architecture developed by researchers at Google. It's a part of the Inception family of models, known for their deep and complex architecture designed to improve the performance of computer vision tasks while maintaining computational efficiency.

The Inception architecture employs a module called the "Inception module," which uses multiple convolutional filters of different sizes (1x1, 3x3, 5x5) and pooling operations in parallel to capture features at various scales. These parallel operations are then concatenated or merged, allowing the network to learn both local and global features efficiently. InceptionV3 introduced

factorization of convolutions into smaller convolutions, such as using two consecutive 3x3 convolutions instead of a single 5x5 convolution. This strategy reduces the number of parameters while maintaining representational capacity, thereby improving computational efficiency. InceptionV3 incorporates auxiliary classifiers at intermediate layers during training, aiming to mitigate the vanishing gradient problem and provide additional regularization, ultimately aiding in training deeper networks. InceptionV3 utilizes batch normalization and rectified linear unit (ReLU) activations to accelerate training convergence and mitigate issues like vanishing gradients.

InceptionV3 has been pre-trained on large-scale image classification tasks, such as the ImageNet dataset, demonstrating competitive performance. The model's architecture balances computational efficiency with accuracy, making it suitable for various computer vision applications, including image classification, object detection, and image segmentation. Moreover, due to its efficiency and effectiveness, InceptionV3 has served as a base model for transfer learning, where practitioners fine-tune the pre-trained model on specific datasets for tasks where labeled data might be limited, achieving impressive results with less computational cost compared to larger architectures.

2.3 VGG-19:

VGG-19 is a convolutional neural network architecture proposed by the Visual Graphics Group (VGG). It is part of the VGG family of models, which are known for their simplicity and uniform architecture. VGG-19 specifically refers to a deep convolutional neural network consisting of 19 layers (including 16 convolutional layers and 3 fully connected layers).

VGG-19 comprises multiple layers of 3x3 convolutional filters stacked on top of each other. These convolutional layers, along with max-pooling layers, systematically decrease spatial dimensions while increasing depth, allowing the network to learn hierarchical features from the input image. The VGG models are characterized by their simplicity, where the convolutional layers are stacked one after the other, maintaining a consistent filter size of 3x3 and using max-pooling to downsample feature maps. After several convolutional layers, VGG-19 ends with three fully connected layers followed by a softmax layer for classification in the case of image classification tasks. These fully connected layers combine high-level features learned by earlier layers to make predictions.

VGG-19 maintains a consistent architecture throughout the network, making it easier to understand and implement. However, this also results in a large number of parameters, making it computationally expensive compared to more modern architectures. VGG-19 and its predecessors, such as VGG-16 and other variants, have been influential in the development of convolutional neural networks (CNNs) for computer vision tasks. They were initially introduced for the ImageNet Large Scale Visual Recognition Challenge (ILSVRC) and achieved competitive results in image classification. Despite its simplicity and ease of implementation, VGG-19's main drawback is its large number of parameters, which can lead to increased memory usage and computational cost compared to more recent architectures designed to achieve similar or better performance with fewer parameters, such as the ResNet, Inception, or EfficientNet models.

CHAPTER 3

SYSTEM DESIGN

The main focus of this chapter is to discuss about the proposed methodology for deep learning-based diabetic retinopathy detection.

3.1 Proposed System

The architecture of the proposed system is shown in Figure 3.1. It consists of

- Data Collection
- Image Preprocessing
- Training and Testing
- Model building
- Performance analysis
- Prediction using GUI

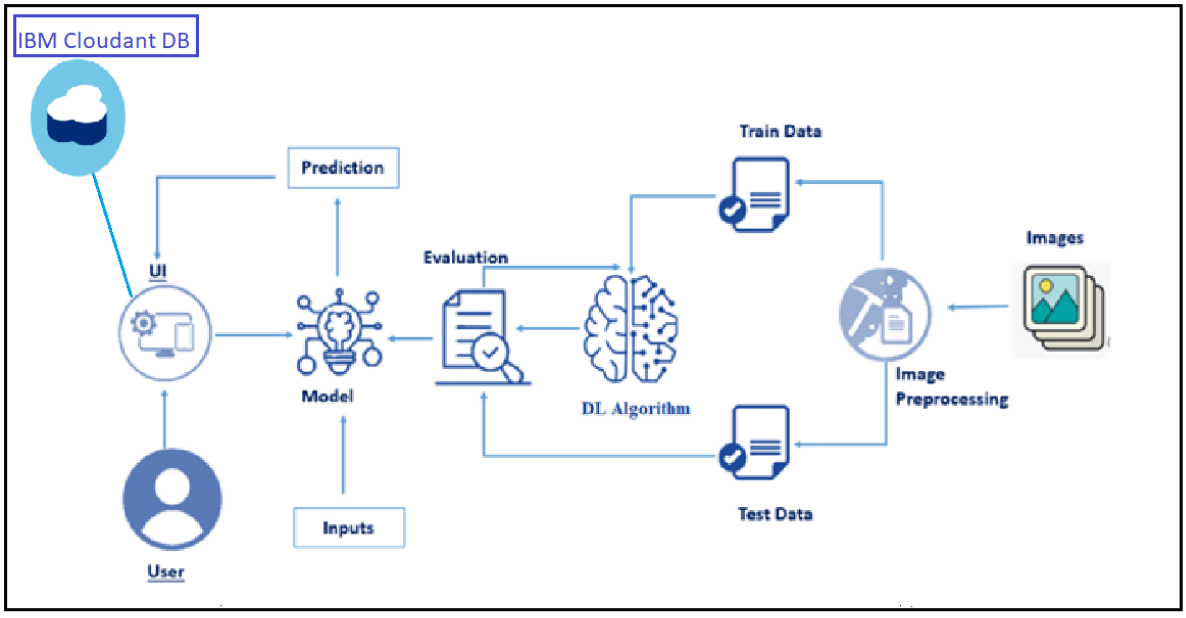

Fig 3.1 Architecture of proposed system

To prepare the training and testing portions, data was gathered from the dataset and processed. With SoftMax as the activation function, the model is built using Xception Net. Next, the Python Flask application's testing photos were used to make the prediction. The accuracy and loss metrics are used to compare the model's performance.

3.2 Data collection

Building machine learning models requires gathering data, especially for initiatives like the diagnosis of diabetic retinopathy. The purpose of data collection in this project is to compile a dataset of retinal fundus images, which are necessary for the deep learning model's training and testing.

3.3 Image Preprocessing

In order to prepare raw retinal fundus images for deep learning model training, image preprocessing is an essential step. The model performs better when proper preprocessing is used since it can better learn pertinent features and patterns. When it comes to detecting diabetic retinopathy, the following picture preparation methods are frequently used.

Adjust the picture sizes to a standard resolution. Standardizing the image dimensions helps with efficient processing during model training and guarantees consistency throughout the dataset. Set the values of the pixels to a standard scale, usually between 0 and 1. Normalization guarantees that the model is not sensitive to changes in pixel intensity and aids in stabilizing the training process. Images should be cropped to highlight pertinent areas of interest, like the macula and optic disc. In addition to ensuring that the model concentrates on important regions for the diagnosis of diabetic retinopathy, this lowers computational complexity.

3.4 Training and Testing

In order to create a machine learning model for the diagnosis of diabetic retinopathy, training and testing are essential stages. In these stages, the model is trained on a labeled dataset, and its performance is assessed on an additional set of data. Separate the dataset into a testing set and a training set. The testing set assesses the model's performance on unobserved data, while the training set is used to train the model.

3.5 Model building

Selecting the pre-trained Xception model, adding more layers, and setting the learning process are the stages involved in developing a model for diabetic retinopathy diagnosis using the Xception architecture. The Keras library, a high-level deep learning API that operates on top of TensorFlow is used.

A deep Convolutional Neural Network (CNN) architecture with impressive results in image-related tasks is the Xception model. Its ability to capture complex hierarchical information is attributed to its great depth and the usage of depthwise separable convolutions.

- **Loading the Pre-trained Xception Model**

  Keras's **Xception** class is used to load the Xception model. The **include_top=False** argument ensures that the top (fully connected) layers of the original Xception model, which are designed for ImageNet classification, are excluded. This permits us to add custom layers suitable for the diabetic retinopathy detection task.

- **Freezing the Pre-trained Layers**

  The layers of the pre-trained Xception model are frozen by iterating through them and setting **layer.trainable = False**. Freezing prevents the weights of these layers from being updated during the initial training, preserving the knowledge learned from ImageNet.

- **Custom Model Architecture**

  To build a customized architecture on top of the pre-trained Xception basis, a new Keras **Sequential model** is developed. To do this, more layers must be added in order to modify the model for the particular purpose of detecting diabetic retinopathy.

- **Global Average Pooling Layer**

  After the Xception base, a Global Average Pooling 2D layer is added. By calculating the average value of every feature map, this layer shrinks the spatial dimensions of the feature maps. It assists in lowering the number of parameters and extracting the crucial data from the acquired features.

- **Dense Layer with Softmax Activation**

  In the output layer of a neural network, the softmax activation function is frequently utilized for multi-class classification issues. It converts the network's raw output scores, or logits, into probabilities. By normalizing the logits, the softmax function makes sure that the probabilities add up to one for every class. This allows the model to produce probabilistic forecasts, with the final prediction being made for the class with the highest probability.

3.6 Performance analysis

A measure of a model's total forecast correctness is its accuracy. It is computed as the proportion of accurately anticipated cases to all instances.

$$Accuracy = \frac{Number\ of\ Correct\ Predictions}{Total\ Number\ of\ Predictions} \qquad (1)$$

The prediction error of the model is expressed as a loss. It measures the degree to which the actual values and expected values agree. Reducing the loss is the main objective of training. Classificative cross entropy is frequently employed in classification problems. One possible tool for regression problems is mean squared error (MSE).Better alignment between the true and forecast values is shown by lower loss values. Understanding the model's convergence is aided by keeping an eye on the loss throughout training. It is possible that the model is learning from the data if the loss decreases across epochs.

3.7 Prediction using GUI

After the model building, testing images has undergone the following steps to identify the level of diabetic retinopathy.

- Load Testing Images
- Preprocess Testing Images
- Model Prediction

Make that the preprocessing done on the testing photos matches the preprocessing done on the training images. Utilize the relevant metrics (accuracy, precision, recall, F1 score, etc.) to assess the model's performance in relation to the task. When evaluating and using the model predictions, bear in mind any particular needs or limitations associated with your application or project. To obtain the projected class index for a multi-class classification problem, think about utilizing the argmax function.

CHAPTER 4

IMPLEMENTATION

This chapter focuses on the software requirements and python libraries required for the implementation of the proposed methodology. The proposed methodology is implemented based on the following,

- Dataset Downloading
- Model Building
- Cloudant DB
- Application Building

The dataset was obtained from Kaggle and preprocessed, with training and testing photos being kept apart. Model constructed with Softmax and Xception. Web applications were built using the Flask Python application, which was designed by CloudantDB.

4.1 Dataset

The training and testing data must be divided into two distinct folders in order to construct a DL model. However, the project dataset folder contains the training and testing files. Thus, in this instance, you must declare a variable and feed it the path to the folder. In this model, four distinct transfer learning models are employed, with the best model, Xception being chosen. For the Xception model, the image input size is 299, 299.

```
imageSize = [299, 299]

trainPath = r"/content/preprocessed dataset/preprocessed dataset/training"

testPath = r"/content/preprocessed dataset/preprocessed dataset/testing"
```

Import the necessary libraries as shown.

```
from tensorflow.keras.layers import Dense, Flatten, Input
from tensorflow.keras.models import Model
from tensorflow.keras.preprocessing import image
from tensorflow.keras.preprocessing.image import ImageDataGenerator, load_img
from tensorflow.keras.applications.xception import Xception, preprocess_input
from glob import glob
import numpy as np
import matplotlib.pyplot as plt
```

ImageDataGenerator class is instantiated and the configuration for the types of data augmentation

There are five main types of data augmentation techniques for image data; specifically:

- Image shifts via the width_shift_range and height_shift_range arguments.
- The image flips via the horizontal_flip and vertical_flip arguments.
- Image rotations via the rotation_range argument
- Image brightness via the brightness_range argument.
- Images zoom via the zoom_range argument.

An instance of the ImageDataGenerator class can be constructed for train and test.

```
train_datagen = ImageDataGenerator(rescale = 1./255,
                                   shear_range = 0.2,
                                   zoom_range = 0.2,
                                   horizontal_flip = True)

test_datagen = ImageDataGenerator(rescale = 1./255)
```

To apply ImageDataGenerator functionality to the Train set and Test set by using the following code. For Training set using flow_from_directory function.

This function will return batches of images from the subdirectories

Arguments:

- Directory: Directory where the data is located. If labels are "inferred", it should contain subdirectories, each containing images for a class. Otherwise, the directory structure is ignored.
- batch_size: Size of the batches of data which is 64.
- target_size: Size to resize images after they are read from disk.
- class_mode:

```
training_set = train_datagen.flow_from_directory('/content/preprocessed dataset/preprocessed dataset/training',
                                                 target_size = (299, 299),
                                                 batch_size = 32,
                                                 class_mode = 'categorical')

test_set = test_datagen.flow_from_directory('/content/preprocessed dataset/preprocessed dataset/testing',
                                            target_size = (299, 299),
                                            batch_size = 32,
                                            class_mode = 'categorical')
```

```
Found 3662 images belonging to 5 classes.
Found 734 images belonging to 5 classes.
```

4.2 Model Building

To use one of the models as a basic feature extractor, we frozen each of the five convolution blocks to prevent the weights from changing as we train our own model. We have taken into consideration the following images: (229, 229, and 3). Also, include_top = False was supplied since we want to train a fully connected layer for our picture classification (because the convolution layer is not included in the ImageNet dataset) and we are using it for feature extraction.

Flatten layer flattens the input and it does not affect the batch size

```
xception = Xception(input_shape=imageSize + [3], weights='imagenet',include_top=False)
```

```
Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/xcep
_kernels_notop.h5
83689472/83683744 [==============================] - 1s 0us/step
83697664/83683744 [==============================] - 1s 0us/step
```

```
# don't train existing weights
for layer in xception.layers:
    layer.trainable = False
```

```
# our layers - you can add more if you want
x = Flatten()(xception.output)
```

A neural network layer with deep connections is called a dense layer. It is the layer that is most often utilized and common. Create a model object named model with inputs as xception.input and output as dense layer.

```
prediction = Dense(5, activation='softmax')(x)

# create a model object
model = Model(inputs=xception.input, outputs=prediction)
```

The number of classes in the training set and the number of neurons in the Dense layer are equal. The final Dense layer neurons translate their outputs into corresponding probabilities using softmax activation. Comprehending the model is a crucial step in using it appropriately for training and forecasting. Keras offers a straightforward summary technique for obtaining comprehensive information about the model and its layers.

```
# view the structure of the model
model.summary()
```

```
Model: "model"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_1 (InputLayer)        [(None, 299, 299, 3       0
                             )]

 block1_conv1 (Conv2D)       (None, 149, 149, 32       864
                             )

 block1_conv1_bn (BatchNormaliz  (None, 149, 149, 32       128
 ation)                      )

 block1_conv1_act (Activation)  (None, 149, 149, 32       0
                             )

 block1_conv2 (Conv2D)       (None, 147, 147, 64       18432
                             )

 block1_conv2_bn (BatchNormaliz  (None, 147, 147, 64       256
 ation)                      )

 block1_conv2_act (Activation)  (None, 147, 147, 64       0
                             )

 block2_sepconv1 (SeparableConv  (None, 147, 147, 12       8768
 2D)                         8)
```

The compilation is the final step in creating a model. Once the compilation is done, we can move on to the training phase. The loss function is used to find errors or deviations in the learning process. Keras requires a loss function during the model compilation process.Optimization is an important process that optimizes the input weights by comparing the

prediction and the loss function. Here by using adam optimizerMetrics are used to evaluate the performance of your model. It is similar to the loss function, but not used in the training process.

```
# tell the model what cost and optimization method to use
model.compile(
  loss='categorical_crossentropy',
  optimizer='adam',
  metrics=['accuracy']
)
```

Now use our image dataset to train the model. The model is trained for thirty epochs. If the model has experienced the fewest losses to that point, the current state of the model is saved at the end of each epoch. It is evident that the training loss diminishes with each epoch up to ten, suggesting that there may be further room for model improvement.

fit_generator functions used to train a deep learning neural network

```
# fit the model
r = model.fit_generator(
  training_set,
  validation_data=test_set,
  epochs=30,
  steps_per_epoch=len(training_set)//32,
  validation_steps=len(test_set)//32
)
```

The model is saved with .h5 extension as follows An H5 file is a data file saved in the Hierarchical Data Format (HDF). It contains multidimensional arrays of scientific data.

```
model.save('Updated-Xception-diabetic-retinopathy.h5')
```

4.3 Cloudant DB

- Log in to IBM Cloud account, and click on Catalog

- To create the connection information that your application needs to connect to the instance, click New credential.
- Enter a name for the new credential in the Add new credential window.
- Accept the Manager role.
- (Optional) Create a service ID or have one automatically generated for you.
- (Optional) Add inline configuration parameters. This parameter isn't used by IBM Cloudant service credentials, so ignore it.
- Click Add.
- To see the credentials that are required to access the service, click the chevron.
- In order to manage a connection from a local system you must first initialize the connection by constructing a Cloudant client.We need to import the cloudant library.
- IBM Cloud Identity & Access Management enables you to securely authenticate users and control access to all cloud resources consistently in the IBM Bluemix Cloud Platform.

4.4 Application Building

In python flask HTML files, CSS and required images are placed in template and static folder. The main application is created by the following steps,

- Import Libraries
- Create Database
- Render HTML pages
- Configure the separate pages
- Predcition on UI

```
import numpy as np
import os
from tensorflow.keras.models import load_model
from tensorflow.keras.preprocessing import image
from tensorflow.keras.applications.inception_v3 import preprocess_input
import requests
from flask import Flask, request, render_template, redirect, url_for
from cloudant.client import Cloudant
```

Load the saved model. Importing the flask module in the project is mandatory. An object of Flask class is our WSGI application. Flask constructor takes the name of the current module (__name__) as argument.

```
model = load_model(r"Updated-Xception-diabetic-retinopathy.h5")

app = Flask(__name__)
```

Create a database using an initiated client.

```
from cloudant.client import Cloudant

# Authenticate using an IAM API key
client = Cloudant.iam('username','apikey', connect=True)

# Create a database using an initialized client
my_database = client.create_database('my_database')
```

Render HTML page

```
# default home page or route
@app.route('/')
def index():
    return render_template('index.html')


@app.route('/index.html')
def home():
    return render_template("index.html")


# registration page
@app.route('/register')
def register():
    return render_template('register.html')
```

Based on user input into the registration form stored it on data dictionary then the data using id parameter is validated with user input that can store it on query variable then can validate by

passing the query variable into the my_database.get_user_result() method. Check the docs length by using len(docs.all()) function. If the length of docs is 0 then user will register successfully on the platform and user data will store on the database. Otherwise its shows the message as user already registered please login and use web application for DR prediction. Based on user input into the login form we stored user id and password into the (user,passw) variables. Then validate the credentials using _id parameter with user input that store it on query variable then we can validate by passing the query variable into the my_database.get_user_result() method. Then we can check the docs length by using len(docs.all()) function. Validate the data that is stored on the database and check the username & password. If it's matched then the user will be able to login and use web application for DR prediction.

```
#login page
@app.route('/login')
def login():
    return render_template('login.html')

@app.route('/afterlogin',methods=['POST'])
def afterlogin():
    user = request.form['_id']
    passw = request.form['psw']
    print(user,passw)

    query = {'_id': {'$eq': user}}

    docs = my_database.get_query_result(query)
    print(docs)

    print(len(docs.all()))


    if(len(docs.all())==0):
        return render_template('login.html', pred="The username is not found.")
    else:
        if((user==docs[0][0]['_id'] and passw==docs[0][0]['psw'])):
            return redirect(url_for('prediction'))
        else:
            print('Invalid User')
```

The image is selected from uploads folder. Image is loaded and resized with load_img() method. To convert image to an array, img_to_array() method is used and dimensions are increased with expand_dims() method. Input is processed for xception model and predict() method is used to predict the probability of classes. To find the max probability np.argmax is used.

```python
@app.route('/result',methods=["GET","POST"])
def res():
    if request.method=="POST":
        f=request.files['image']
        basepath=os.path.dirname(__file__) #getting the current path i.e where app.py is present
        #print("current path",basepath)
        filepath=os.path.join(basepath,'uploads',f.filename) #from anywhere in the system we can give image but we want that i
        #print("upload folder is",filepath)
        f.save(filepath)

        img=image.load_img(filepath,target_size=(299,299))
        x=image.img_to_array(img)#img to array
        x=np.expand_dims(x,axis=0)#used for adding one more dimension
        #print(x)
        img_data=preprocess_input(x)
        prediction=np.argmax(model.predict(img_data), axis=1)

        #prediction=model.predict(x)#instead of predict_classes(x) we can use predict(X) ---->predict_classes(x) gave error
        #print("prediction is ",prediction)
        index=['No Diabetic Retinopathy', 'Mild DR', 'Moderate DR', 'Severe DR', 'Proliferative DR']
        #result = str(index[output[0]])
        result=str(index[_prediction[0]])
        print(result)
        return render_template('prediction.html',prediction=result)
```

CHAPTER 5

RESULTS AND DISCUSSION

This chapter discusses about the results of the project with prediction by Xception Net using Python Flask application.

5.1 Web pages

The application consists of following html pages,

- Index page
- Login page
- Register page
- Prediction page
- Logout page

Index page is shown in figure 5.1 and figure 5.2.

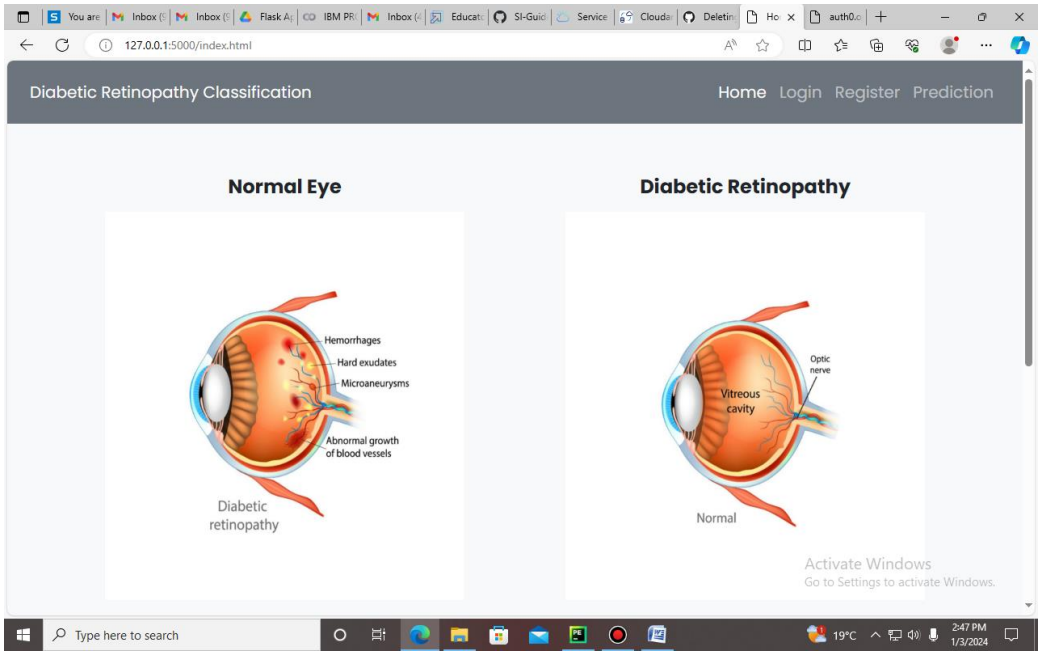

Fig 5.1 Index Page

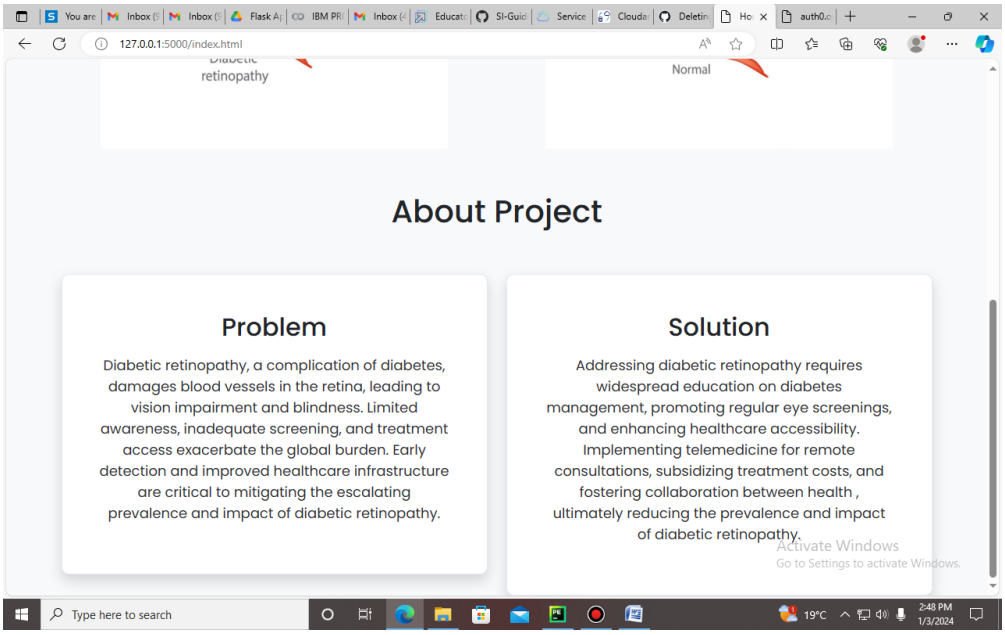

Fig 5.1 Index Page

Layout of the Register page is shown in figure 5.3.

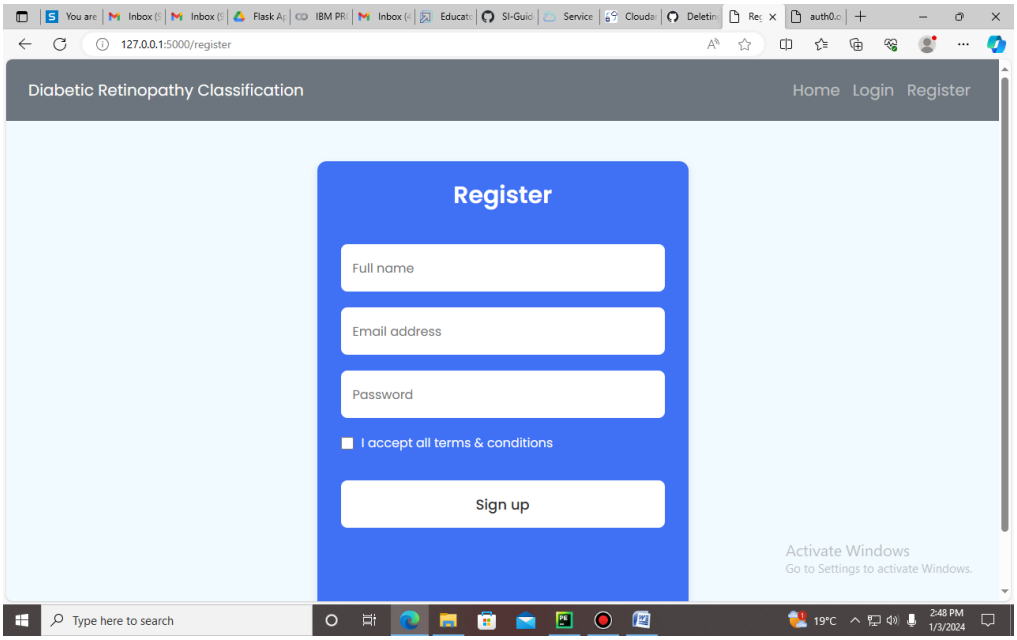

Fig 5.3 Register Page

Layout of the Login page is shown in figure 5.4.

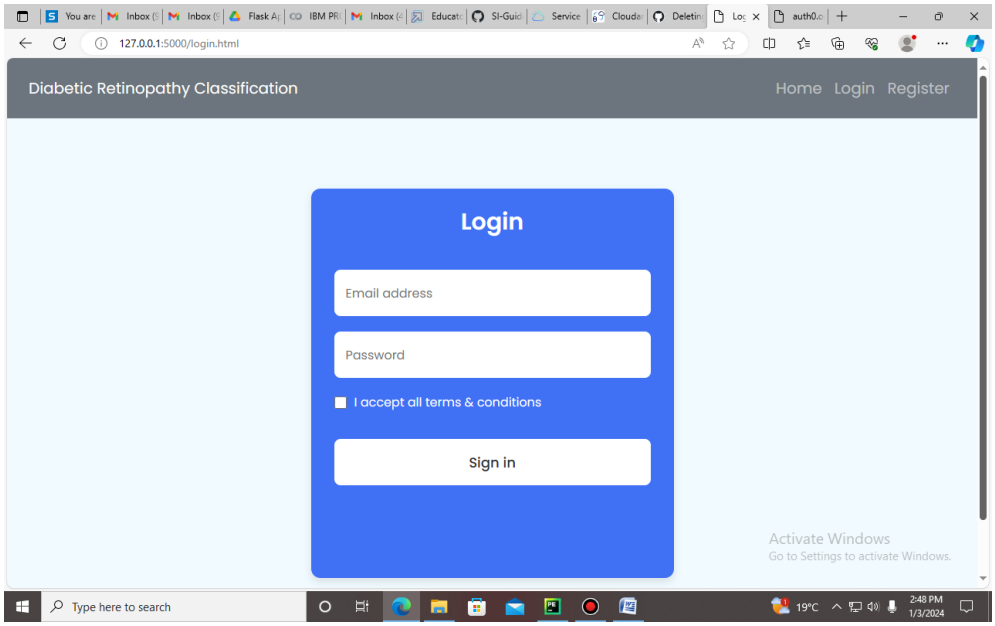

Fig 5.4 Login Page

After registration, the details are stored in CloudantDB. Connection is shown in figure 5.5.

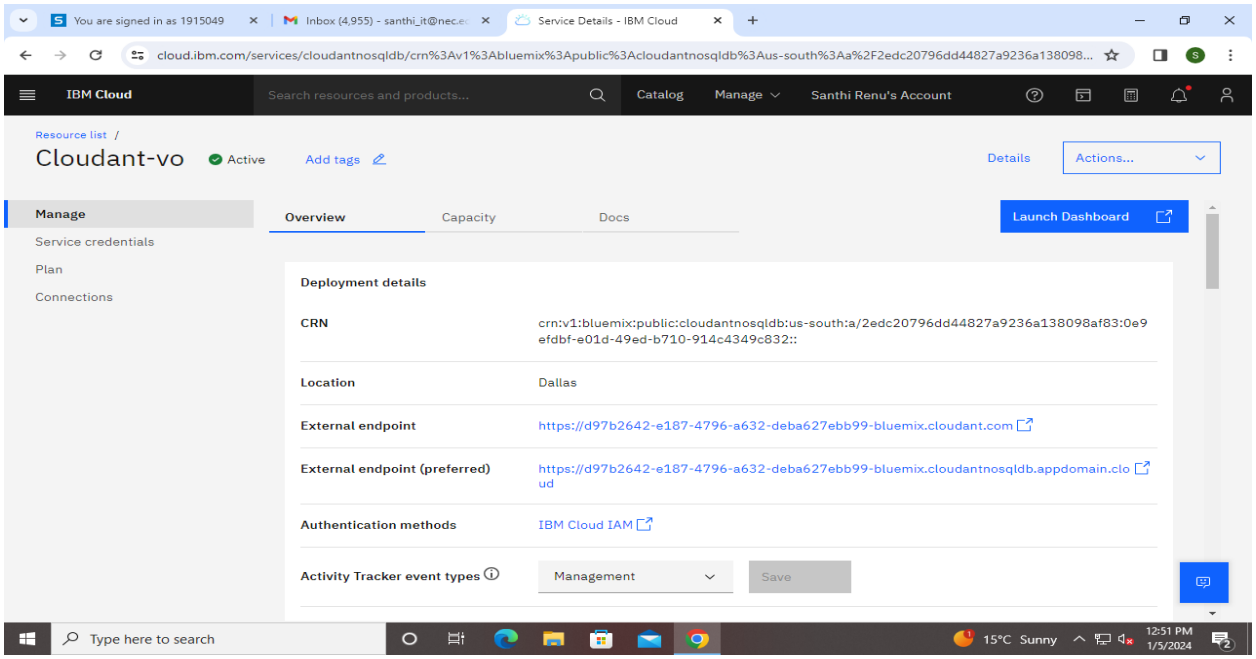

Fig 5.5 Cloudant DB creation

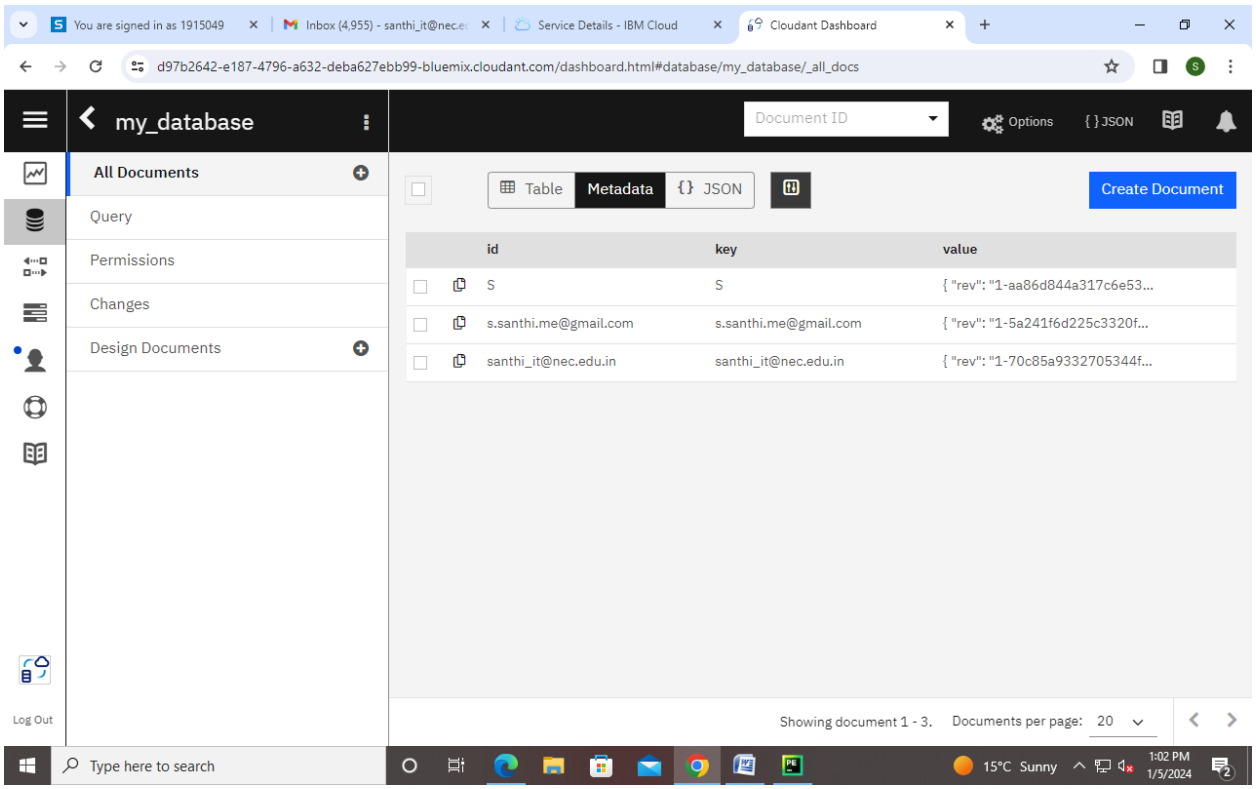

Fig 5.6 Insertion of details from registration page

After login prediction done in page shown in figure 5.7.

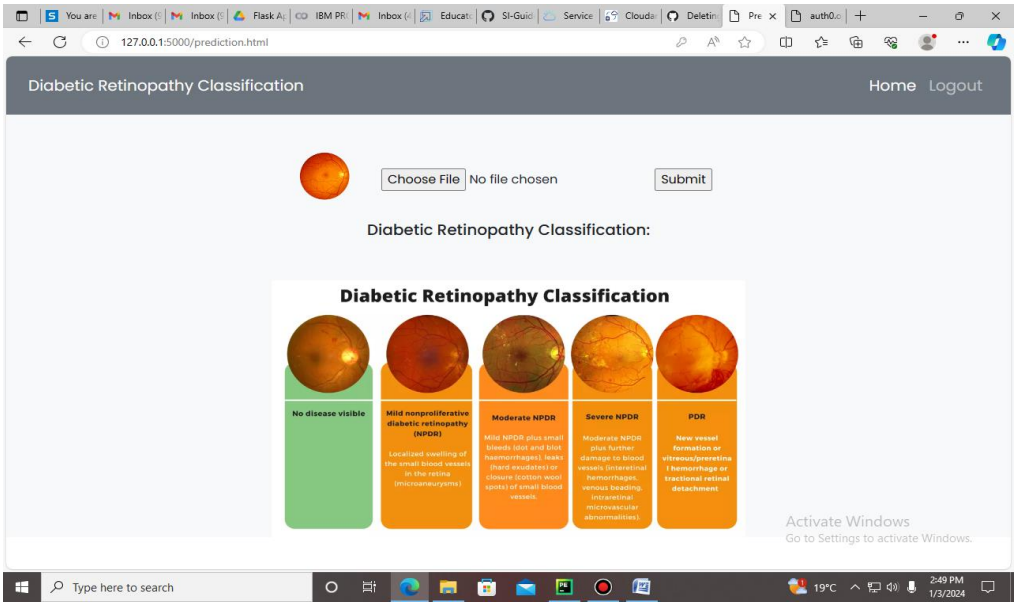

Fig 5.7 Prediction page layout

Testing image is loaded in the choose file option, and then the predicted class is shown below. A different kind of prediction is shown in figure 5.8, 5.9 and 5.10.

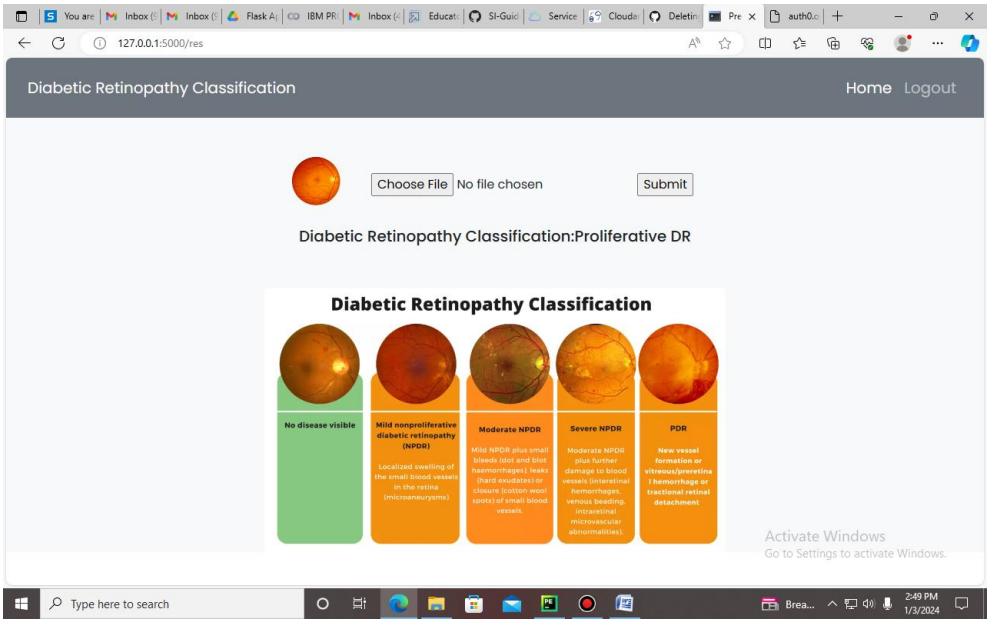

Fig 5.8 Classification of Proiferative DR

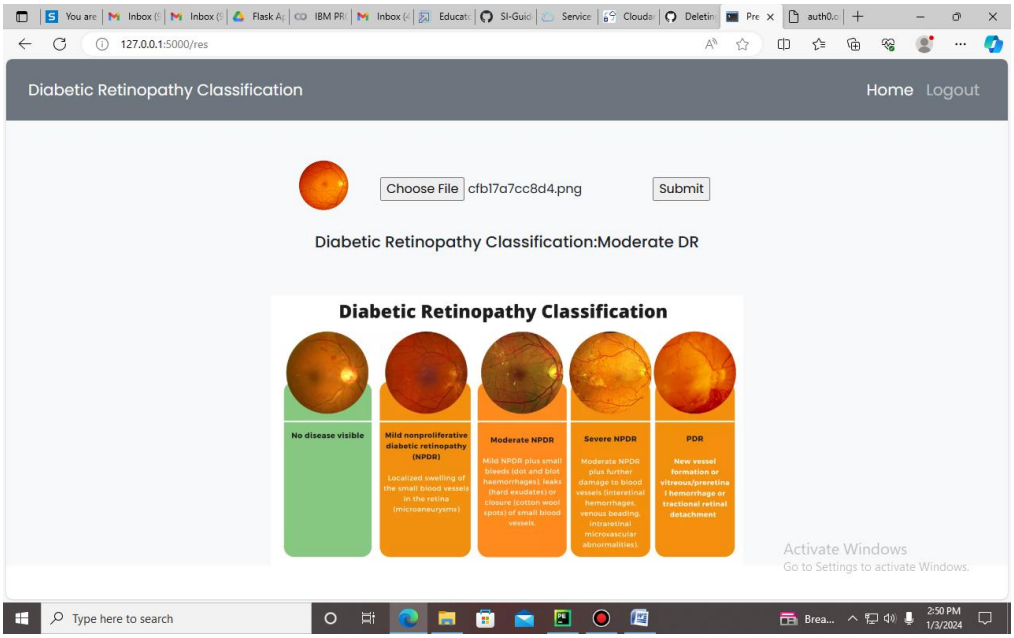

Fig 5.9 Classification of Moderate DR

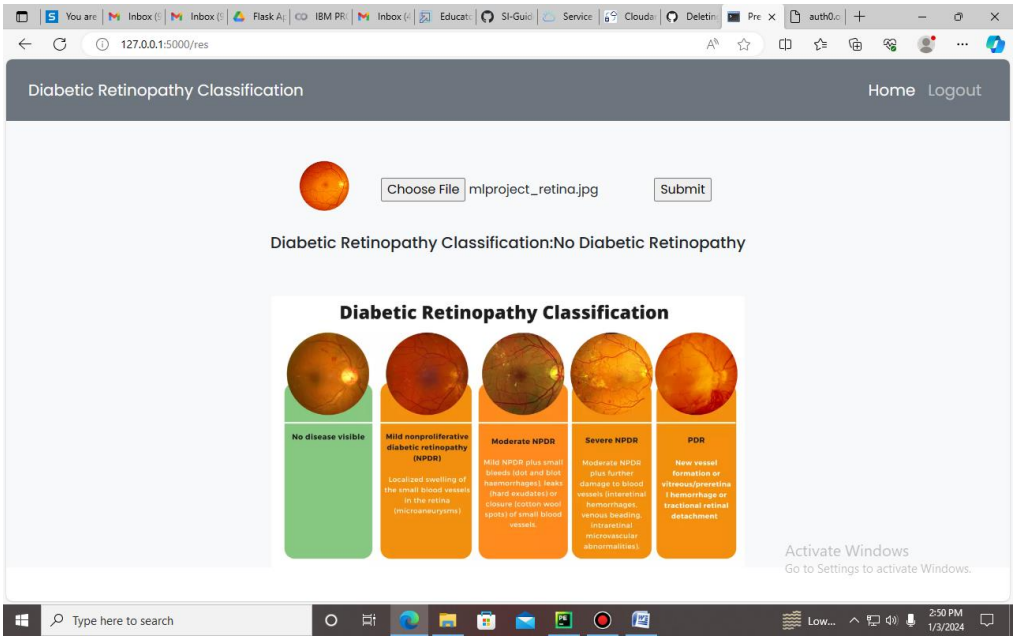

Fig 5.10 Classification of No Diabetic Retinopathy

CHAPTER 6

CONCLUSION

Early Identification of Diabetic Retinopathy through Deep Learning-based Fundus Image Analysis offers a viable solution to the problems with manual diagnosis and the urgent need for diabetes individuals to be detected at an early stage. In order to create an automated and effective solution for diabetic retinopathy detection, the project makes use of cutting-edge deep learning techniques, with a particular focus on the processing of retinal fundus images. The deep learning model that was constructed demonstrates the ability to automatically assess images of the retinal fundus and identify indicators of diabetic retinopathy. This can result in assessments and interventions being completed more quickly by greatly reducing the need for manual diagnosis. It is imperative to identify diabetic retinopathy early in order to stop permanent damage and vision loss. Healthcare personnel are empowered to undertake timely interventions and management measures due to the model's capacity to spot early indications. Time and resource efficiency are gains provided by the automated system.

REFERENCES

1 Dolly Das, Saroj Kumar Biswas, and Sivaji Bandyopadhyay, “Detection of Diabetic Retinopathy using Convolutional Neural Networks for Feature Extraction and Classification (DRFEC)”, Multimed Tools Appl. 2022 Nov 29 : 1–59.

2 Revathy, Nithya B, Reshma J, Ragendhu S, Sumithra M D, “Diabetic Retinopathy Detection using Machine Learning”, International Journal of Engineering Research & Technology (IJERT), Vol. 9 Issue 06, June-2020.

3 AbdelMaksoud, E, Barakat, S, Elmogy, M (2022) A computer-aided diagnosis system for detecting various diabetic retinopathy grades based on a hybrid deep learning technique, Med Biol Eng Comput, pp.1–24 10.1007/s11517-022-02564-6.

4 Atwany MZ, Sahyoun AH, Yaqub M. Deep learning techniques for diabetic Retinopathy classification: a survey. IEEE Access. 2022;10:28642–28655. doi: 10.1109/ACCESS.2022.3157632.

5 Bodapati JD, Veeranjaneyulu N, Shareef SN, Hakak S, Bilal M, Maddikunta PKR, Jo O. Blended multi-modal deep convnet features for diabetic retinopathy severity prediction. Electronics. 2020;9(6):914. doi: 10.3390/electronics9060914.

6 Bora A, Balasubramanian S, Babenko B, Virmani S, Venugopalan S, Mitani A, Marinho GO, Cuadros J, Ruamviboonsuk P, Corrado GS, Peng L, Webster DR, Varadarajan AV, Hammel N, Liu Y, Bavishi P. Predicting the risk of developing diabetic retinopathy using deep learning. Lancet Digit Health. 2021;2020(3):e10–e19. doi: 10.1016/S2589-7500(20)30250-8.

7 Chaturvedi SS, Gupta K, Ninawe V, Prasad PS (2020) Automated diabetic retinopathy grading using deep convolutional neural network, arXiv:2004.06334v1 [eess.IV], pp 1–12. 10.48550/arXiv.2004.06334.

8 Das S, Kharbanda K, Suchetha M, Raman R, Edwin DD. Deep learning architecture based on segmented fundus image features for classification of diabetic retinopathy. Biomed Signal Process Control. 2021;68:1–10. doi: 10.1016/j.bspc.2021.102600.

9 Deepa, V, Kumar, SC, Cherian, T (2022) Automated grading of diabetic retinopathy using CNN with hierarchical clustering of image patches by siamese network, Physical and Engineering Sciences in Medicine, pp.1–13 10.1007/s13246-022-01129-z.

10 Dong, B, Wang, X, Qiang, X, Du, F, Gao, L, Wu, Q, Cao, G, Dai, C (2022) A Multi-Branch Convolutional Neural Network for Screening and Staging of Diabetic Retinopathy Based on Wide-Field Optical Coherence Tomography Angiography, IRBM, pp.1–7 10.1016/j.irbm.2022.04.004.