Project Report

On

**Deep Learning Fundus Image Analysis For Early Detection Of Diabetic Retinopathy**

Submitted

In fulfilment of the requirement of

**Faculty Buildathon 2023**

Submitted by Submitted to

Dr. Jatin Arora IBM SmartInternz

Chitkara University Faculty Buildathon

Punjab

**Content**

**1. Introduction 3**

1.1 Overview 3

1.2. Purpose 3

**2.  LITERATURE SURVEY 4**

2.1. Existing problem 4

2.2. Pre Requisite 5

2.3. Proposed solution 6

**3.   Diabetic Retinopathy Level Detection Model 7**

3.1. Dataset Download 7

3.2. Create Training And Testing Path 8

3.3. Importing The Libraries 8

3.4. Configure ImageDataGenerator Class 9

3.5. Apply ImageDataGenerator Functionality To Train Set And Test Set 9

**4. Model Building 10**

4.1. Pre-Trained CNN Model As A Feature Extractor 10

4.2. Adding Dense Layers 10

4.3. Configure The Learning Process 12

4.4. Train The Model 12

4.5. Save The Model 15

**5. IBM Cloud Access 15**

5.1. Register & Login To IBM Cloud 15

5.2. Creating Service Credentials 17

5.3. Launch Cloudant DB 18

5.4. Create Database 19

**6 Application Building 20**

6.1. Building Html Pages 20

6.2. Build Python Code 21

**7. Result and Discussion 23**

7.1. Run the application 23

**8.  Advantages & Disadvantages 24**

**9.  Applications 25**

**10.  Conclusion ………………………………………………………………………...……25**

**Bibilography**

**1. Introduction**

Diabetic Retinopathy (DR) is a common complication of diabetes mellitus, which causes lesions on the retina that affect vision. If it is not detected early, it can lead to blindness. Unfortunately, DR is not a reversible process, and treatment only sustains vision. DR early detection and treatment can significantly reduce the risk of vision loss. The manual diagnosis process of DR retina fundus images by ophthalmologists is time, effort and cost-consuming and prone to misdiagnosis unlike computer-aided diagnosis systems.

**1.1 Overview**

Transfer learning has become one of the most common techniques that has achieved better performance in many areas, especially in medical image analysis and classification. We used Transfer Learning techniques like Inception V3,Resnet50,Xception V3 that are more widely used as a transfer learning method in medical image analysis and they are highly effective. The architecture of CNN Model Deployment is shown in fig. 1.1.

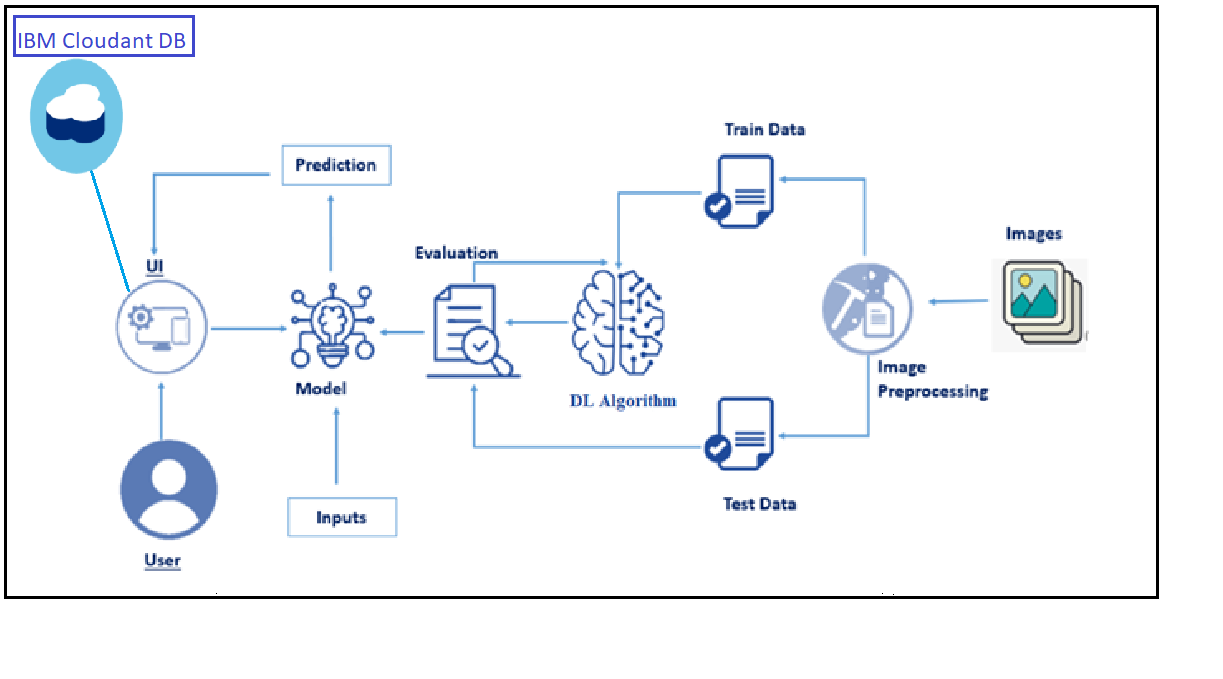


Fig. 1.1. CNN Model Deployment Architecture

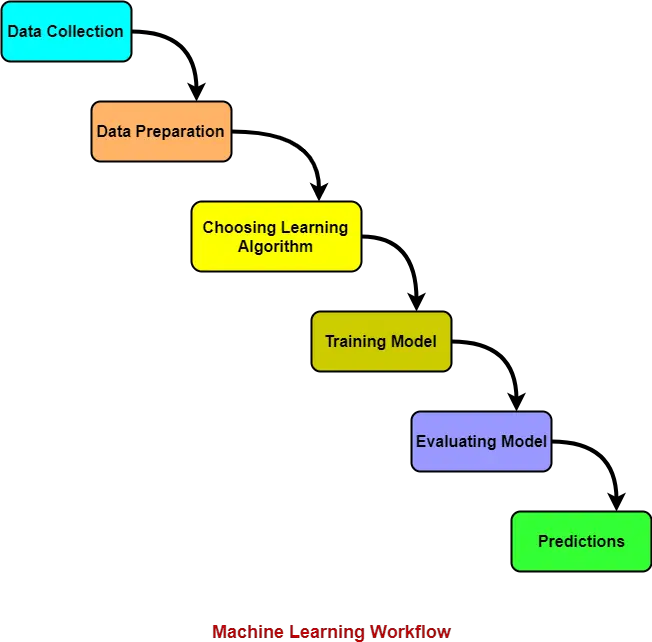
**1.2. Purpose**

Recently, machine learning algorithms have enabled computers to learn from large datasets in a way that exceeds human capabilities in many areas. Several machine learning algorithms with high specificity and sensitivity have been developed for the classification or detection of certain disease conditions based on medical images, including retinal images. Current machine learning systems for DR screening have been predominantly focused on the identification of patients with referable DR (moderate NPDR or worse) or vision-threatening DR, which means the patients should be referred to ophthalmologists for treatment or closer follow-up. However, the importance of identifying early-stage DR should not be neglected. Evidence suggests that proper intervention at an early stage to achieve optimal control of glucose, blood pressure, and lipid profiles could significantlydelay the progression of DR and even reverse mild NPDR to DR-free stage25.

**2.  LITERATURE SURVEY**

Diabetic retinopathy (DR) is a medical condition due to diabetes mellitus that can damage the patient retina and cause blood leaks. This condition can cause different symptoms from mild vision problems to complete blindness if it is not timely treated. Haemorrhages, hard Exudates, and Micro-aneurysms (HEM) that appear in the retina are the early signs of DR. Early diagnosis of HEM is crucial to prevent blindness. Textures features such as LBP have been widely used in the past as a technique for DR detection. In this work, we introduce the use of different texture features for DR, mainly Local Ternary Pattern (LTP) and Local Energy-based Shape Histogram (LESH). We show that they outperform LBP extracted features. Support Vector Machines (SVM) are used for the classification of the extracted histogram.

In addition, the integration of these machine learning advances into DR screening is not straightforward because of some challenges. The entire DR system analytics is shown in fig. 2.1. First, there are a few end-to-end and multi-task learning methods that can share the multi-scale features extracted from convolutional layers for correlated tasks, and further improve the performance of DR grading based on the lesion detection and segmentation, due to the fact that DR grading inherently relies on the global presence and distribution of the DR lesions. Second, despite being helpful in DR screening, there are a few learning methods providing on-site image quality assessment with latency compatible with real-time use, which is one of the most needed additions at primary DR screening level and will have the impact on screening delivery at the community level.

  
Fig. 2.1. Block diagram of DR system analytics

**2.1. Existing problem**

Diabetic Retinopathy (DR) is a complication of diabetes that causes the blood vessels of the retina to swell and to leak fluids and blood. DR can lead to a loss of vision if it is in an advanced stage. Worldwide, DR causes 2.6% of blindness. The possibility of DR presence increases for diabetes patients who suffer from the disease for a long period. Retina regular screening is essential for diabetes patients to diagnose and to treat DR at an early stage to avoid the risk of blindness. DR is detected by the appearance of different types of lesions on a retina image. These lesions are microaneurysms (MA), haemorrhages (HM), soft and hard exudates.

Microaneurysms (MA) is the earliest sign of DR that appears as small red round dots on the retina due to the weakness of the vessel's walls. The size is less than 125 μm and there are sharp margins. Michael et al. [8] classified MA into six types, as shown in Fig. 1. The types of MA were seen with AOSLO reflectance and conventional fluorescein imaging.

Haemorrhages (HM) appear as larger spots on the retina, where its size is greater than 125 μm with an irregular margin. There are two types of HM, which are flame (superficial HM) and blot (deeper HM).

Hard exudates appear as bright-yellow spots on the retina caused by leakage of plasma. They have sharp margins and can be found in the retina's outer layers.

Soft exudates (also called cotton wool) appear as white spots on the retina caused by the swelling of the nerve fiber. The shape is oval or round.

Table 2.1. Type of DR

| **DR Severity Level** | **Lesions** |
| --- | --- |
| No DR | Absent of lesions |
| Mild DR | MA only |
| Moderate DR | More than just MA but less than severe DR |
| Severe DR | Any of the following:  more than 20 intraretinal HM in each of 4 quadrants  definite venous beading in 2+quadrants  Prominent intraretinal microvascular abnormalities in 1+ quadrant  no signs of proliferative DR |
| Proliferative DR | One or more of the following: vitreous/pre-retinal HM, neovascularization |

The automated methods for DR detection are cost and time saving and are more efficient than a manual diagnosis. A manual diagnosis is prone to misdiagnosis and requires more effort than automatic methods. This project train the CNN model on recent DR dataset to automate the method of detection and classification of DR.

**2.2. Pre Requisite**

To complete this project, it require the following software’s, concepts and packages:

1. Anaconda navigator and Spyder:

Some of the Python packages required are as follows:

Open anaconda prompt as administrator

1. Type “pip install numpy” and click enter.
2. Type “pip install pandas” and click enter..
3. Type “pip install tensorflow==2.3.2” and click enter.
4. Type “pip install keras==2.3.1” and click enter.
5. Type “pip install Flask” and click enter.

**2.3. Proposed solution**

The above problem could be solved by developing a project on Diabetic Retinopathy by using Deep Learning model i.e. CNN. The Convolutional neural networks are regularized versions of multilayer perceptron (MLP). They were developed based on the working of the neurons of the animal visual cortex. Let’s say there is a color image in JPG form and its size is 480 x 480. The representative array will be 480 x 480 x 3. Each of these numbers is given a value from 0 to 255 which describes the pixel intensity at that point. RGB intensity values of the image are visualized by the computer for processing.

The idea is that given the computer this array of numbers and it will output numbers that describe the probability of the image being a certain class (.80 for a cat, .15 for a dog, .05 for a bird, etc.). It works similar to how our brain works. When we look at a picture of a dog, we can classify it as such if the picture has identifiable features such as paws or 4 legs. In a similar way, the computer is able to perform image classification by looking for low-level features such as edges and curves and then building up to more abstract concepts through a series of convolutional layers. The computer uses low-level features obtained at the initial levels to generate high-level features such as paws or eyes to identify the object. Contents of a classic Convolutional Neural Network: -

1. Convolutional Layer.
2. Activation operation following each convolutional layer.
3. Pooling layer especially Max Pooling layer and also others based on the requirement.
4. Finally Fully Connected Layer.

The training process is also known as backpropagation, which is further separated into 4 distinct sections or processes.

1. Forward Pass
2. Loss Function
3. Backward Pass
4. Weight Update

In order to accomplish the above project objectives, certain activities are required to complete. Firstly, the user interacts with the UI (User Interface) to choose the image and then the chosen image is 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.

The entire project flow is divided into below activities and tasks listed below:

1. Data Collection.
   * Create a Train and Test path.
2. 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

**3.   Diabetic Retinopathy Level Detection Model**

**3.1. Dataset Download**

There are many popular open sources for collecting the data. Eg: kaggle.com, UCI repository, etc. The training model is built on Google colab and IBM Watson studio. In this project, the dataset is uploaded on colab/IBM Watson studio directly by using Kaggle API. Dataset is cloned from kaggle data repository to colab/IBM Watson studio by running below commands on the python notebook.

* Clone kaggle in google colab as shown in fig. 3.1.

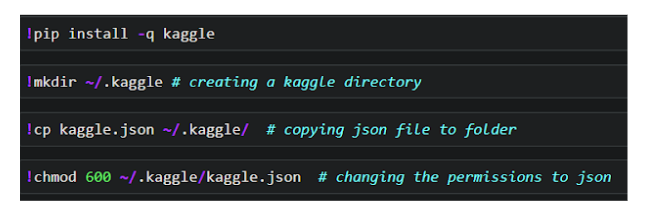


Fig. 3.1. Kaggle library installation

* Download the dataset by running the command shown in fig. 3.2.

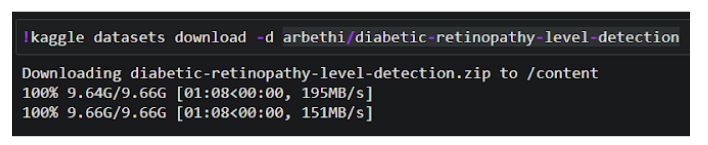


Fig. 3.2. Kaggle dataset download

* Unzip the dataset as shown in fig. 3.3.

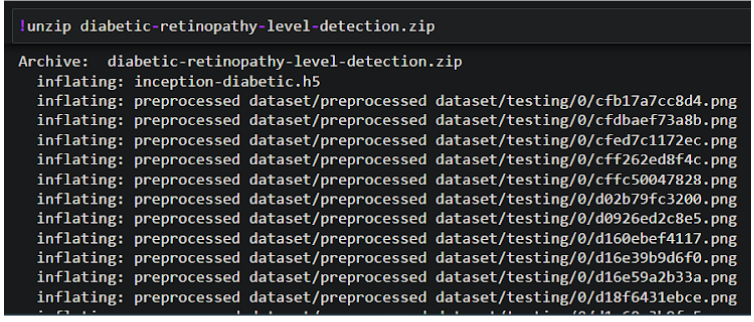


Fig. 3.3. Unzipping the Kaggle dataset

### 3.2. 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 initialized to 299, 299 as shown in fig. 3.4.

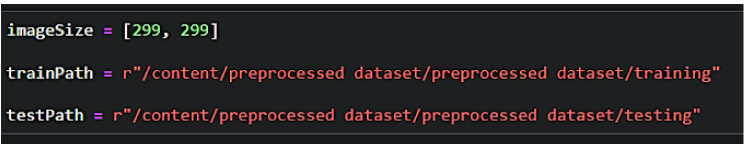


Fig. 3.4 Image size and dataset path setting

### 3.3. Importing The Libraries

Import the necessary libraries as shown in the fig. 3.5.

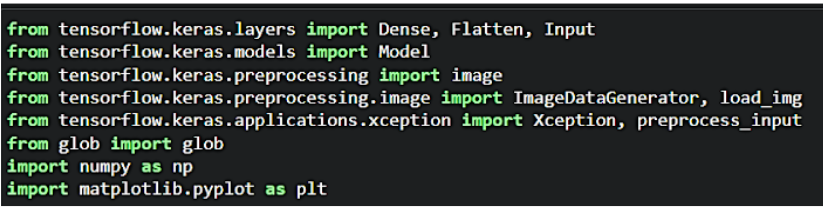


Fig. 3.5 Importing Python Libraries

### 3.4. 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 as shown in fig. 3.6.

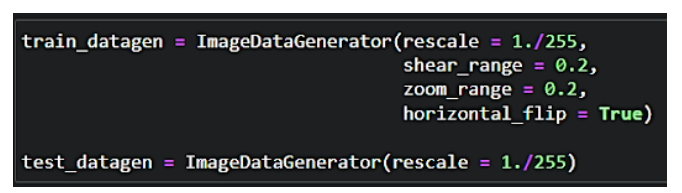


Fig. 3.6 Image Data Generator

### 3.5. 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 as shown in fig. 3.7. 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).

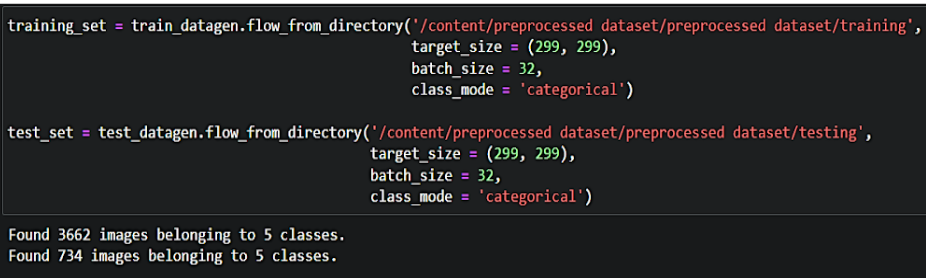


Fig. 3.7 ImageDataGenerator applied on training and test data set.

### 4. Model Building

### 4.1. 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, it has considered the images of dimension (229,229,3).

Also, we have assigned include\_top = False because we are using convolution layer for features extraction and wants to train fully connected layer for our images classification(since it is not the part of Imagenet dataset). Flatten layer flattens the input as shown in fig. 4.1. Does not affect the batch size.

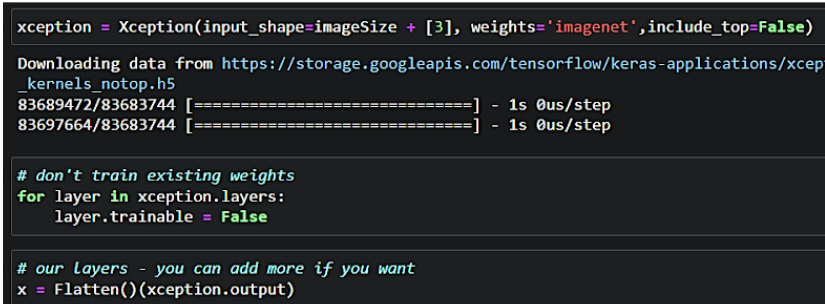


Fig. 4.1. Flattening of layer

### 4.2. 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 as shown in fig. 4.2.

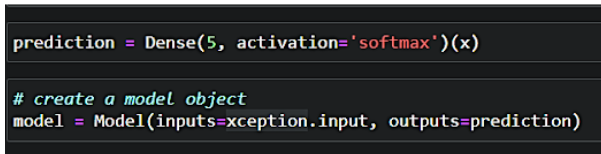


Fig. 4.2. Apply Model function

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 use it for training and prediction purposes. Keras provides a simple method, summary to get the full information about the model and its layers as shown in fig. 4.3 and fig. 4.4.

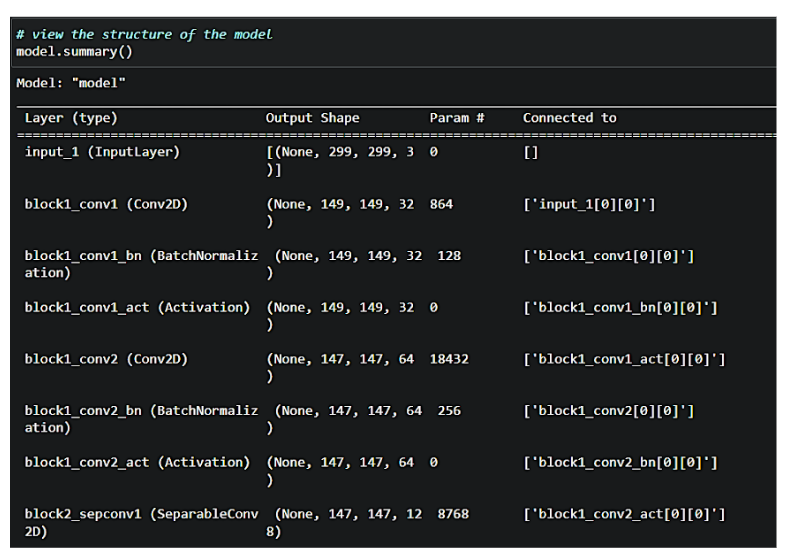


Fig. 4.3. Model Summary

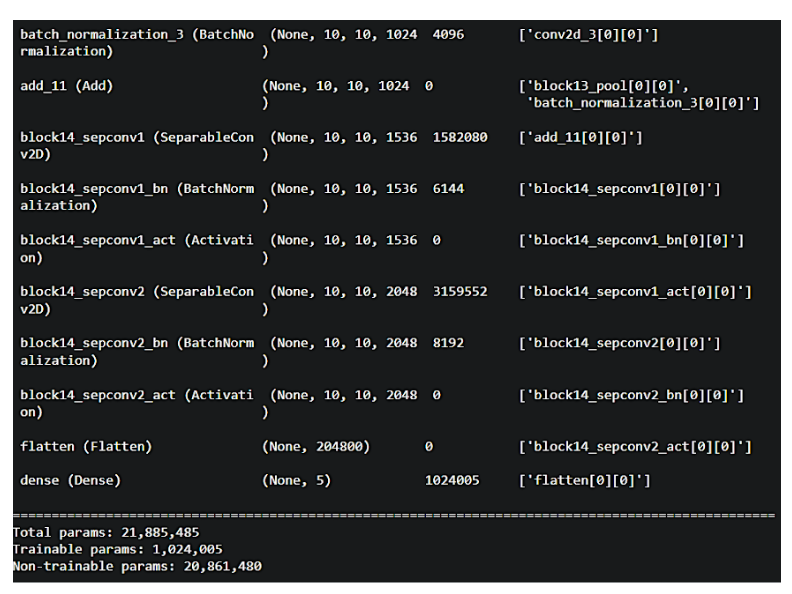


Fig. 4.4. Model Summary summarized

### 4.3. 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 as shown in fig. 4.5.

Metrics are used to evaluate the performance of your model. It is similar to the loss function, but not used in the training process

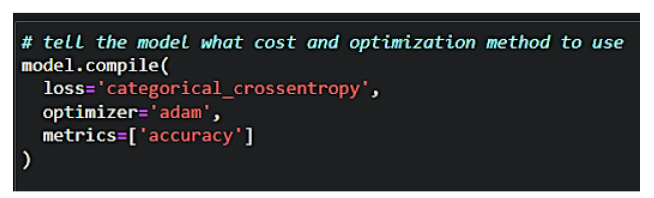


Fig. 4.5. Model Optimization

### 4.4. 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 as shown in fig. 4.6.

**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

                      - an 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.

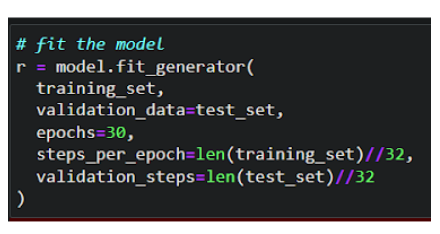


Fig. 4.6. Model fit generator

The execution of model and its corresponding accuracy is shown in fig. 4.7

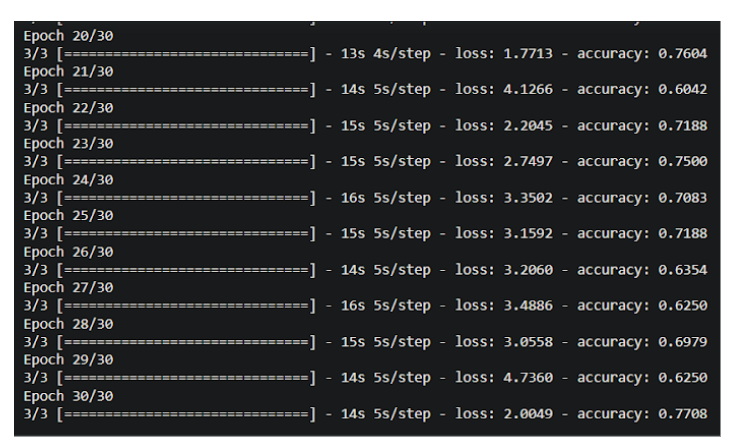
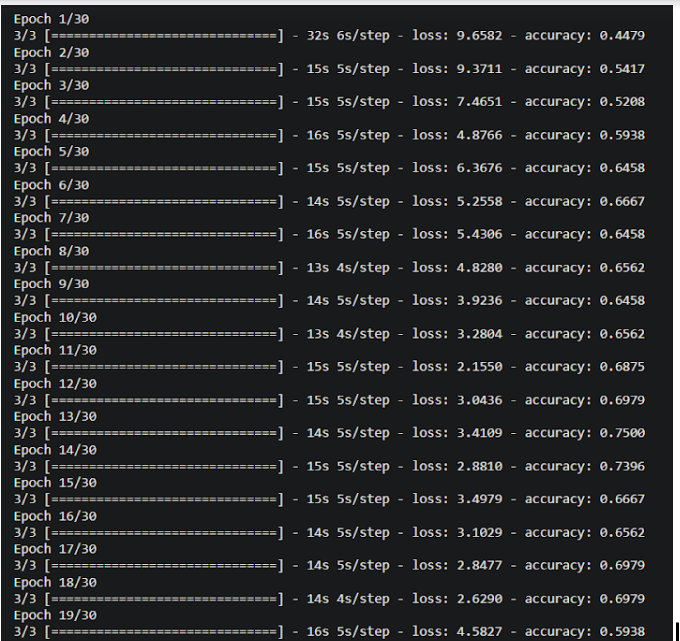


Fig. 4.7. Model accuracy

### 4.5. Save The Model

The model is saved with .h5 extension as shown in the below fig. 4.8.

An H5 file is a data file saved in the Hierarchical Data Format (HDF). It contains multidimensional arrays of scientific data.

****

Fig. 4.8 Save Model

### 5. IBM Cloud Access

### Watson Studio democratizes data science and AI to drive innovation in your business. With a suite of tools for all skill levels, everyone can collaborate to prepare, analyze, and model data. You can write Python or R code in notebooks, visually code on a graphical canvas, or automatically build models. There are some features and capabilities of IBM Watson cloud as mentioned below:

### Speed AI development with AutoAI: With AutoAI, beginners can build models without coding and expert data scientists can speed experimentation in AI development. AutoAI automates data preparation, model development, feature engineering, and hyperparameter optimization.

### Optimize decisions: Decision Optimization streamlines the selection and deployment of prescriptive models. Write code in Python or OPL, or in natural language expressions with the intelligent Modelling Assistant. Investigate and compare solutions for multiple scenarios with dashboards.

### Develop models visually: With the IBM SPSS Modeler graphical canvas, create a flow of steps to prepare data and build predictive models. Drag and drop from over 100 data preparation, data visualization, statistical analysis, predictive modelling, and text analytics nodes.

### Federated model training: With Federated Learning, train a model on a set of federated data sources while maintaining data security. Each participating party in the federation trains the common machine learning model without moving or sharing data. The training results are fused into a more accurate model.

### 5.1. Register & Login To IBM Cloud

### Create Service Instance

* Log in to your IBM Cloud account, and click on Catalog as shown in fig. 5.1.

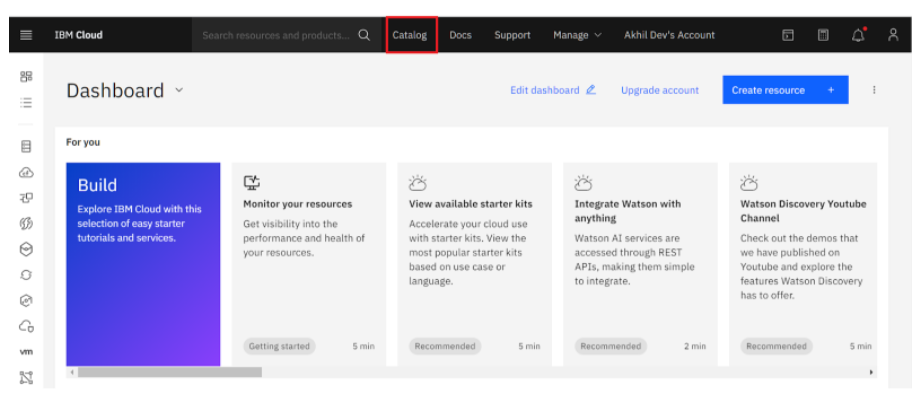


Fig. 5.1. IBM cloud account

* Type Cloudant in the Search bar and click to open it as shown in fig. 5.2.

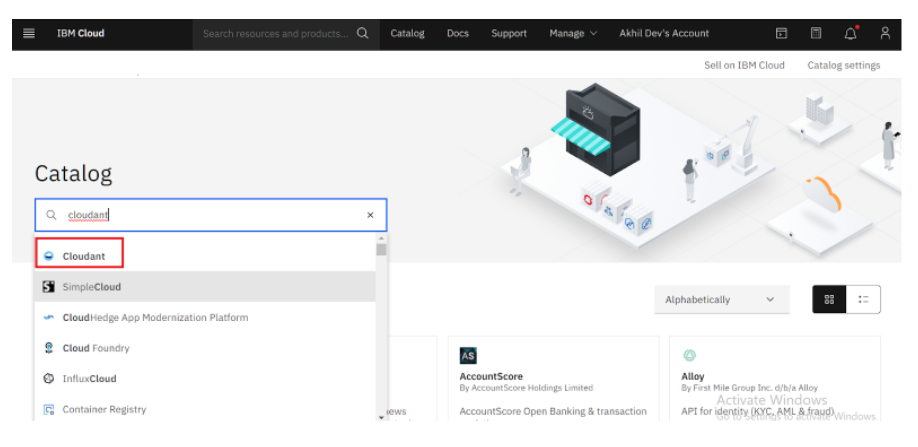


Fig. 5.2 Search in Cloudant Catalog

* Select an offering and an environment

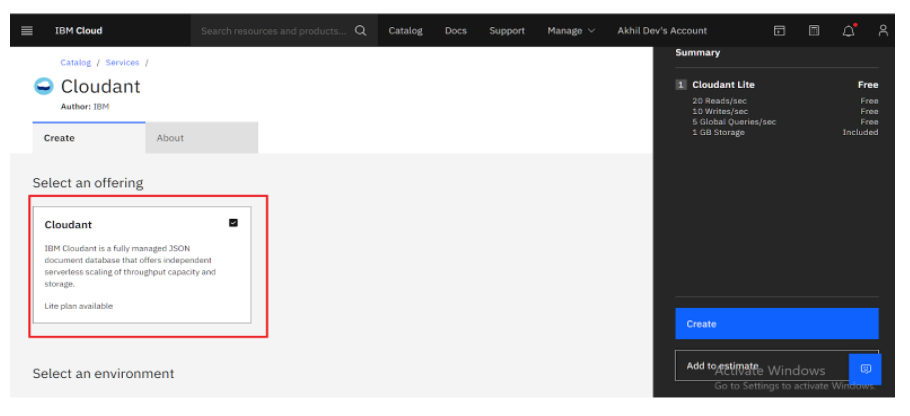


Fig. 5.3 Select Environment

* Select region as Dallas & Type an instance name then click on create service as shown in fig. 5.4.

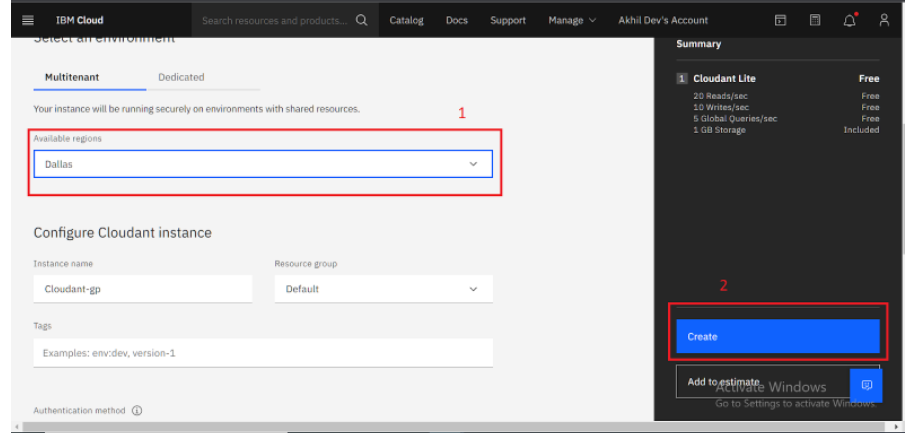


Fig. 5.4 Select Region and click on create

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.

### 5.2. Creating Service Credentials

1. To create the connection information that your application needs to connect to the instance, click New credential as shown in fig. 5.5.

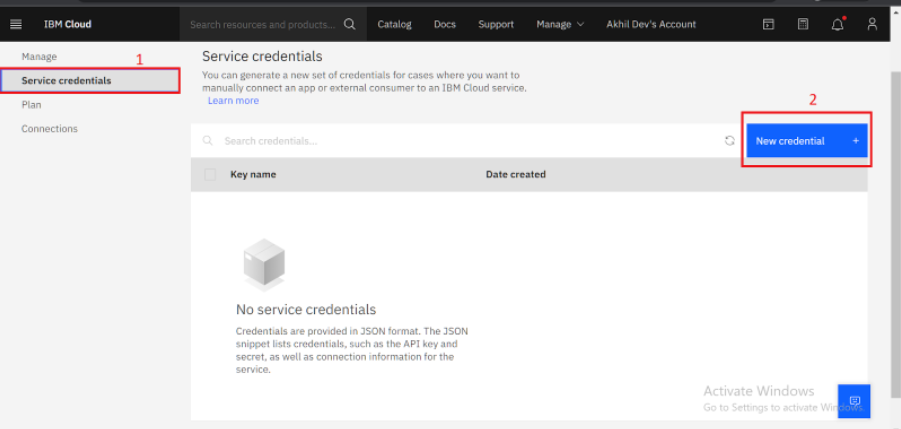


Fig. 5.5 Connection creation

1. Enter a name for the new credential in the Add new credential window.
2. Accept the Manager role.
3. Create a service ID or have one automatically generated for you.
4. Add inline configuration parameters. This parameter isn't used by IBM Cloudant service credentials, so ignore it. Click Add as shown in fig. 5.6.

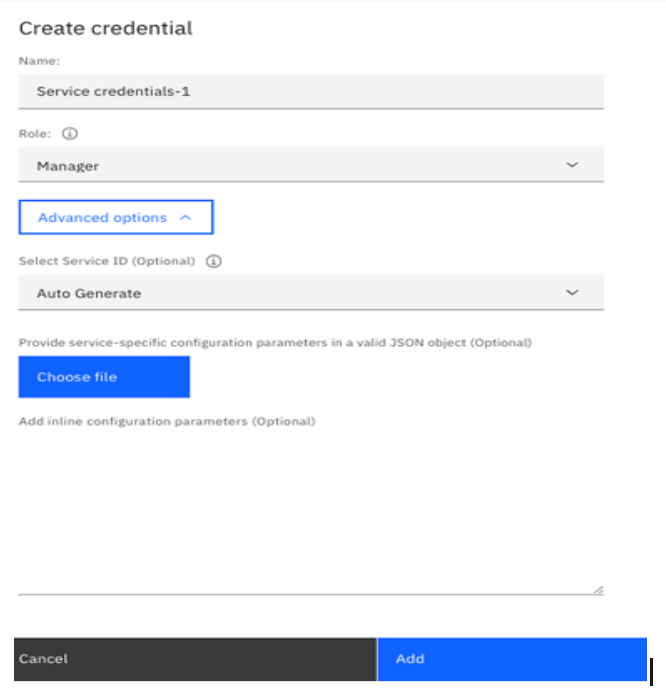


Fig. 5.6 Account credentials

### 5.3. Launch Cloudant DB

Launch the Cloudant DB by clicking on the Launch Dashboard as shown in fig. 5.7.

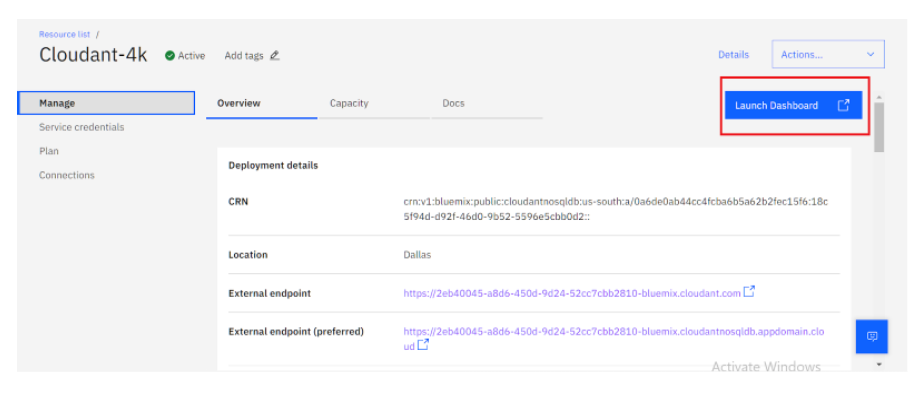


Fig. 5.7. Launch Cloudant DB

Note: If it is a New User then it will show empty database otherwise it will list already existing DB as shown in fig. 5.8.

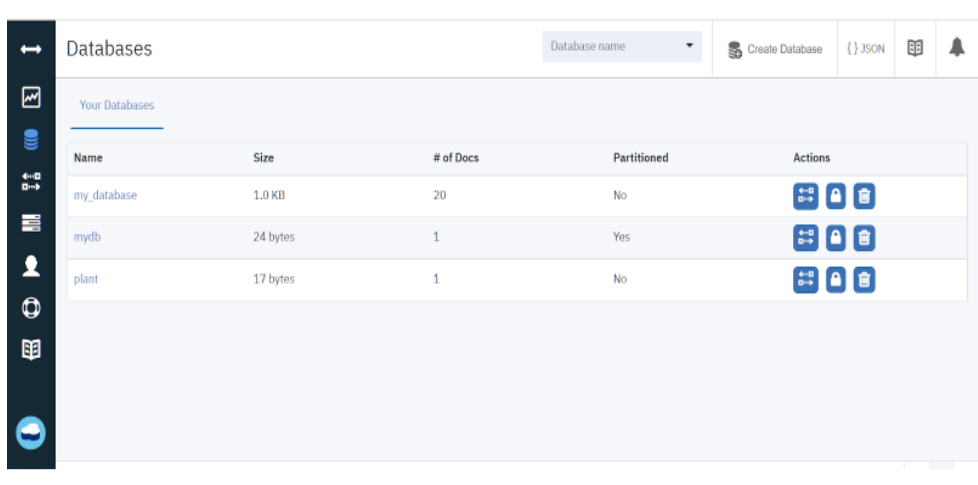
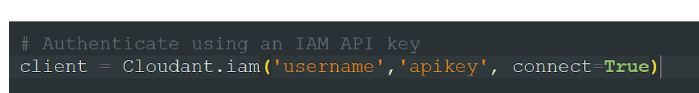


Fig. 5.8 Home page of Cloudant DB

### 5.4. Create Database

In order to manage a connection from a local system, it 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. In the above cloudant.iam() method we have to give  username & apikey to build the connection with cloudant DB as shown in fig. 5.9.

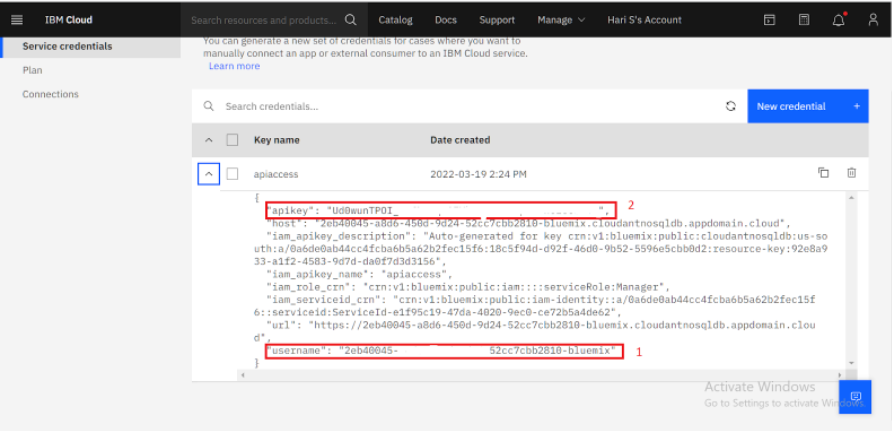
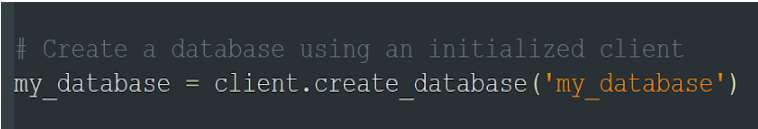


Fig. 5.9. Access the Username and API key

Once a connection is established you can then create a database, open an existing database. Create a database as my\_database.



### 6 Application Building

In this section, a web application is built that is integrated to the model built earlier. A UI is provided to the user where he can upload 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

### 6.1. Building Html Pages

For this project create one HTML file namely, index.html and save them in the templates folder. The outcome index.html page look is shown in fig. 6.1.

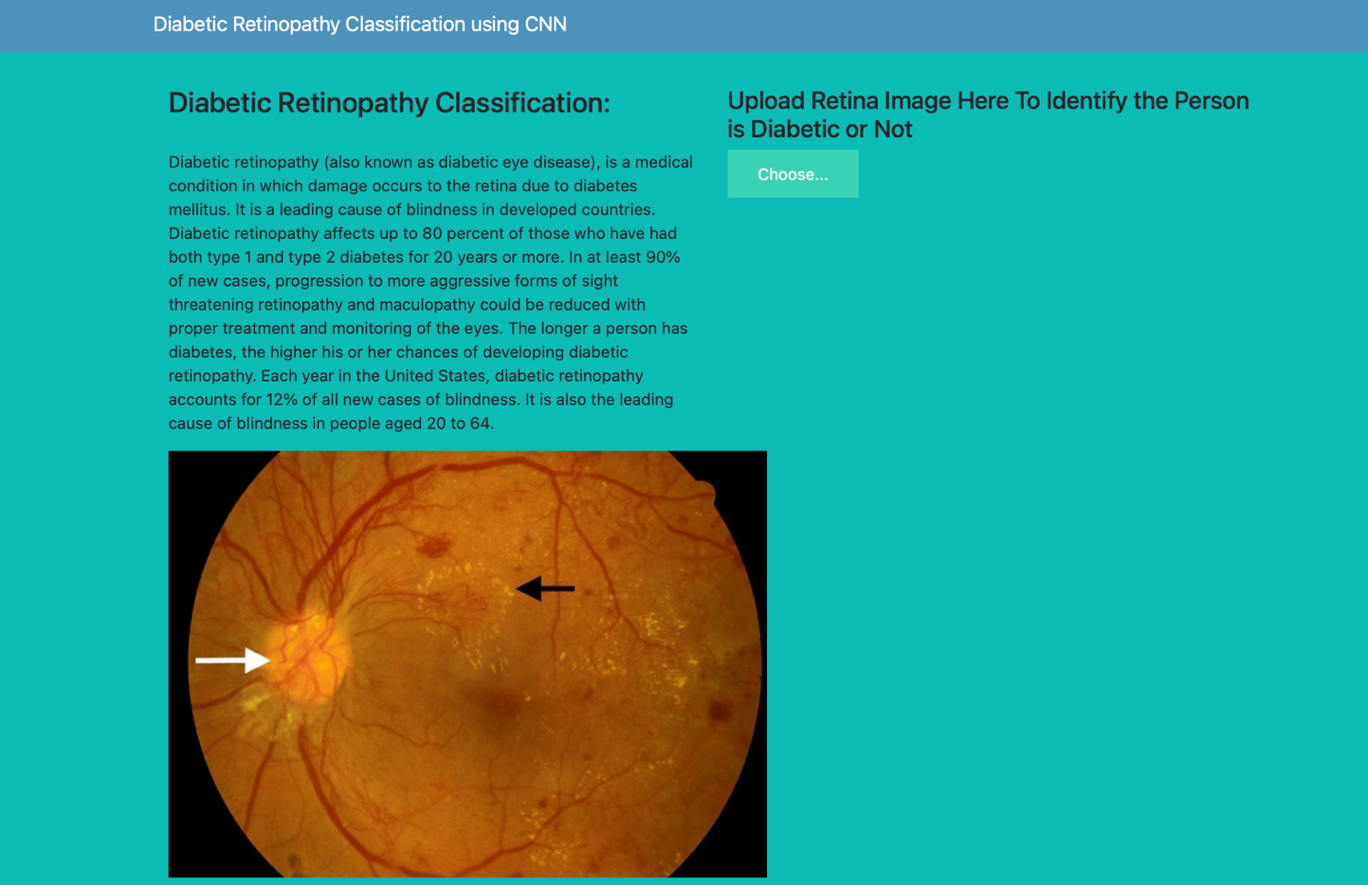


Fig. 6.1. Home page of Diabetic Retinopathy Classification Webpage

Now click on choose button to select the image to predict the type of diabetic condition as shown in fig. 6.2.

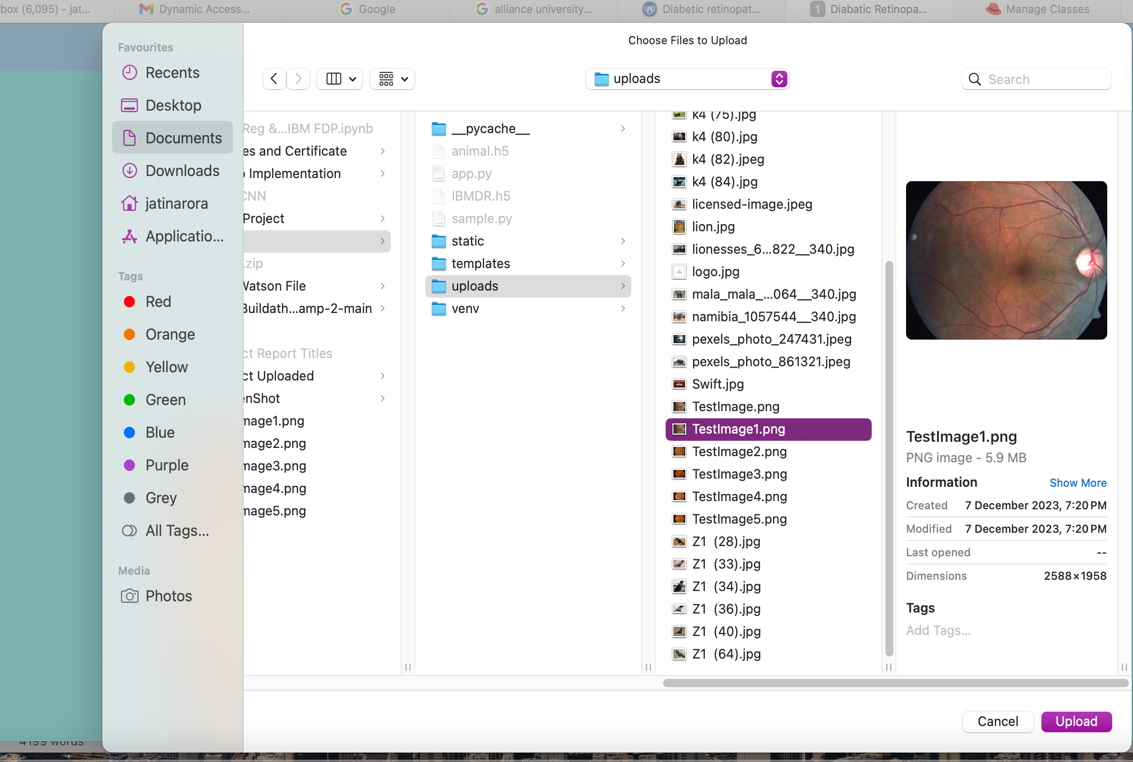


Fig. 6.2 Select an image to predict the type of diabetic condition.

After the successful upload of the image, press the prediction button to print the class of diabetes of the concerned person. The prediction button is shown below the image as shown in fig. 6.3.

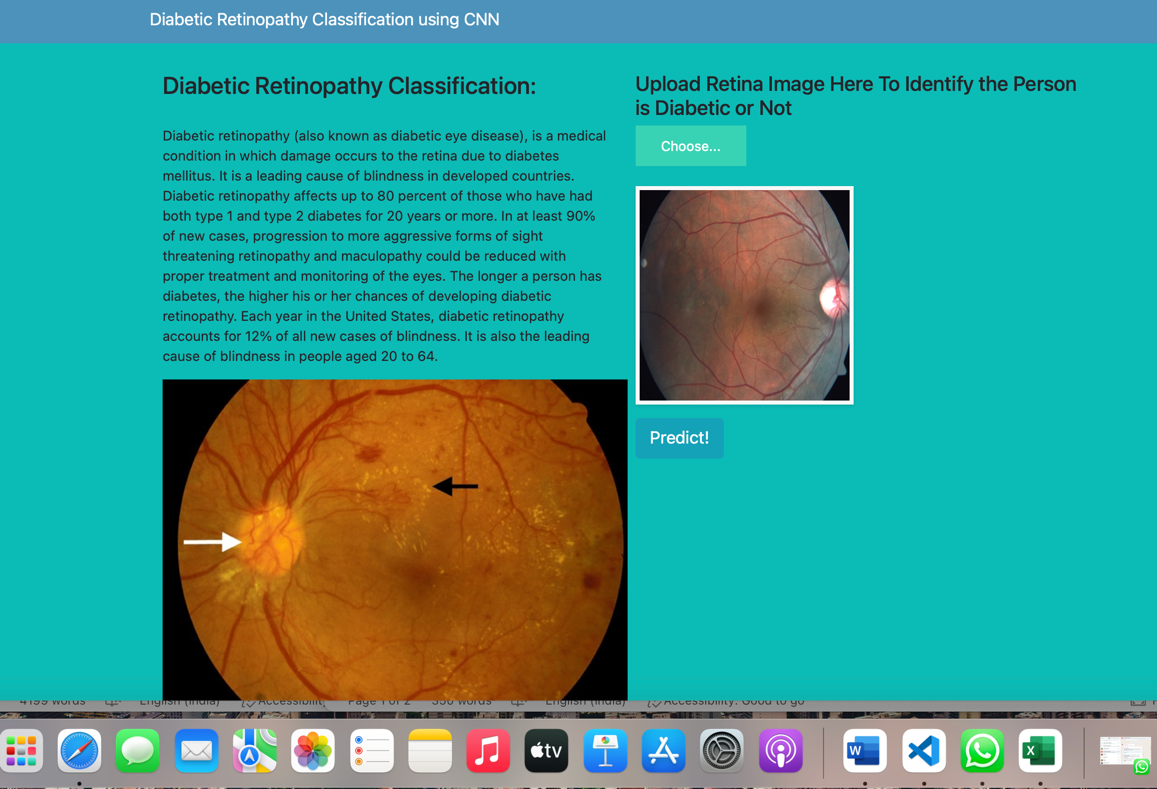


Fig. 6.3 Prediction button to print the result.

### 6.2. Build Python Code

Import the required libraries for running the project. 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.

A screenshot of a computer

Description automatically generated

Fig. 6.4. Building python code in Visual studio code

Here we will be using a declared constructor to route to the HTML page which we have created earlier as shown in fig. 6.4.

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.

**Showcasing prediction on UI:**

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:**



### 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.

###### **7. Result and Discussion**

###### **7.1. 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.

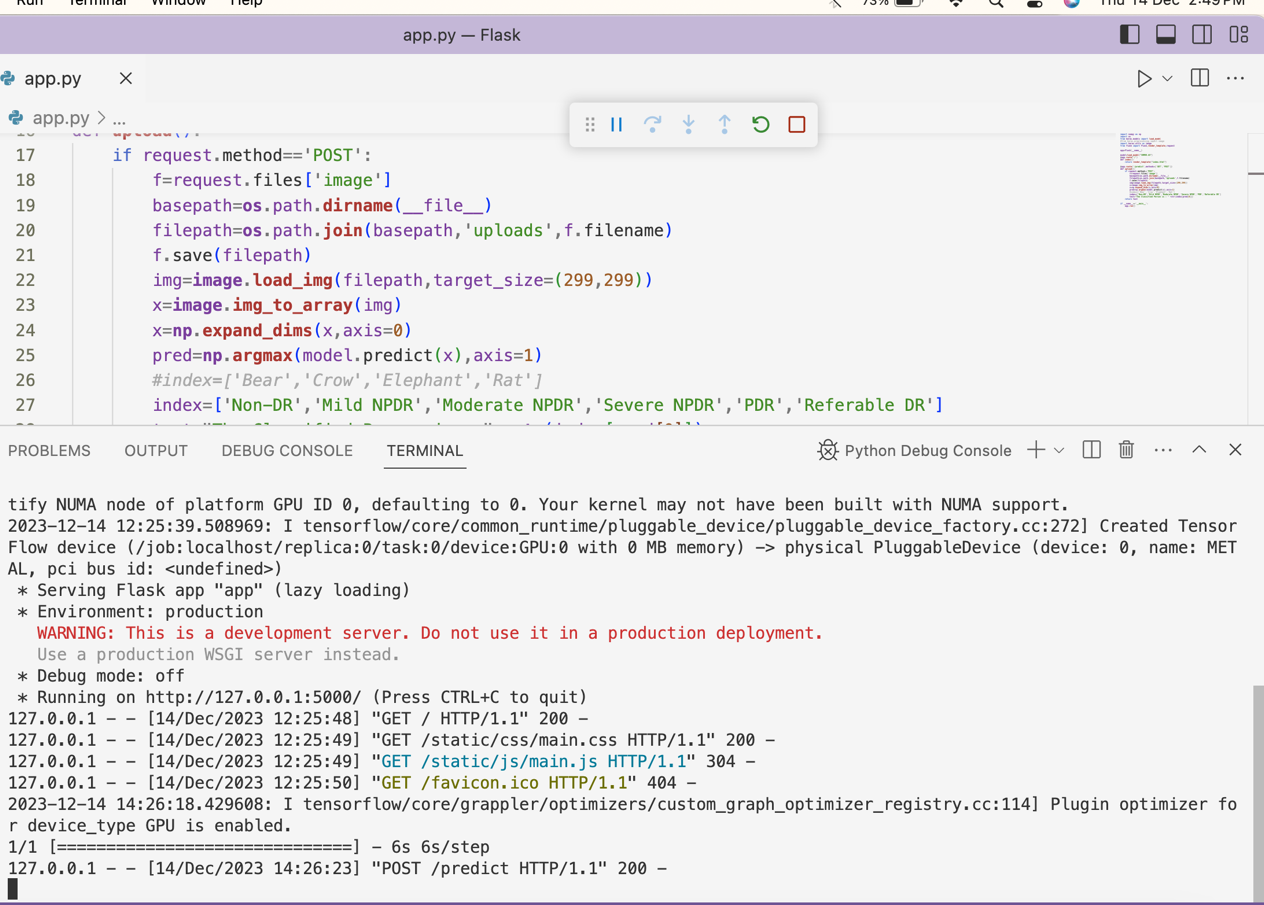


Fig. 7.1. Running the flask application.

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

The final home page displayed after building python code and running the flask application on visual code is shown in fig. 7.2.

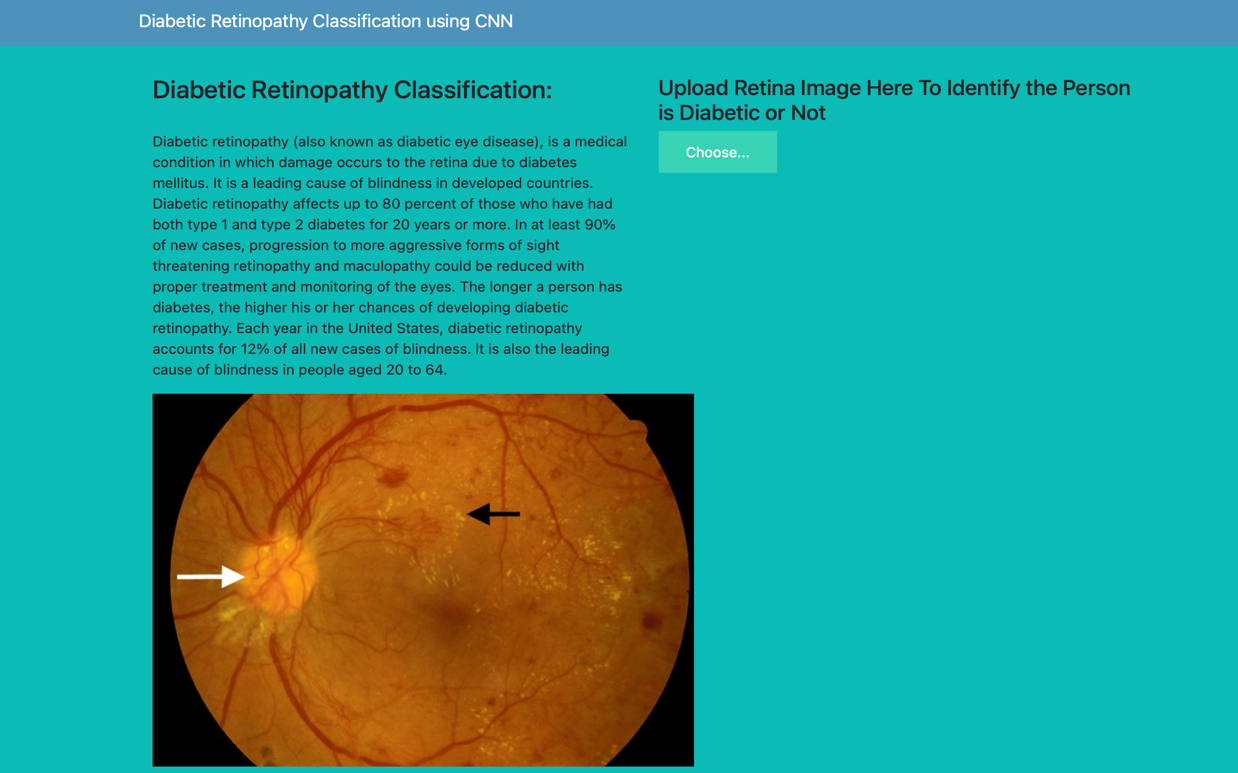


Fig. 7.2. Select an image to predict the class of DR.

After successfully loading of the prediction page where it is showing choose button to upload the image to predict the outcomes as shown in fig. 7.3.

A screenshot of a computer screen

Description automatically generated

Fig. 7.3 Predict the outcome of selected image

**8.  ADVANTAGES & DISADVANTAGES**

For image recognition, image classification and computer vision (CV) applications, CNNs are particularly useful because they provide highly accurate results, especially when a lot of data is involved. The CNN also learns the object's features in successive iterations as the object data moves through the CNN's many layers. This direct (and deep) learning eliminates the need for manual feature extraction (feature engineering).

CNNs can be retrained for new recognition tasks and built on preexisting networks. These advantages open up new opportunities to use CNNs for real-world applications without increasing computational complexities or costs.

The downside of CNN is that it requires large amounts of data. High computational cost causes hard to optimize due to large parameter size.

**9.  APPLICATIONS**

The most common applications of above project are as follows:

**Healthcare application:** It can examine thousands of visual reports to detect any anomalous conditions in patients, such as type of Diabetes.

**Automotive:** It is powering research into autonomous health care system.

**E-medical**: It provides platform that incorporate visual search allow smart system to recommend medication.

**Facial recognition:** Generative adversarial networks (GANs) are used to produce new images that can then be used to train deep learning models for DR recognition.

**AutoML processing of CNNs**: In autoML assistants learn and process the model is updated automatically and predict mode accurate results.

**10.  CONCLUSION**

DR in diabetics prevents the worsening of DR. However, the CNN classification model should be used as supplements to a healthy and balanced health and prevents all the complications of diabetes. In conclusion, the data presented in this model strongly gives 80% accuracy to identify the classification of DR thus improving the quality of life of millions of diabetics. The

**BIBILOGRAPHY**

1. www.embibe.com
2. www.slideahare.com
3. www.researchgaresearchgate.com
4. https://towardsdatascience.com/basics-of-the-classic-cnn-a3dce1225add
5. https://towardsdatascience.com/understanding-and-coding-a-resnet-in-keras-446d7ff84d33
6. https://iq.opengenus.org/inception-v3-model-architecture/
7. https://pyimagesearch.com/2017/03/20/imagenet-vggnet-resnet-inception-xception-keras/
8. Flask Link: https://www.youtube.com/watch?v=lj4I\_CvBnt0