

Deep Learning Fundus Image Analysis For Early Detection Of Diabetic Retinopathy

Project Description:

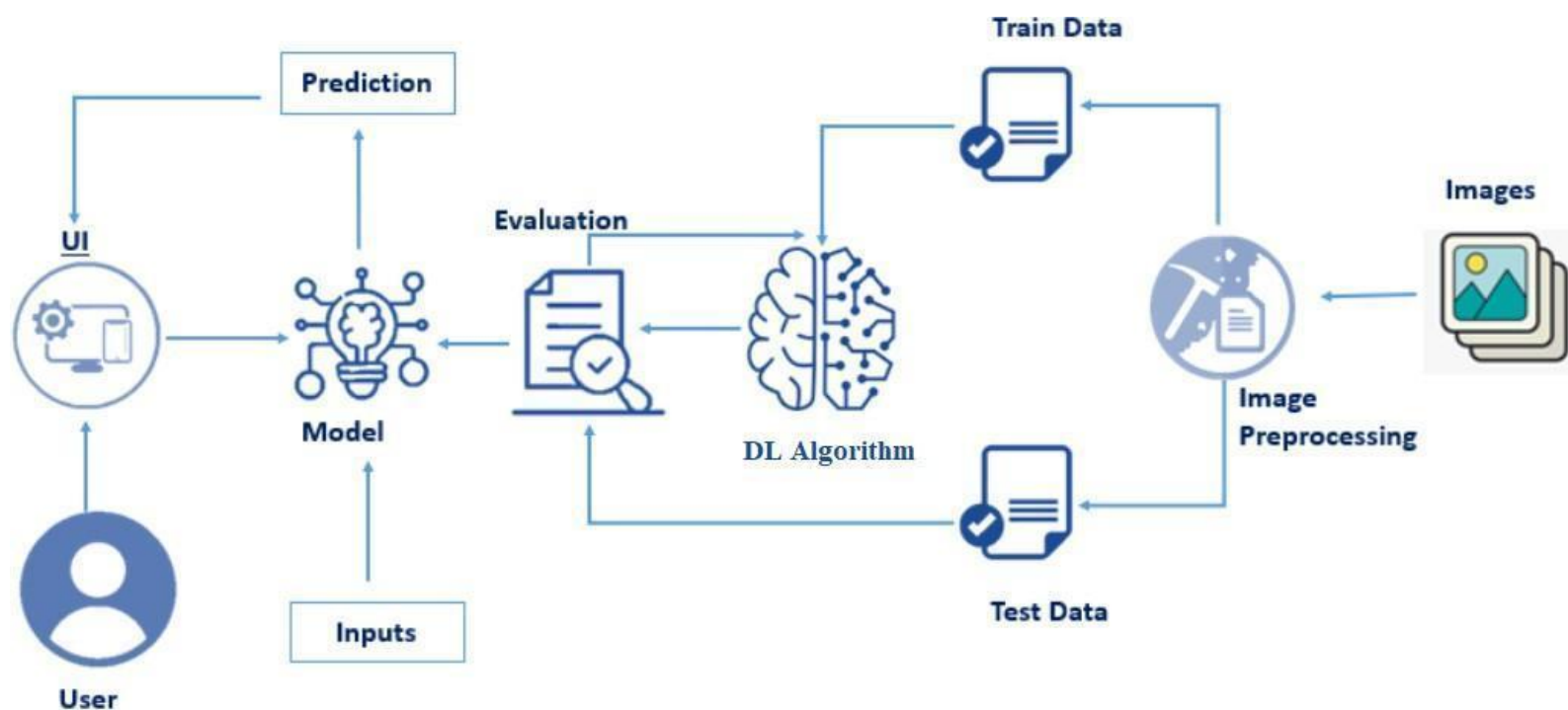
Deep Learning Fundus Image Analysis for Early Detection of Diabetic Retinopathy is a medical imaging project aimed at using advanced deep learning techniques to analyze fundus images of the retina. The goal is to detect signs of diabetic retinopathy in its early stages, enabling timely intervention and treatment to prevent vision loss in diabetic patients. By leveraging deep learning models, such as convolutional neural networks (CNNs), the system can automatically identify and classify retinal abnormalities with high accuracy and efficiency.

Scenario 1: Early Disease Detection Healthcare professionals can utilize the deep learning system for early detection of diabetic retinopathy in patients with diabetes. By analyzing fundus images, the system can flag potential abnormalities, allowing doctors to intervene early, monitor disease progression, and provide timely treatment to prevent vision impairment or blindness.

Scenario 2: Screening Programs Public health organizations and clinics can implement screening programs using the deep learning-based analysis tool. This enables large-scale screening of diabetic patients for retinopathy, helping to identify at-risk individuals who may require further evaluation and management by eye care specialists.

Scenario 3: Telemedicine and Remote Monitoring Telemedicine platforms can integrate the deep learning fundus image analysis tool to facilitate remote monitoring of diabetic retinopathy. Patients can capture fundus images using portable retinal cameras, and the deep learning system can analyze these images to assess disease status, allowing for remote consultations and follow-ups with healthcare providers.

Technical Architecture:



Pre requisites:

To complete this project, you must required following software's, concepts and packages

- **Anaconda navigator and PyCharm / Spyder:**
 - Refer 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 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 analyzed by the model which is integrated with flask application.
- The Xception Model analyzes the image, then the prediction is showcased on the Flask UI.

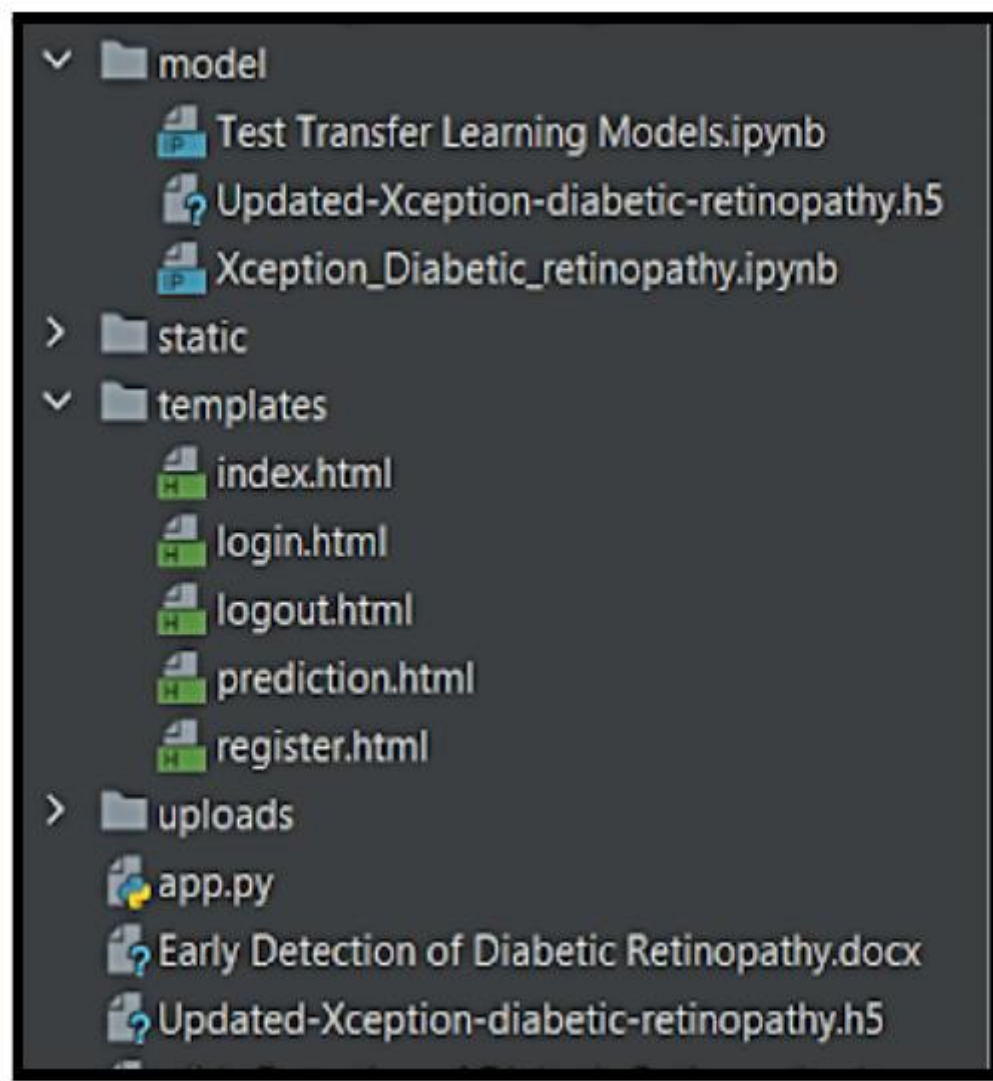
To accomplish this, we have to complete all the activities and tasks listed below

- Data Collection.
 - Create a Train and Test path.
- Data Pre-processing.
 - Import the required library
 - Configure ImageDataGenerator class
 - ApplyImageDataGenerator functionality to 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

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

Project Structure:

Create the Project folder which contains files as shown below



- We are building a flask application that needs HTML pages stored in the templates folder, CSS, Images stored in a static folder, and a python script app.py for scripting.
- Updated-Xception-diabetic-retinopathy.h5 is our saved model. Further, we will use this model for flask integration.
- The model contains model training files.

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](#)

We are going to build our training model on Google colab.

Introduction to google colab: <https://www.youtube.com/watch?v=vVe648dJOdI>

In our project, we are not going to upload the dataset on colab. We are going to clone the Kaggle dataset on colab. Kindly refer to this - <https://www.analyticsvidhya.com/blog/2021/06/how-to-load-kaggle-datasets-directly-into-google-colab/>

- Clone kaggle in google colab

```
!pip install -q kaggle

!mkdir ~/.kaggle # creating a kaggle directory

!cp kaggle.json ~/.kaggle/ # copying json file to folder

!chmod 600 ~/.kaggle/kaggle.json # changing the permissions to json
```

- Download the dataset

```
!kaggle datasets download -d arbethi/diabetic-retinopathy-level-detection

Downloading diabetic-retinopathy-level-detection.zip to /content
100% 9.64G/9.66G [01:08<00:00, 195MB/s]
100% 9.66G/9.66G [01:08<00:00, 151MB/s]
```

- Unzip the dataset

```
!kaggle datasets download -d arbethi/diabetic-retinopathy-level-detection

Downloading diabetic-retinopathy-level-detection.zip to /content
100% 9.64G/9.66G [01:08<00:00, 195MB/s]
100% 9.66G/9.66G [01:08<00:00, 151MB/s]
```

Milestone 2:

Create Training And Testing Path

To build a DL model we have to split training and testing data into two separate folders. But In the project dataset folder training and testing folders are presented. 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.

```
imageSize = [299, 299]
trainPath = r"/content/preprocessed dataset/preprocessed dataset/training"
testPath = r"/content/preprocessed dataset/preprocessed dataset/testing"
```

Milestone 3: Data Pre-processing

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

Activity 1: import the necessary libraries as shown in the image.

```
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

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.
- Image 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)
```

Activity 3: Apply ImageDataGenerator Functionality To Train Set And Test Set

Let us 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`:
 - 'int': means that the labels are encoded as integers (e.g. for `sparse_categorical_crossentropy` loss).
 - 'categorical' means that the labels are encoded as a categorical vector (e.g. for `categorical_crossentropy` loss).
 - 'binary' means that the labels (there can be only 2) are encoded as float32 scalars with values 0 or 1 (e.g. for `binary_crossentropy`).
 - None (no labels).

```
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 4: 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 as a very good model for Image classification.

Deep understanding on 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

For one of the models, we will use it as a simple feature extractor by freezing all the 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 features extraction and want to train a fully connected layer for our images classification(since it is not the part of Imagenet dataset)

Flatten layer flattens the input. 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)
```

Activity 2: Adding Dense Layers

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

Let us 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 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 to 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.

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

Model: "model"

Layer (type)	Output Shape	Param #	Connected to
input_1 (InputLayer)	[(None, 299, 299, 3)]	0	[]
block1_conv1 (Conv2D)	(None, 149, 149, 32)	864	['input_1[0][0]']
block1_conv1_bn (BatchNormaliz ation)	(None, 149, 149, 32)	128	['block1_conv1[0][0]']
block1_conv1_act (Activation)	(None, 149, 149, 32)	0	['block1_conv1_bn[0][0]']
block1_conv2 (Conv2D)	(None, 147, 147, 64)	18432	['block1_conv1_act[0][0]']
block1_conv2_bn (BatchNormaliz ation)	(None, 147, 147, 64)	256	['block1_conv2[0][0]']
block1_conv2_act (Activation)	(None, 147, 147, 64)	0	['block1_conv2_bn[0][0]']
block2_sepconv1 (SeparableConv 2D)	(None, 147, 147, 12)	8768	['block1_conv2_act[0][0]']

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

Model: "model"

Layer (type)	Output Shape	Param #	Connected to
input_1 (InputLayer)	[(None, 299, 299, 3)]	0	[]
block1_conv1 (Conv2D)	(None, 149, 149, 32)	864	['input_1[0][0]']
block1_conv1_bn (BatchNormaliz ation)	(None, 149, 149, 32)	128	['block1_conv1[0][0]']
block1_conv1_act (Activation)	(None, 149, 149, 32)	0	['block1_conv1_bn[0][0]']
block1_conv2 (Conv2D)	(None, 147, 147, 64)	18432	['block1_conv1_act[0][0]']
block1_conv2_bn (BatchNormaliz ation)	(None, 147, 147, 64)	256	['block1_conv2[0][0]']
block1_conv2_act (Activation)	(None, 147, 147, 64)	0	['block1_conv2_bn[0][0]']
block2_sepconv1 (SeparableConv 2D)	(None, 147, 147, 12)	8768	['block1_conv2_act[0][0]']

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.

Optimization is an important process that optimizes the input weights by comparing the prediction and the loss function. Here we are using adam optimizer

Metrics 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']
)
```

Activity 4: Train The Model

Now, let us train our model with our image dataset. The model is trained for 30 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 number of epochs we want to train our model for.
- validation_data can be either:
 - an inputs and targets 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 has ended.
- validation_steps: only if the validation_data is a generator then only this argument

can be used. It specifies the total number of steps taken from the generator before it is stopped at every epoch and its value is calculated as the total number of validation data points in your dataset divided by the validation batch size.

```
# 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
)
```

```
Epoch 1/30
3/3 [=====] - 32s 6s/step - loss: 9.6582 - accuracy: 0.4479
Epoch 2/30
3/3 [=====] - 15s 5s/step - loss: 9.3711 - accuracy: 0.5417
Epoch 3/30
3/3 [=====] - 15s 5s/step - loss: 7.4651 - accuracy: 0.5208
Epoch 4/30
3/3 [=====] - 16s 5s/step - loss: 4.8766 - accuracy: 0.5938
Epoch 5/30
3/3 [=====] - 15s 5s/step - loss: 6.3676 - accuracy: 0.6458
Epoch 6/30
3/3 [=====] - 14s 5s/step - loss: 5.2558 - accuracy: 0.6667
Epoch 7/30
3/3 [=====] - 16s 5s/step - loss: 5.4306 - accuracy: 0.6458
Epoch 8/30
3/3 [=====] - 13s 4s/step - loss: 4.8280 - accuracy: 0.6562
Epoch 9/30
3/3 [=====] - 14s 5s/step - loss: 3.9236 - accuracy: 0.6458
Epoch 10/30
3/3 [=====] - 13s 4s/step - loss: 3.2804 - accuracy: 0.6562
Epoch 11/30
3/3 [=====] - 15s 5s/step - loss: 2.1550 - accuracy: 0.6875
Epoch 12/30
3/3 [=====] - 15s 5s/step - loss: 3.0436 - accuracy: 0.6979
Epoch 13/30
3/3 [=====] - 14s 5s/step - loss: 3.4109 - accuracy: 0.7500
Epoch 14/30
3/3 [=====] - 15s 5s/step - loss: 2.8810 - accuracy: 0.7396
Epoch 15/30
3/3 [=====] - 15s 5s/step - loss: 3.4979 - accuracy: 0.6667
Epoch 16/30
3/3 [=====] - 14s 5s/step - loss: 3.1029 - accuracy: 0.6562
Epoch 17/30
3/3 [=====] - 14s 5s/step - loss: 2.8477 - accuracy: 0.6979
Epoch 18/30
3/3 [=====] - 14s 4s/step - loss: 2.6290 - accuracy: 0.6979
Epoch 19/30
3/3 [=====] - 16s 5s/step - loss: 4.5827 - accuracy: 0.5938
```

```
Epoch 20/30
3/3 [=====] - 13s 4s/step - loss: 1.7713 - accuracy: 0.7604
Epoch 21/30
3/3 [=====] - 14s 5s/step - loss: 4.1266 - accuracy: 0.6042
Epoch 22/30
3/3 [=====] - 15s 5s/step - loss: 2.2045 - accuracy: 0.7188
Epoch 23/30
3/3 [=====] - 15s 5s/step - loss: 2.7497 - accuracy: 0.7500
Epoch 24/30
3/3 [=====] - 16s 5s/step - loss: 3.3502 - accuracy: 0.7083
Epoch 25/30
3/3 [=====] - 15s 5s/step - loss: 3.1592 - accuracy: 0.7188
Epoch 26/30
3/3 [=====] - 14s 5s/step - loss: 3.2060 - accuracy: 0.6354
Epoch 27/30
3/3 [=====] - 16s 5s/step - loss: 3.4886 - accuracy: 0.6250
Epoch 28/30
3/3 [=====] - 15s 5s/step - loss: 3.0558 - accuracy: 0.6979
Epoch 29/30
3/3 [=====] - 14s 5s/step - loss: 4.7360 - accuracy: 0.6250
Epoch 30/30
3/3 [=====] - 14s 5s/step - loss: 2.0049 - accuracy: 0.7708
```

Milestone 5: Save The Model & Create Cloudant DB

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')
```

Cloudant is a non-relational, distributed database service.

Below are steps that need to follow for creating and using Cloudant service.

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

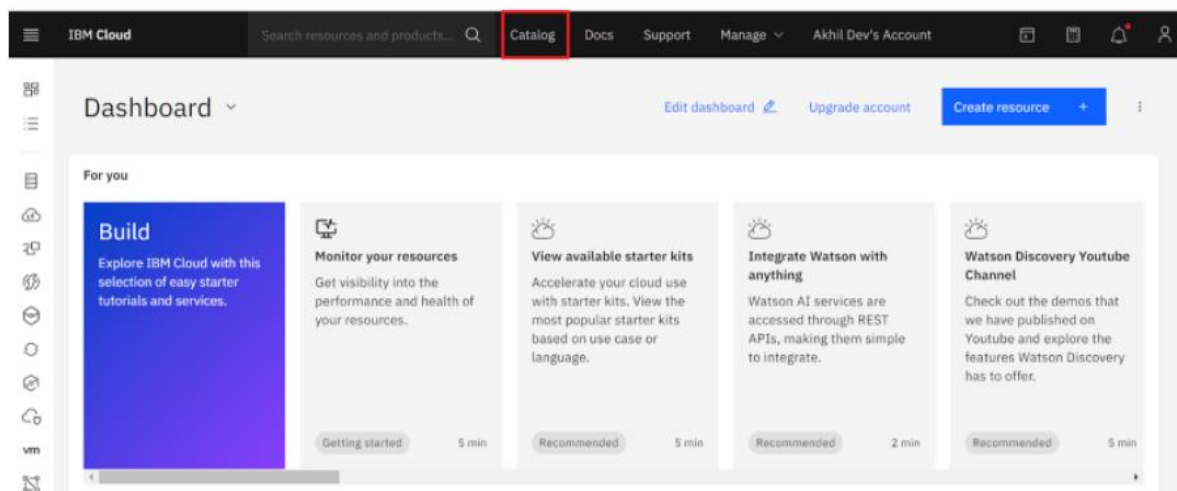
Activity 1:

Register & Login To IBM Cloud

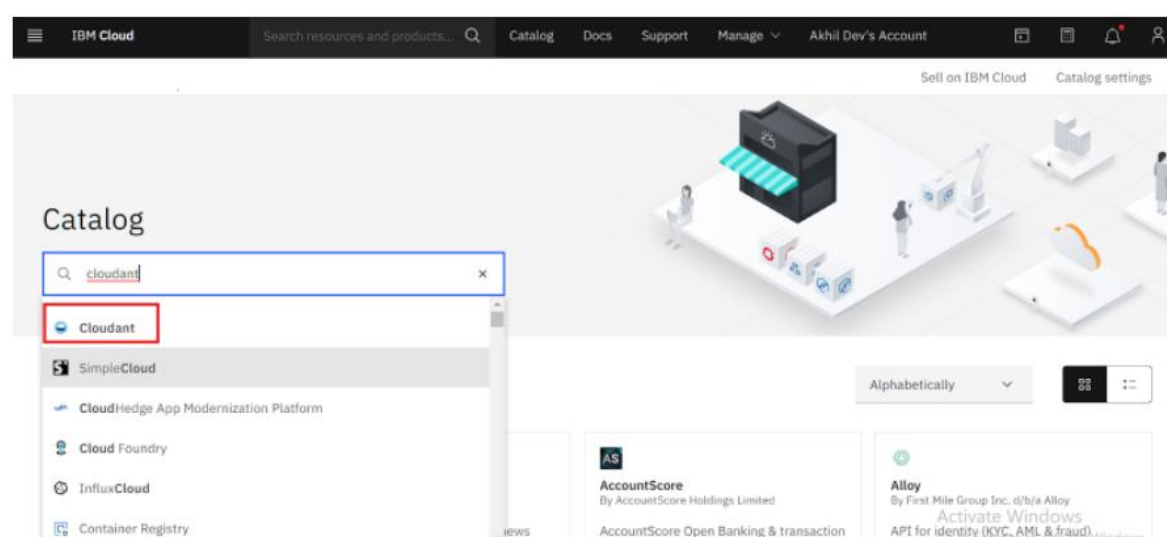
1. Register To IBM Cloud:- <https://cloud.ibm.com/registration/trial>
2. Sign in with your credentials: <https://cloud.ibm.com/login>

Activity 2: Create Service Instance

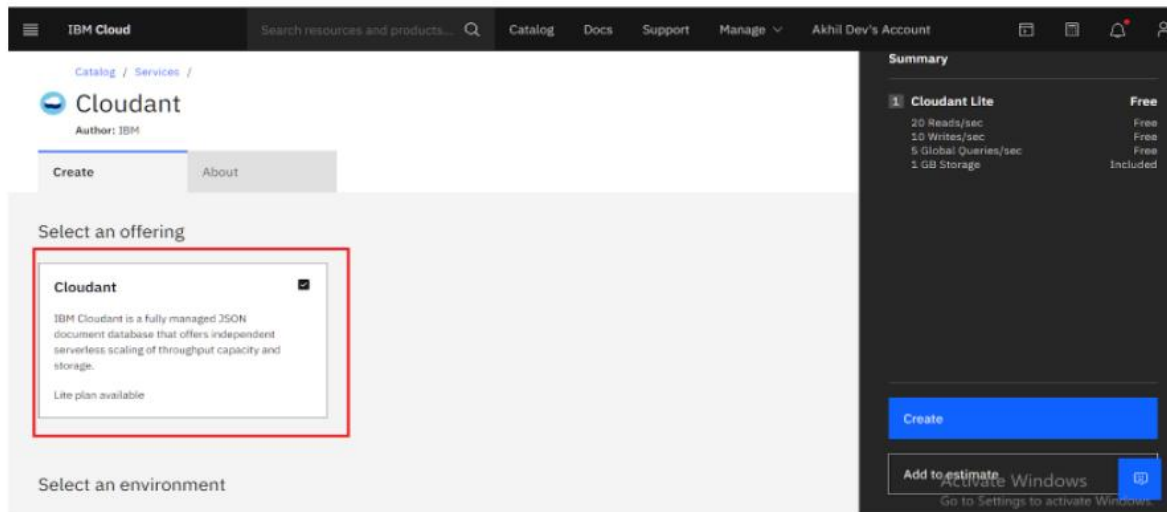
- Log in to your IBM Cloud account, and click on Catalog



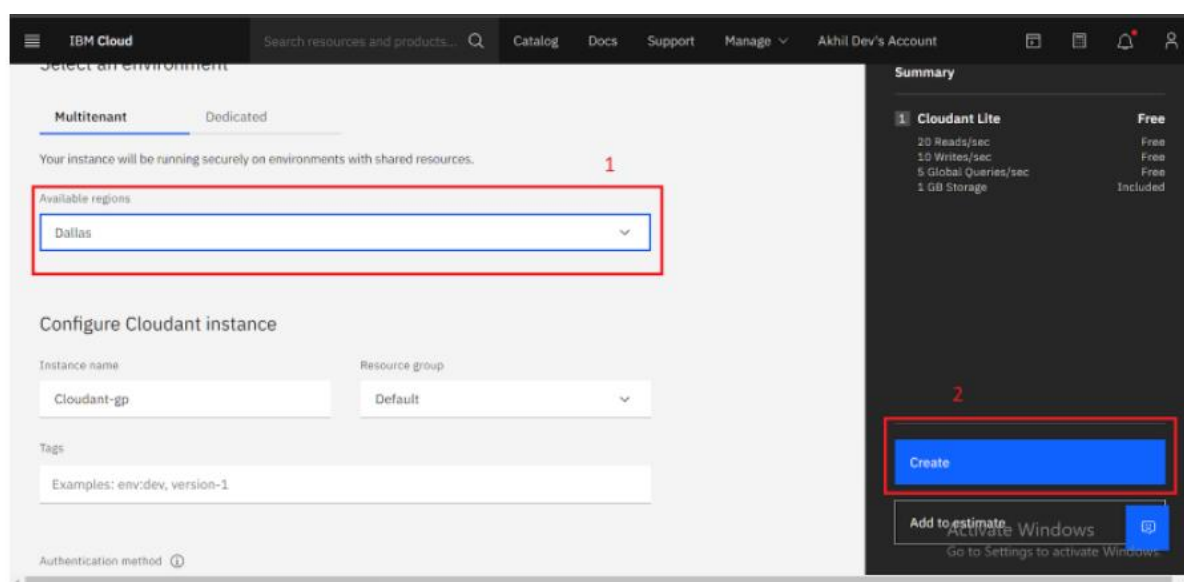
- Type Cloudbant in the Search bar and click to open it.



- Select an offering and an environment



- Select region as Dallas & Type an instance name then click on create service.

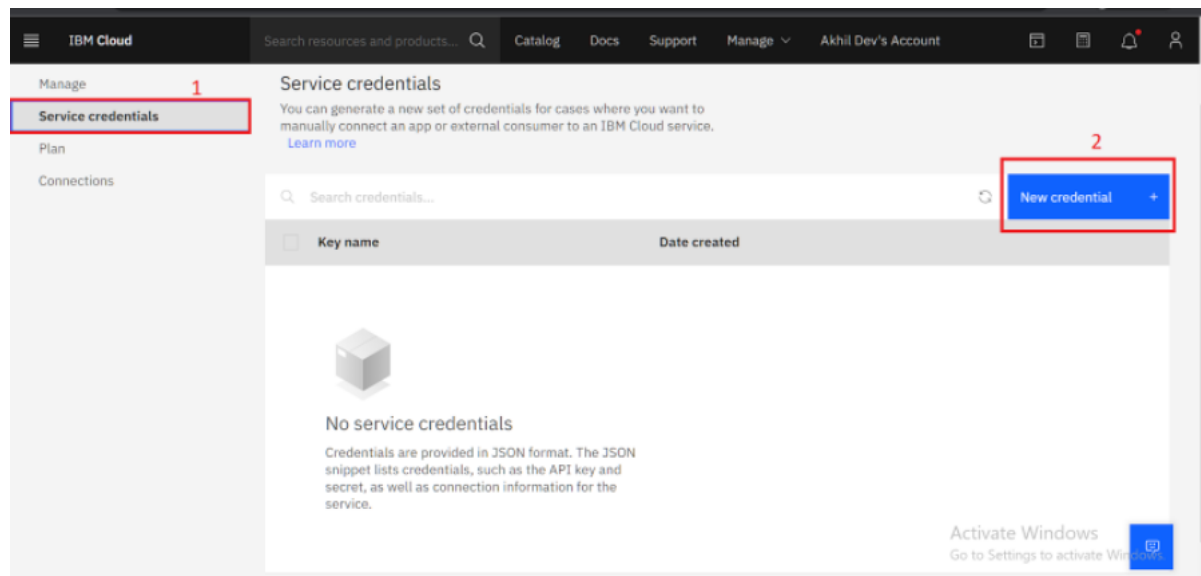


After you click create the system displays a message to say that the instance is being provisioned, which returns you to the Resource list. From the Resource list, you see that the status for your instance is, Provision in progress.

- When the status changes to Active, click the instance.

Activity 3: Creating Service Credentials

1. To create the connection information that your application needs to connect to the instance, click New credential.



2. Enter a name for the new credential in the Add new credential window.
3. Accept the Manager role.
4. (Optional) Create a service ID or have one automatically generated for you.
5. (Optional) Add inline configuration parameters. This parameter isn't used by IBM Cloudant service credentials, so ignore it.
6. Click Add.

Create credential

Name:

Service credentials-1

Role: ⓘ

Manager

Advanced options ^

Select Service ID (Optional) ⓘ

Auto Generate

Provide service-specific configuration parameters in a valid JSON object (Optional)

Choose file

Add inline configuration parameters (Optional)

Cancel

Add

7. To see the credentials that are required to access the service, click the chevron.

IBM Cloud

Search resources and products...

Catalog Docs Support Manage ▾ Hari S's Account

Resource list /

Cloudant-4k Active Add tags

Details Actions...

Manage

Service credentials

Plan

Connections

Service credentials

You can generate a new set of credentials for cases where you want to manually connect an app or external consumer to an IBM Cloud service. [Learn more](#)

Search credentials...

New credential +

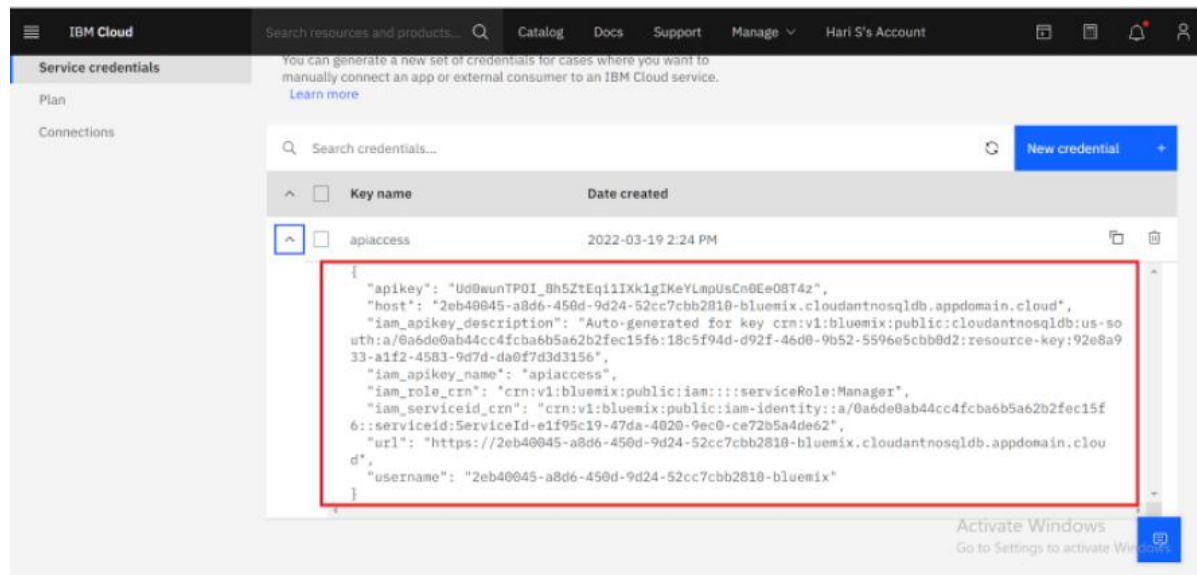
Key name

Date created

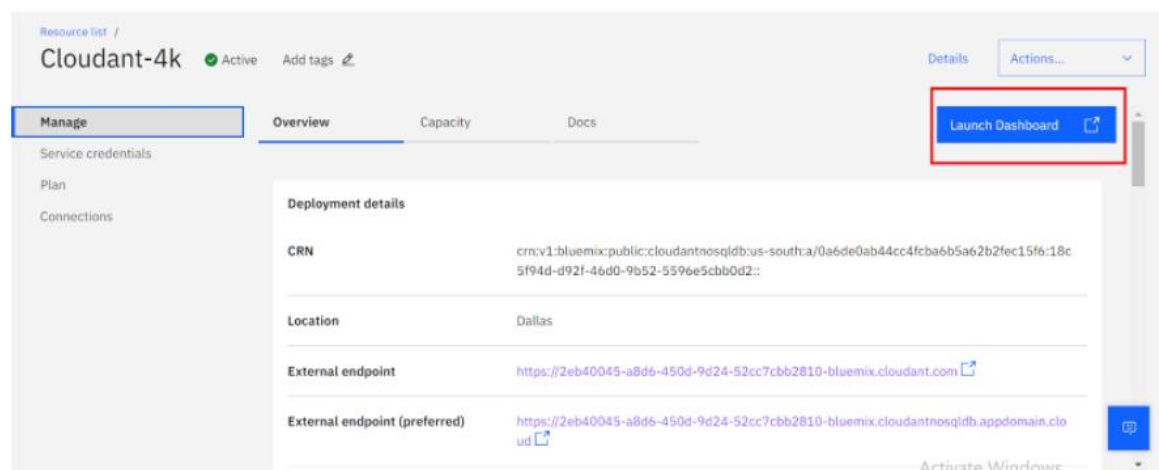
apiaccess

2022-03-19 2:24 PM

8. The details for the service credentials open like the following example:

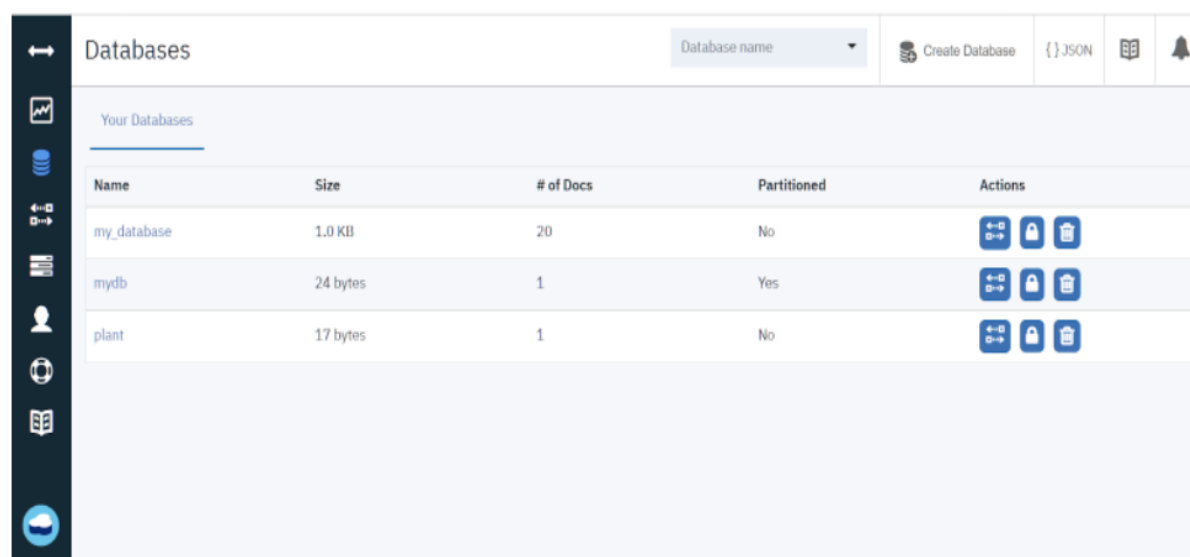


Activity 4: Launch Cloudant DB



Your Cloudant DB launches

Note: If You are a New User you will find empty database



Let's create the Database Now

Activity 5: Create Database

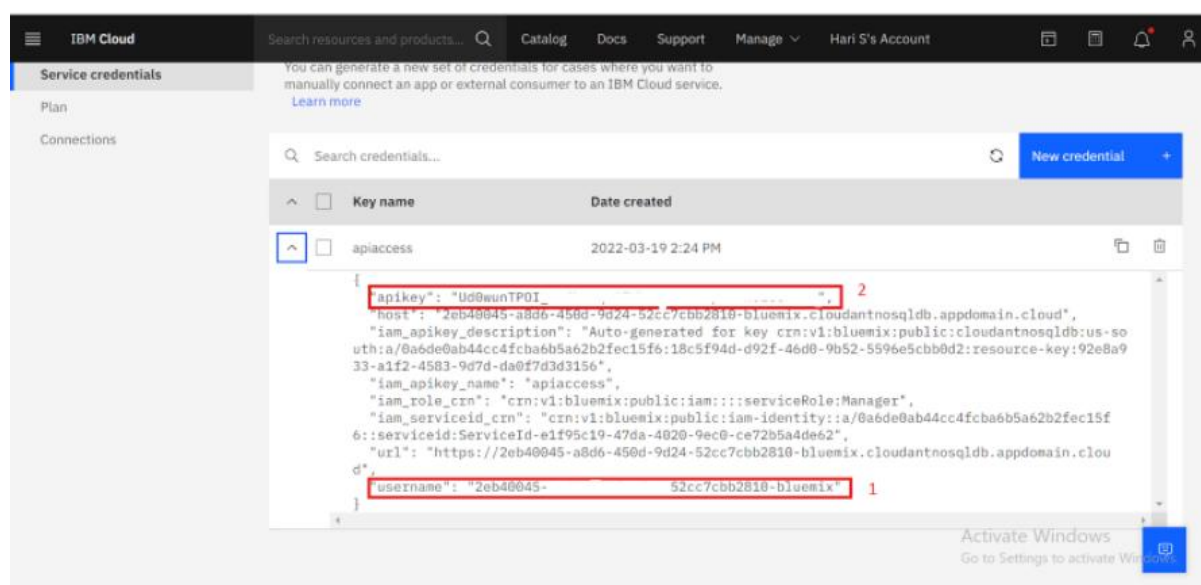
- In order to manage a connection from a local system, you must first initialize the connection by constructing a Cloudbant client. We need to import the cloudbant library.

```
from cloudbant.client import Cloudbant
```

- 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.

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

In the above cloudbant.iam() method we have to give a username & apikey to build the connection with cloudbant DB.



- Once a connection is established you can then create a database, and open an existing database.
- Create a database as my_database.

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

Milestone 5: Application Building

In this section, we will be building a web application that is integrated to 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 analyzed and prediction is showcased on the UI.

This section has the following tasks

- Building HTML Pages
- Building server side script

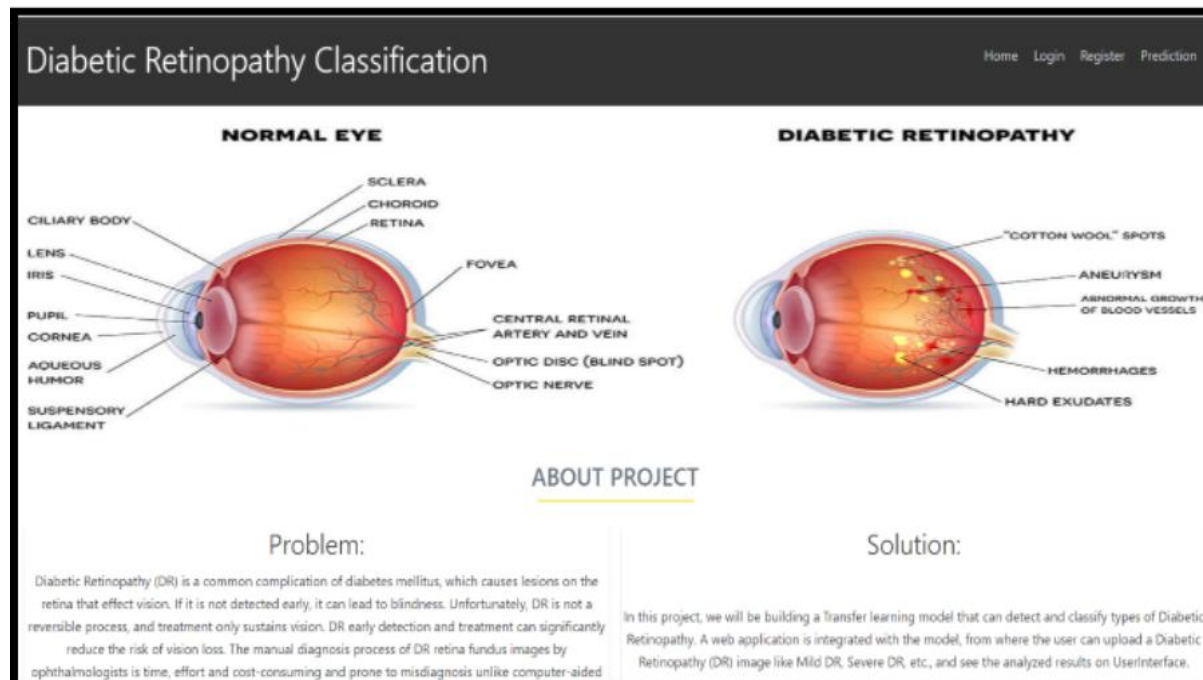
Activity 1 :Building Html Pages:

For this project create three HTML files namely

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

and save them in the templates folder.

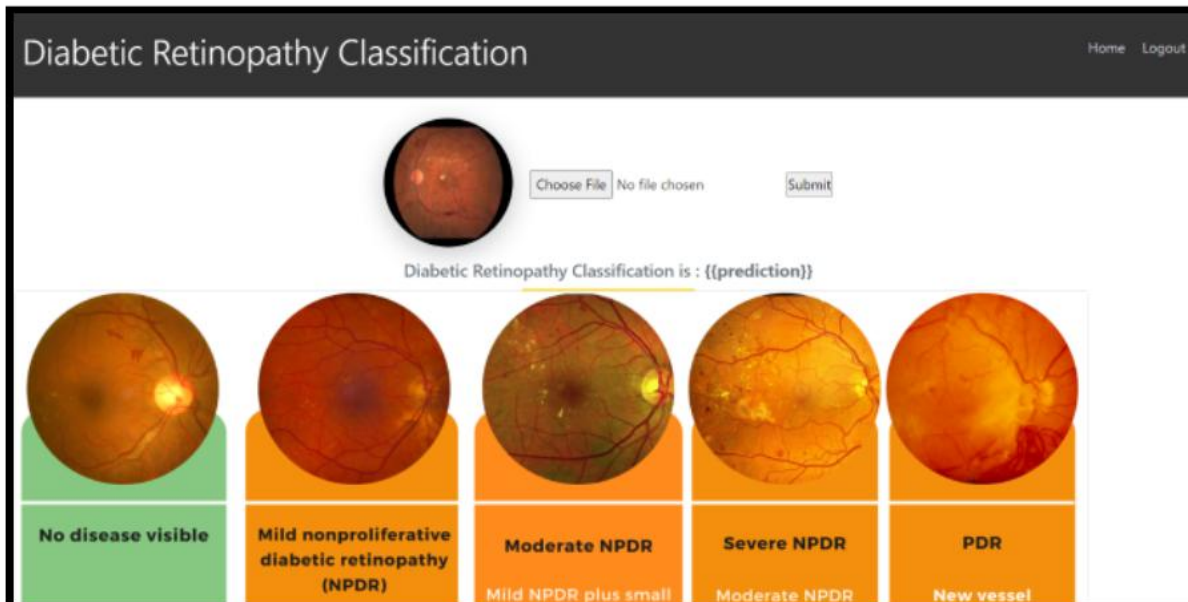
Let's see how our index.html page looks like:



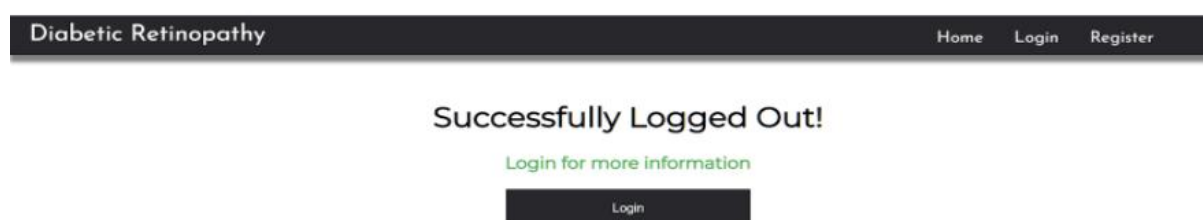
Let's look how our register.html file looks like:

Let's look how our login.html file looks like:

Now it will redirect to the prediction.html page.



When logout is clicked it redirects to the logout.html page.



Activity 2: Build Python code:

Import the libraries

```
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')

```

Here we will be using a declared constructor to route to the HTML page which we have created earlier.

In the above example, '/' URL is bound with the home.html function. Hence, when the home page of the web server is opened in the browser, the html page will be rendered. Whenever you enter the values from the html page the values can be retrieved using POST Method.

Configure the registration page

Based on user input into the registration form we stored it on data dictionary then we can validate the data using _id parameter with user input that we can 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. If the length of docs is 0 then user will register successfully on the platform and user data will store on the database. Otherwise it shows the message as user already registered please login and use our web application for DR prediction.

```

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

@app.route('/afterreg', methods=['POST'])
def afterreg():
    x = [x for x in request.form.values()]
    print(x)
    data = {
        'id': x[1], # Setting _id is optional
        'name': x[0],
        'psw': x[2]
    }
    print(data)

    query = {'_id': {'$eq': data['_id']}}

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

    print(len(docs.all()))

    if(len(docs.all())==0):
        url = my_database.create_document(data)
        #response = requests.get(url)
        return render_template('register.html', pred="Registration Successful, please login using your details")
    else:
        return render_template('register.html', pred="You are already a member, please login using your details")

```

Configure the login page

Based on user input into the login form we stored user id and password into the (user,passw) variables. Then we can validate the credentials using _id parameter with user input that we can 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. If the length of doc is 0 then it means username is not found. Otherwise its 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 our web application for DR prediction. Otherwise the user needs to provide correct credentials.

```

#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')

```

For logout from web application.

```

@app.route('/logout')
def logout():
    return render_template('logout.html')

```

Showcasing prediction on UI:

```

@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)

```

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.

Main Function:

```

if __name__ == "__main__":
    app.run(debug=False)

```

Activity 3: Run the application

Run The Application

- Open anaconda prompt from the start menu
- Navigate to the folder where your python script is.
- Now type “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.

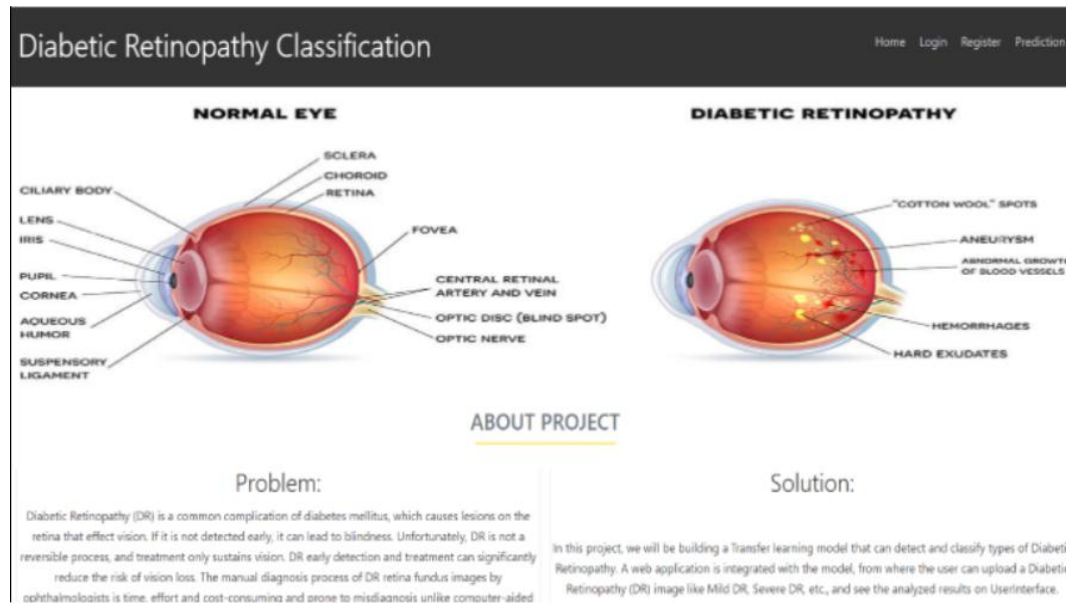
1: Run the application

In the anaconda prompt, navigate to the folder in which the flask app is present. When the python file is executed the localhost is activated on 5000 port and can be accessed through it.

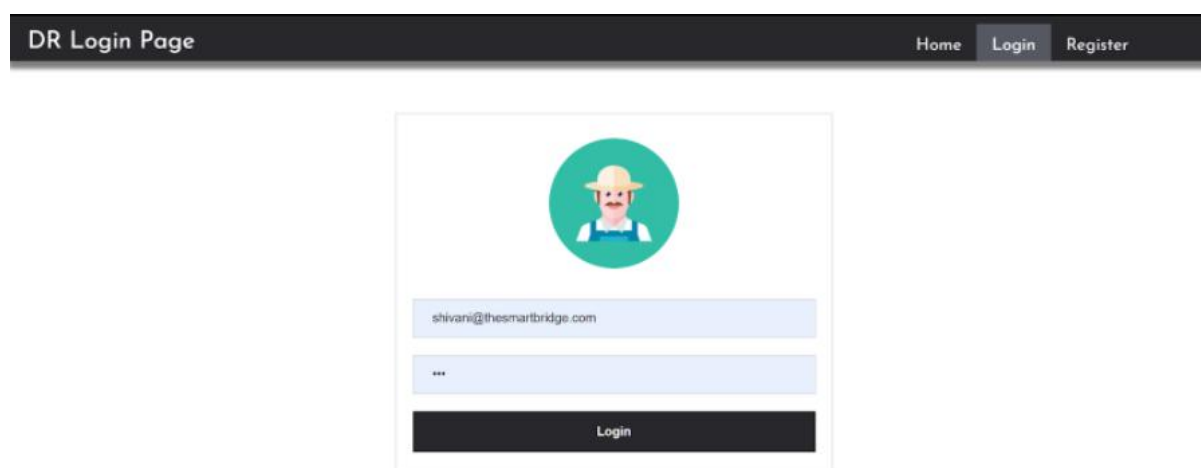
```
Serving Flask app "app" (lazy loading)
Environment: production
WARNING: This is a development server. Do not use it in a
Use a production WSGI server instead.
Debug mode: off
Running on http://127.0.0.1:5000/ (Press CTRL+C to quit)
```

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 redirect to the prediction page where we have to upload the image to predict the outcomes.

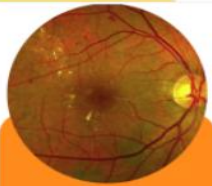
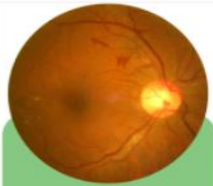


Choose File

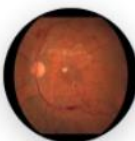
No file chosen

Submit

Diabetic Retinopathy Classification is :



Output:



Choose File

No file chosen

Submit

Diabetic Retinopathy Classification is : **Proliferative DR**

