Deep Learning Fundus Image Analysis for Early Detection of Diabetic Retinopathy

Project Description

This project mainly focuses on developing an automated system to detect and classify Diabetic Retinopathy using retinal fundus images. Diabetic Retinopathy is a serious eye complication caused by prolonged diabetes and is one of the leading causes of vision impairment and blindness worldwide. Early detection and timely treatment are crucial to prevent permanent vision loss.

Traditional diagnosis of diabetic Retinopathy relies on manual examination of retinal images by ophthalmologists, which is time-consuming, costly, and subject to human error. With the increasing number of diabetic patients, there is a growing need for scalable and automated screening solutions. This project addresses this need by leveraging deep learning techniques for accurate and efficient DR detection.

This system utilises Convolutional Neural Networks and transfer learning models to automatically extract meaningful features from retinal fundus images. The model is trained to classify images into five stages of Diabetic Retinopathy: No DR, Mild, Moderate, Severe, and Proliferative. Image preprocessing techniques such as resizing, normalisation, and data augmentation are applied to enhance model performance and generalisation.

It can be deployed as a web-based application or integrated into hospital management systems to assist ophthalmologists in large-scale diabetic eye screening programs. The project demonstrates how artificial intelligence can contribute to improving healthcare accessibility and preventing avoidable blindness.

Pre-Requisites:

- **Anaconda Navigator and PyCharm / Spyder**:
    - Refer to the link below to download Anaconda Navigator
    - Link (PyCharm) : https://youtu.be/1ra4zH2G4o0
    - Link (Spyder) : https://youtu.be/5mDYijMfSzs
- **Python packages**:
- Open Anaconda prompt as administrator

- Type “pip install NumPy” and click enter.
- Type “pip install pandas” and click enter.
- Type “pip install TensorFlow==2.3.2” and click enter.
- Type “pip install keras==2.3.1” and click enter.
- Type “pip install Flask” and click enter.

Prior Knowledge:

You must have prior knowledge of the following topics to complete this project.

- Deep Learning Concepts
    - CNN: https://towardsdatascience.com/basics-of-the-classic-cnn-a3dce1225add
    - VGG16: https://medium.com/@mygreatlearning/what-is-vgg16-introduction-to-vgg16-f2d63849f615
    - ResNet-50: https://towardsdatascience.com/understanding-and-coding-a-resnet-in-keras-446d7ff84d33
    - Inception-V3: https://iq.opengenus.org/inception-v3-model-architecture/
    - Xception: https://pyimagesearch.com/2017/03/20/imagenet-vggnet-resnet-inception-xception-keras/
- Flask: Flask is a popular Python web framework, meaning it is a third-party Python library used for developing web applications.

Link: https://www.youtube.com/watch?v=lj4I_CvBnt0

Project Objectives:

By the end of this project, you will:

- Know fundamental concepts and techniques of transfer learning, like Xception.
- Gain a broad understanding of image data.
- Know how to pre-process/clean the data using different data pre-processing techniques.
- Know how to build a web application using the Flask framework.

Project Flow:

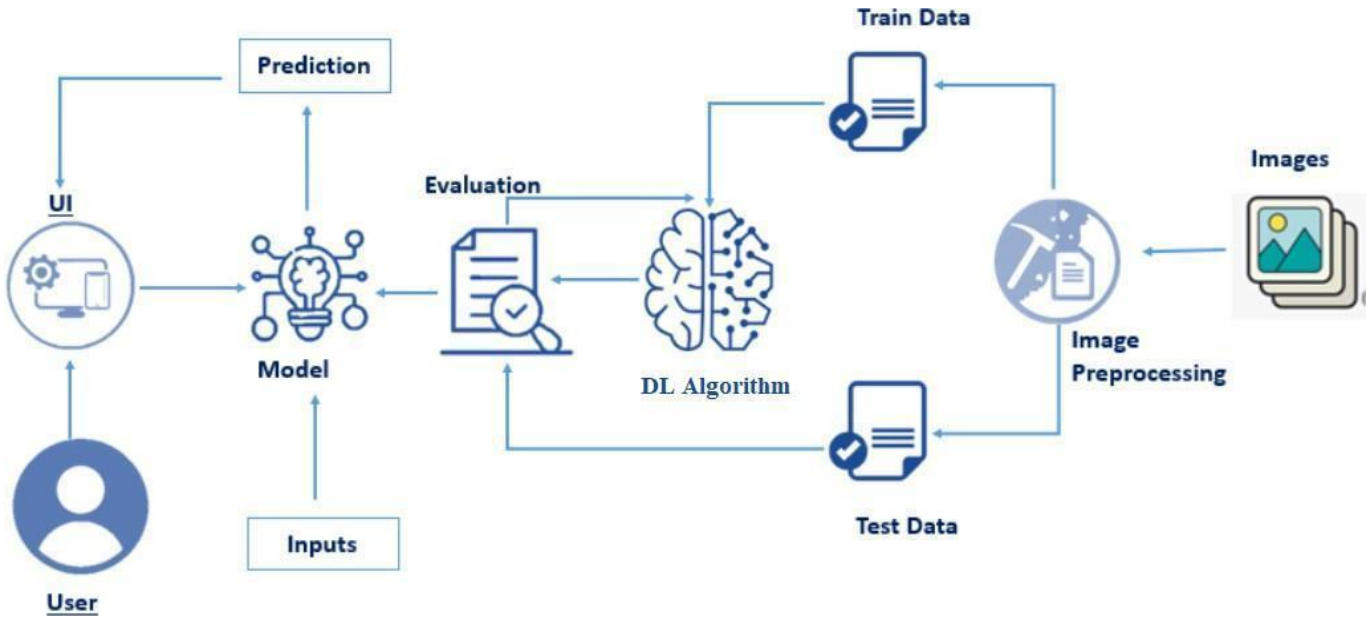
- The user interacts with the UI (User Interface) to choose the image.
- The chosen image is analysed by the model, which is integrated with a Flask application.

- The Xception Model analyses the image, then the prediction is showcased on the Flask UI.

To accomplish this, we have to complete all the activities and tasks listed below

- Data Collection.
    - Create a Train and Test path.
- Data Pre-processing.
- Import the required library
- Configure ImageDataGenerator class
- Apply the ImageDataGenerator functionality to the Trainset and Testset
- Model Building
    - Pre-trained CNN model as a Feature Extractor
    - Adding Dense Layer
    - Configure the Learning Process
    - Train the model
    - Save the Model
    - Test the model
- Cloudant DB
    - Register & Login to IBM Cloud
    - Create Service Instance
    - Creating Service Credentials
    - Launch Cloudant DB
    - Create Database
- Application Building
    - Create an HTML file
    - Build Python Code

Project Architecture:



Milestone 1: Data Collection

DL depends heavily on data; it is the most crucial aspect that makes algorithm training possible. So this section allows you to download the required dataset.

Activity 1: Download the Dataset

There are many popular open sources for collecting the data. Eg: kaggle.com, UCI repository, etc.

Download dataset: link

Activity 2: Create a Training and Testing Path

To build a DL model, we have to split the training and testing data into two separate folders. But in the project dataset folder, training and testing folders are present. So, in this case, we just have to assign a variable and pass the folder path to it.

Four different transfer learning models are used in our project, and the best model (Xception) is selected.

The image input size of xception model is 299, 299.

Milestone 2: Pre-Processing

In this milestone, we will be improving the image data that suppresses unwanted distortions or enhances some image features important for further processing, although perform some geometric transformations of images like rotation, scaling, translation, etc.

Activity 1: Importing the Libraries

```
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
```

Activity 2: Configure ImageDataGenerator class

The ImageDataGenerator class is instantiated, and the configuration for the types of data augmentation is provided. There are 5 types of data augmentation.

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

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

Activity 3: Apply ImageDataGenerator functionality to the Train set and the Test set

Let us apply ImageDataGenerator functionality to the Train set and Test set by using the following code. This function will return a batch of images from subdirectories.

```
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.
```

Milestone 3: Model Building

Now it's time to build our model. Let’s use the pre-trained model, which is Xception, one of the convolution neural net (CNN) architectures, which is considered a very good model for Image classification.

Deep understanding of the Xception model – Link is referred to in the prior knowledge section. Kindly refer to it before starting the model-building part.

Activity 1: Pre-trained CNN model as a Feature Extractor

We will use it as a simple feature extractor by freezing all five convolution blocks to make sure their weights don't get updated after each epoch as we train our own model.

Here, we have considered images of dimension (229,229,3).

Also, we have assigned include_top = False because we are using the convolution layer for feature extraction and want to train a fully connected layer for our image classification.

```
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)
```

Activity 2: Adding Dense Layers

A dense layer is a deeply connected neural network layer. It is the most common and frequently used layer.

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

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

The number of neurons in the Dense layer is the same as the number of classes in the training set. The neurons in the last Dense layer use softmax activation to convert their outputs into respective probabilities.

Understanding the model is a very important phase in properly using it for training and prediction purposes. Keras provides a simple method, a summary, to get the full information about the model and its layers.

Activity 3: Configure the Learning Process

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.

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

Activity 4: Train the Model

Now, let us train our model with our image dataset. The model is trained for 20 epochs, and after every epoch, the current model state is saved if the model has the least loss encountered till that time. We can see that the training loss decreases in almost every epoch till 10 epochs, and probably there is further scope to improve the model.

fit generator functions used to train a deep learning neural network

Arguments:

- steps_per_epoch: it specifies the total number of steps taken from the generator as soon as one epoch is finished and the next epoch has started. We can calculate the value of steps_per_epoch as the total number of samples in your dataset divided by the batch size.
- Epochs: an integer and the number of epochs we want to train our model for.
- validation_data can be either:

  - an input and target list

  - a generator

  - inputs, targets, and sample_weights list, which can be used to evaluate

  The loss and metrics for any model after any epoch have ended.
- validation_steps: only if the validation_data is a generator, then only this argument

Milestone 4: Save the Model & Create Cloudant DB

The model is saved with the .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')
```

Cloudant is a non-relational, distributed database service.

Below are the steps that need to be followed for creating and using the Cloudant service.

- Register & Login to IBM Cloud
- Create Service Instance
- Creating Service Credentials
- Launch Cloudant DB
- Create Database

Milestone 5: Application Building

In this section, we will be building a web application that is integrated with the model we built. A UI is provided to the user where he has uploaded the image. Based on the saved model, the uploaded image will be analysed, and the prediction is showcased on the UI.

This section has the following tasks

- Building HTML Pages
- Building server-side script

Activity 1: Building Html Pages

For this project, create five HTML files, namely

- index.html
- register.html
- login.html
- prediction.html
- logout.html

Activity 2: Build Python code

Activity 3: Run the application

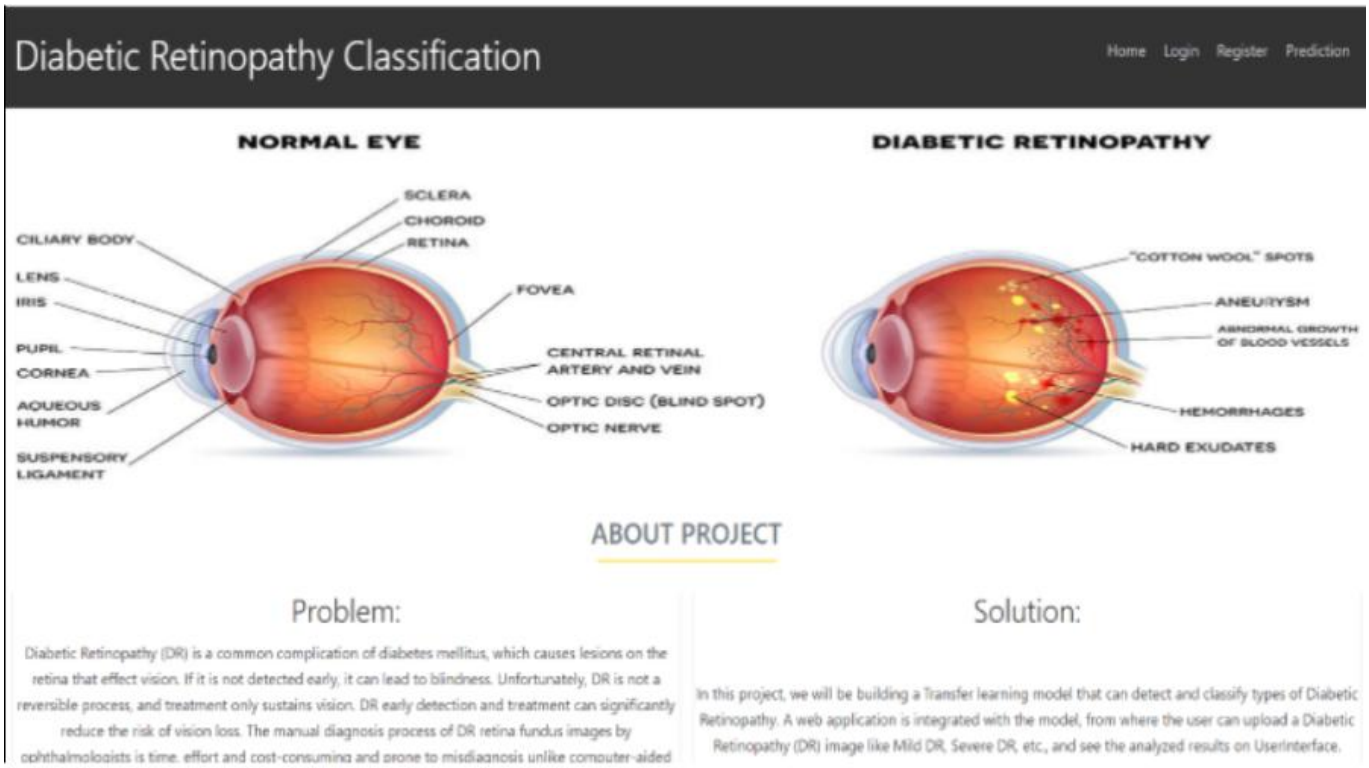
- Open Anaconda prompt from the start menu
- Navigate to the folder where your Python script is.
- Now type the “python app.py” command
- Navigate to the localhost where you can view your web page.

- Click on the predict button from the top right corner, enter the inputs, click on the submit button, and see the result/prediction on the web.

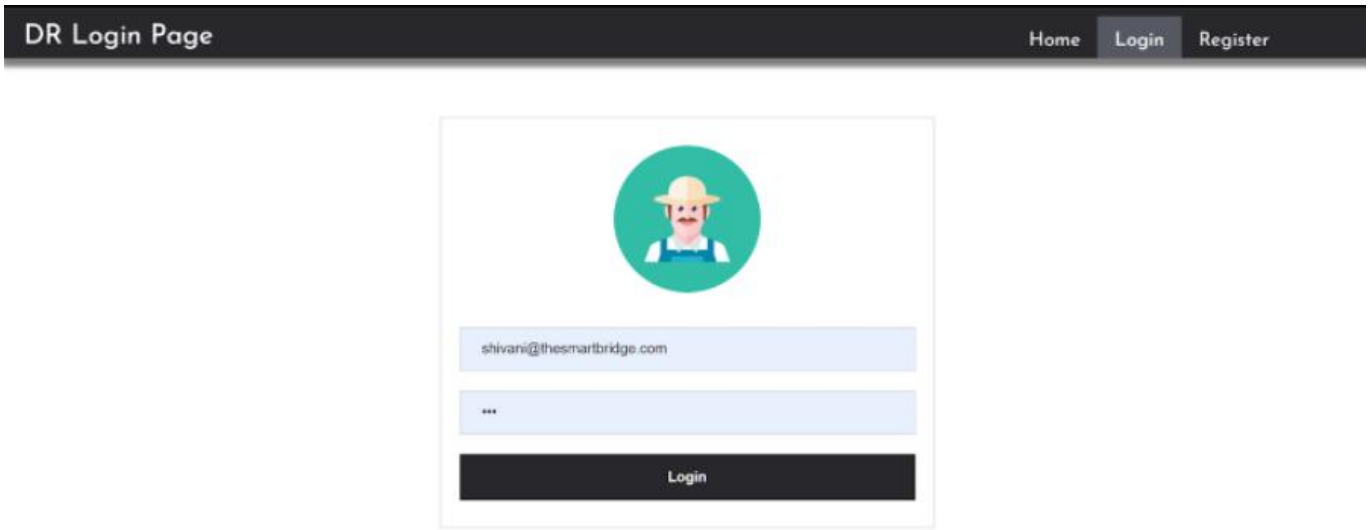
1: Run the application

2: Open the browser and navigate to localhost:5000 to check your application

The home page looks like this. You can click on login or register.

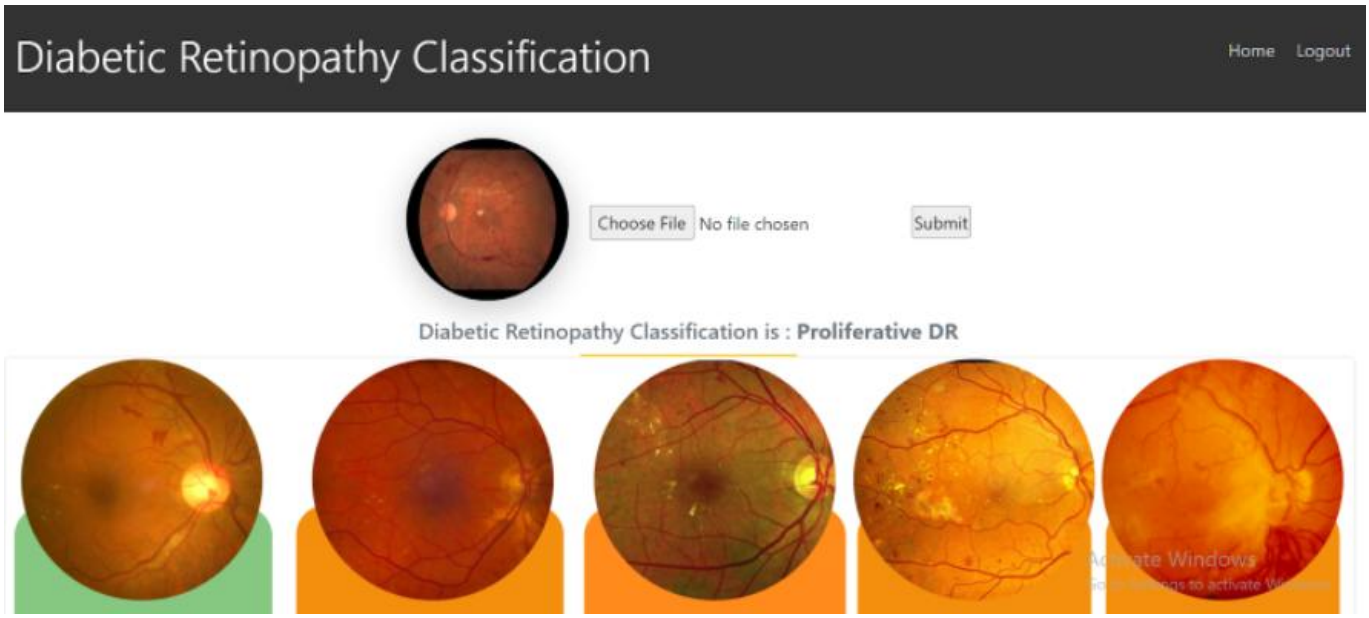


While logging in, you need to provide your registered credentials,



After successfully login you will be redirected to the prediction page where we have to upload the image to predict the outcomes.

Output:


Diabetic Retinopathy Classification
Home
Logout
Choose File
No file chosen
Submit
Diabetic Retinopathy Classification is : Proliferative DR