

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
Dr. Yakuta Tayyebi
Prestige Institute of Engineering Management and Research
Indore (M.P)

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	8
3.1. Dataset Download.....	8
3.2. Create Training And Testing Path.....	9
3.3. Importing The Libraries	9
3.4. Configure ImageDataGenerator Class.....	10
3.5. Apply ImageDataGenerator Functionality To Train Set And Test Set	10
4. Model Building	11
4.1. Pre-Trained CNN Model As A Feature Extractor	11
4.2. Adding Dense Layers	11
4.3. Configure The Learning Process.....	13
4.4. Train The Model	14
4.5. Save The Model.....	16
5. IBM Cloud Access	16
5.1. Register & Login To IBM Cloud.....	16
5.2. Creating Service Credentials	18
5.3. Launch Cloudbant DB.....	19
5.4. Create Database.....	20
6 Application Building	21
6.1. Building Html Pages	21
6.2. Build Python Code.....	22
7. Result and Discussion	24
Run the application	24
8. ADVANTAGES & DISADVANTAGES	25
9. APPLICATIONS	26
10. CONCLUSION	26
BIBLIOGRAPHY	

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 have achieved better performance in many areas, especially in medical image analysis and classification. We used Transfer Learning techniques like Inception V3, Resnet50, and Xception V3 which are more widely used as transfer learning methods in medical image analysis and they are highly effective. The architecture of CNN Model Deployment is shown in Fig. 1.1.

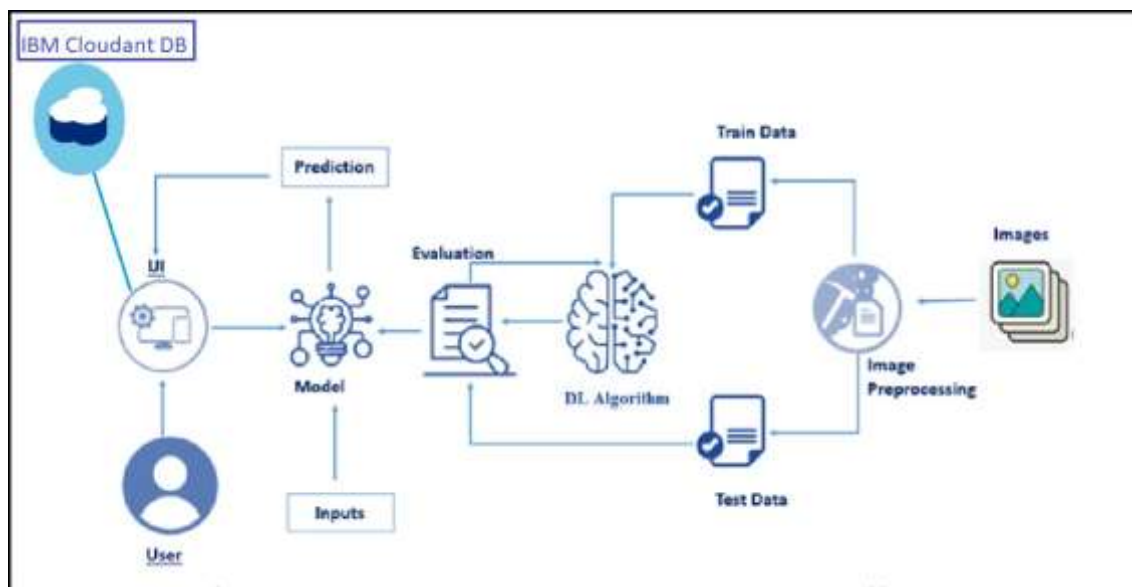


Fig. 1.1. CNN Model Deployment Architecture

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 significantly delay the progression of DR and even reverse mild NPDR to the DR-free stage²⁵.

2. LITERATURE SURVEY

Diabetic retinopathy (DR) is a medical condition due to diabetes mellitus that can damage the patient's retina and cause blood leaks. This condition can cause different symptoms from mild vision problems to complete blindness if it is not timely treated. Hemorrhages, 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. Texture 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 transfer learning 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 the primary DR screening level and will have an impact on screening delivery at the community level.

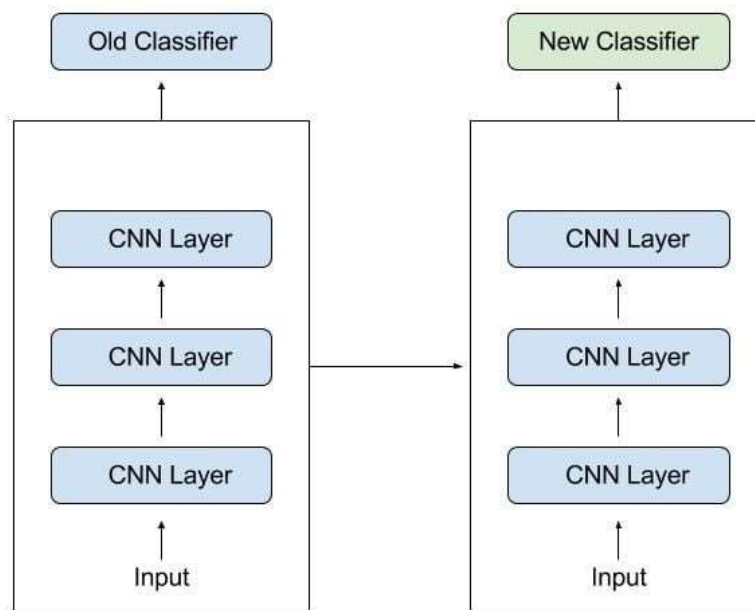


Fig. 2.1. Block diagram of transfer learning system

2.1. Existing problem

Diabetic Retinopathy (DR) is a complication of diabetes that causes the blood vessels of the retina to swell and 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 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), hemorrhages (HM), and 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.

Hemorrhages (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: <ul style="list-style-type: none"> • 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 trains the CNN model on a recent DR dataset to automate the method of detection and classification of DR.

2.2. Pre Requisite

To complete this project, it requires 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 the Deep Learning model i.e. transfer learning technique. Transfer learning has become one of the most common techniques that have achieved better performance in many areas, especially in medical image analysis and classification. We used Transfer Learning techniques like Inception V3, Resnet50, and Xception V3 which are more widely used as transfer learning methods in medical image analysis and they are highly effective. 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.

With transfer learning, it is basically used to exploit what has been learned in one task to improve generalization in another. It transfers the weights that a network has learned at "task A" to a new "task B." The general idea is to use the knowledge a model has learned from a task with a lot of available labeled training data in a new task that doesn't have much data. Instead of starting the learning process from scratch, transfer learning starts with patterns learned from solving a related task. Transfer learning is mostly used in computer vision and natural language processing tasks like sentiment analysis due to the huge amount of computational power required. Transfer learning tries to transfer as much knowledge as possible from the previous task the model was trained on to the new task at hand. This knowledge can be in various forms depending on the problem and the data. For example, it could be how models are composed, which allows us to more easily identify novel objects.

Usually, a lot of data is needed to train a neural network from scratch but access to that data isn't always available — this is where transfer learning comes in handy. With transfer learning a solid machine learning model can be built with comparatively little training data because the model is already pre-trained. This is especially valuable in natural language processing because mostly expert knowledge is required to create large labelled data sets. Additionally, training time is reduced because it can sometimes take days or even weeks to train a deep neural network from scratch on a complex task.

If the original model was trained using an open-source library like TensorFlow, it can be simply restored and retrain some layers for another task. Transfer learning only works if the features learned from the first task are general, meaning they can be useful for another related task as well. Also, the input of the model needs to be the same size as it was initially trained with. If you don't have that, add a pre-processing step to resize the input to the needed size. There are three approaches to Transfer Learning as listed below:

1. **Training A Model To Reuse it:** Imagine task A is required to solve but enough data is not available to train a deep neural network. One way around this is to find a related task B with an abundance of data. Train the deep neural network on task B and use the model as a starting point for solving task A. If the same input is fed into both tasks, possibly reusing the model and making predictions for new input is an option.
2. **Using A Pre-Trained Model:** The second approach is to use an already pre-trained model. There are a lot of these models out there, so make sure to do a little research. How many layers to reuse and how many to retrain depends on the problem. Keras, for example, provides numerous pre-trained models that can be used for transfer learning, prediction, feature extraction, and fine-tuning. This type of transfer learning is most commonly used throughout deep learning.
3. **Feature Extraction:** Another approach is to use deep learning to discover the best representation of the problem, which means finding the most important features. This approach is also known as representation learning, and can often result in a much better performance than can be obtained with hand-designed representation.

The idea of learning is to give the computer an 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 similarly to how the human brain works. While looking at a picture of a dog, a person 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 be completed. 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 the 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
 - Apply ImageDataGenerator 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. The dataset is cloned from the Kaggle data repository to Colab/IBM Watson studio by running the below commands on the python notebook.

- Clone kaggle in google colab as shown in Fig. 3.1.

```
!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
```

Fig. 3.1. Kaggle library installation

- Download the dataset by running the command shown in Fig. 3.2.


```
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]
```

Fig. 3.2. Kaggle dataset download

- Unzip the dataset as shown in Fig. 3.3.

```
unzip diabetic-retinopathy-level-detection.zip

Archive: diabetic-retinopathy-level-detection.zip
  inflating: inception-diabetic.h5
  inflating: preprocessed dataset/preprocessed dataset/testing/0/cfb17a7cc8d4.png
  inflating: preprocessed dataset/preprocessed dataset/testing/0/cfdbaef73a8b.png
  inflating: preprocessed dataset/preprocessed dataset/testing/0/cfed7c1172ec.png
  inflating: preprocessed dataset/preprocessed dataset/testing/0/cff262ed8f4c.png
  inflating: preprocessed dataset/preprocessed dataset/testing/0/cffc50047828.png
  inflating: preprocessed dataset/preprocessed dataset/testing/0/d02b79fc3200.png
  inflating: preprocessed dataset/preprocessed dataset/testing/0/d0926ed2c8e5.png
  inflating: preprocessed dataset/preprocessed dataset/testing/0/d160ebef4117.png
  inflating: preprocessed dataset/preprocessed dataset/testing/0/d16e39b9d6f0.png
```

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, create a variable and assign a value of 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 the xception model is initialized to 299, 299 as shown in Fig. 3.4.

```
imageSize = [299, 299]

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

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.

```
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
```

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.

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

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).

```
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.
```

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.

```
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)
```

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.

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

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

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.

```
# 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 (BatchNormalization)	(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 (BatchNormalization)	(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 (SeparableConv2D)	(None, 147, 147, 128)	8768	['block1_conv2_act[0][0]']

Fig. 4.3. Model Summary

batch_normalization_3 (Batch Normalization)	(None, 10, 10, 1024)	4096	['conv2d_3[0][0]']
add_11 (Add)	(None, 10, 10, 1024)	0	['block13_pool[0][0]', 'batch_normalization_3[0][0]']
block14_sepconv1 (Separable Conv2D)	(None, 10, 10, 1536)	1582080	['add_11[0][0]']
block14_sepconv1_bn (Batch Normalization)	(None, 10, 10, 1536)	6144	['block14_sepconv1[0][0]']
block14_sepconv1_act (Activation)	(None, 10, 10, 1536)	0	['block14_sepconv1_bn[0][0]']
block14_sepconv2 (Separable Conv2D)	(None, 10, 10, 2048)	3159552	['block14_sepconv1_act[0][0]']
block14_sepconv2_bn (Batch Normalization)	(None, 10, 10, 2048)	8192	['block14_sepconv2[0][0]']
block14_sepconv2_act (Activation)	(None, 10, 10, 2048)	0	['block14_sepconv2_bn[0][0]']
flatten (Flatten)	(None, 204800)	0	['block14_sepconv2_act[0][0]']
dense (Dense)	(None, 5)	1024005	['flatten[0][0]']

Total params: 21,885,485
 Trainable params: 1,024,005
 Non-trainable params: 20,861,480

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

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

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 used to train the model.
- **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.

```
# 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
)
```

Fig. 4.6. Model fit generator

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

```

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

```

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.

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

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:

1. **Speed AI development with AutoAI:** With AutoAI, beginners can build models without coding, and expert data scientists can speed up experimentation in AI development. AutoAI automates data preparation, model development, feature engineering, and hyperparameter optimization.
2. **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 Modeling Assistant. Investigate and compare solutions for multiple scenarios with dashboards.
3. **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.
4. **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.

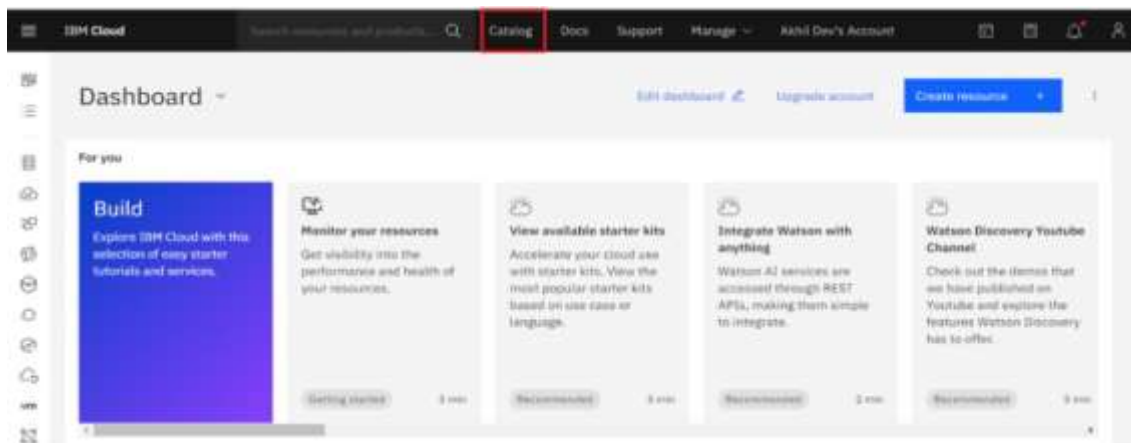


Fig. 5.1. IBM cloud account

- Type Cloudant in the Search bar and click to open it as shown in Fig. 5.2.

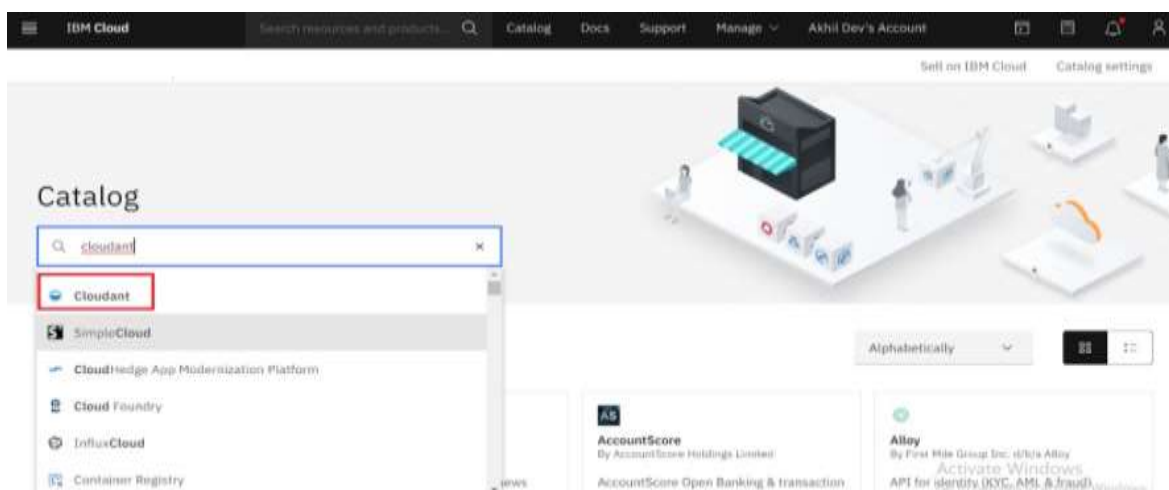


Fig. 5.2 Search in Cloudant Catalog

- Select an offering and an environment as shown in Fig. 5.3.

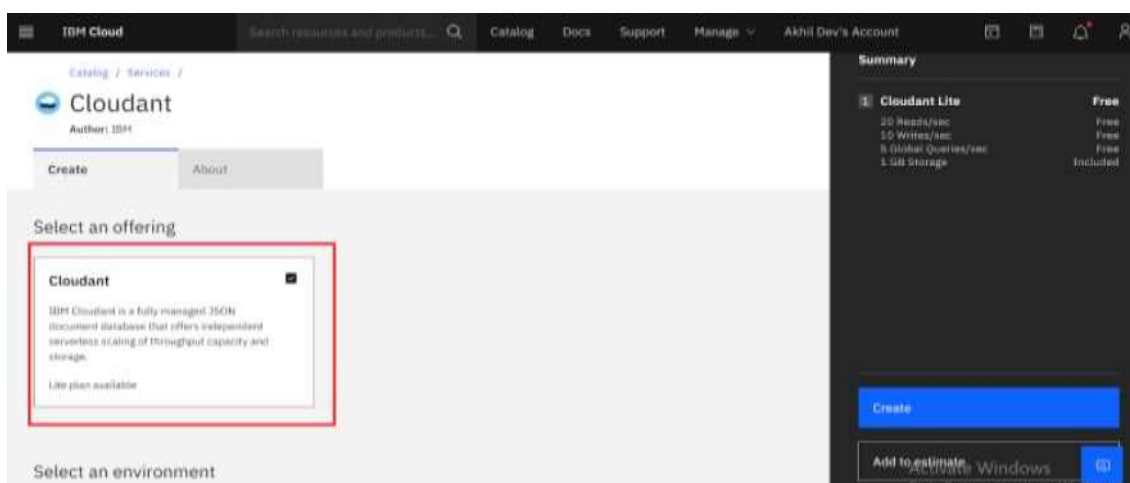


Fig. 5.3 Select Environment

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

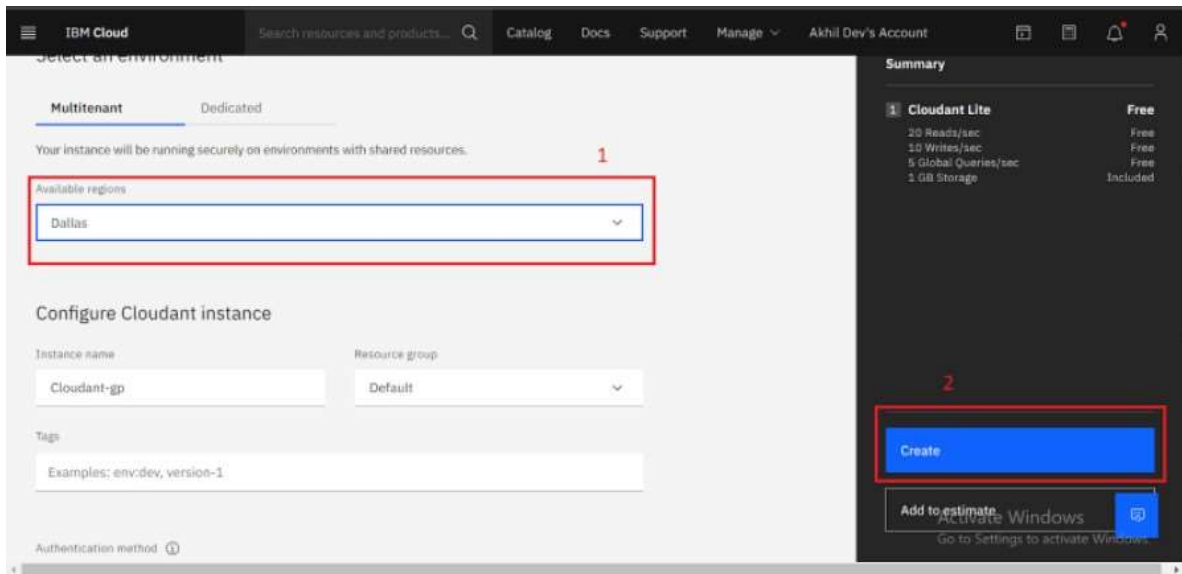


Fig. 5.4 Select Region and click on create

After clicking on create, the system displays a message to say that the instance is being provisioned, which returns to the Resource list. From the Resource list, it displays the status for the current 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 application needs to connect to the instance, click on New credential as shown in Fig. 5.5.

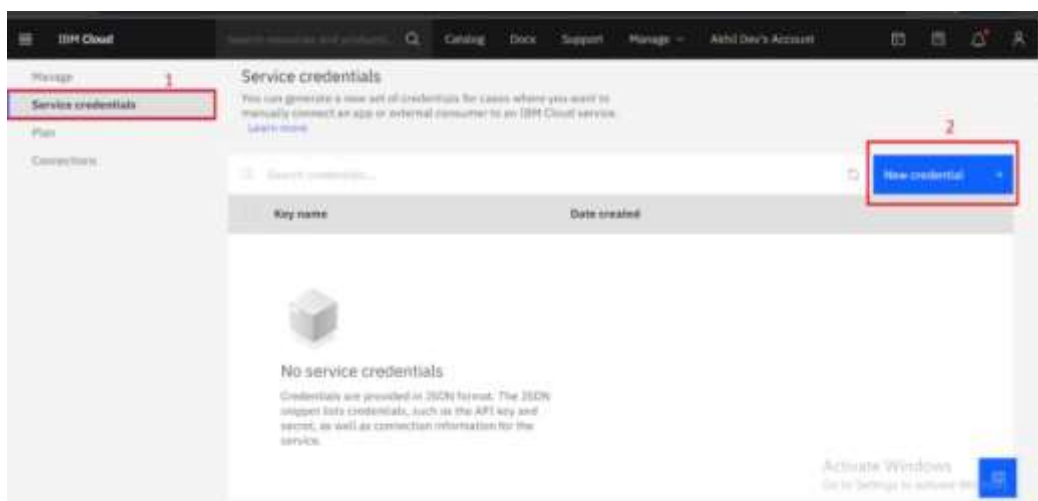
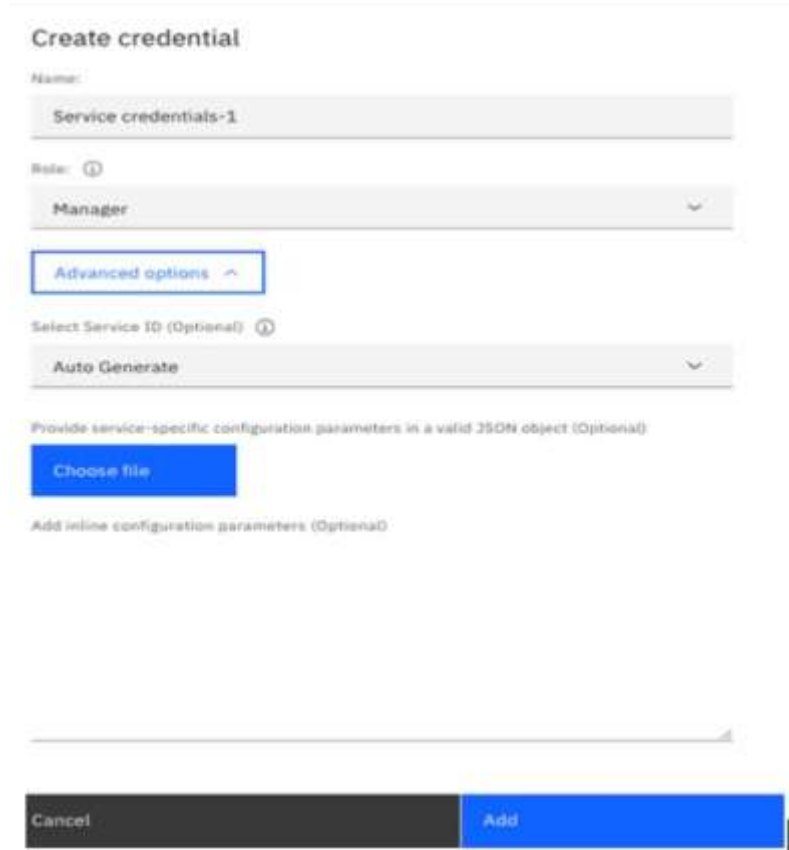


Fig. 5.5 Connection creation

2. Enter a name for the new credential in the Add new credential window.
3. Accept the Manager role.
4. Create a service ID or have one automatically generated for user.

5. 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.



Create credential

Name:

Role:

[Advanced options](#)

Select Service ID (Optional):

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

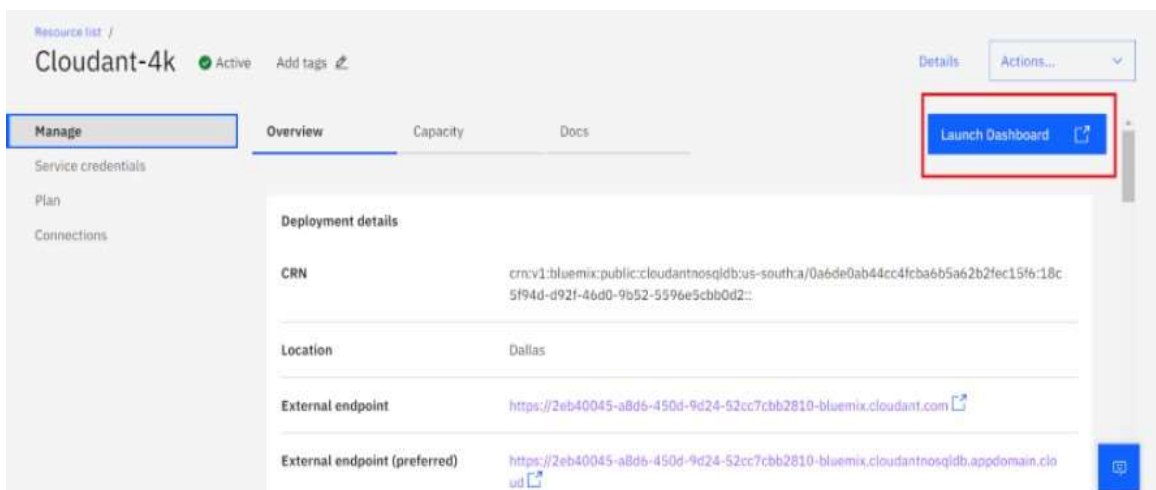
[Choose file](#)

Add inline configuration parameters (Optional)

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.



Resource list / **Cloudant-4k** Active [Add tags](#)

Manage Overview Capacity Docs

Service credentials
Plan
Connections

Launch Dashboard

Deployment details

CRN	crn:v1:bluemix:public:cloudantnosqldb:us-south:a/0a6de0ab44cc4fcbab5a62b2fec15f6:18c5f94d-d92f-46d0-9b52-5596e5cbb0d2::
Location	Dallas
External endpoint	https://2eb40045-a8d6-450d-9d24-52cc7cbb2810-bluemix.cloudant.com
External endpoint (preferred)	https://2eb40045-a8d6-450d-9d24-52cc7cbb2810-bluemix.cloudantnosqldb.appdomain.cloud

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.

```
from cloudant.client import Cloudant
```

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

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.

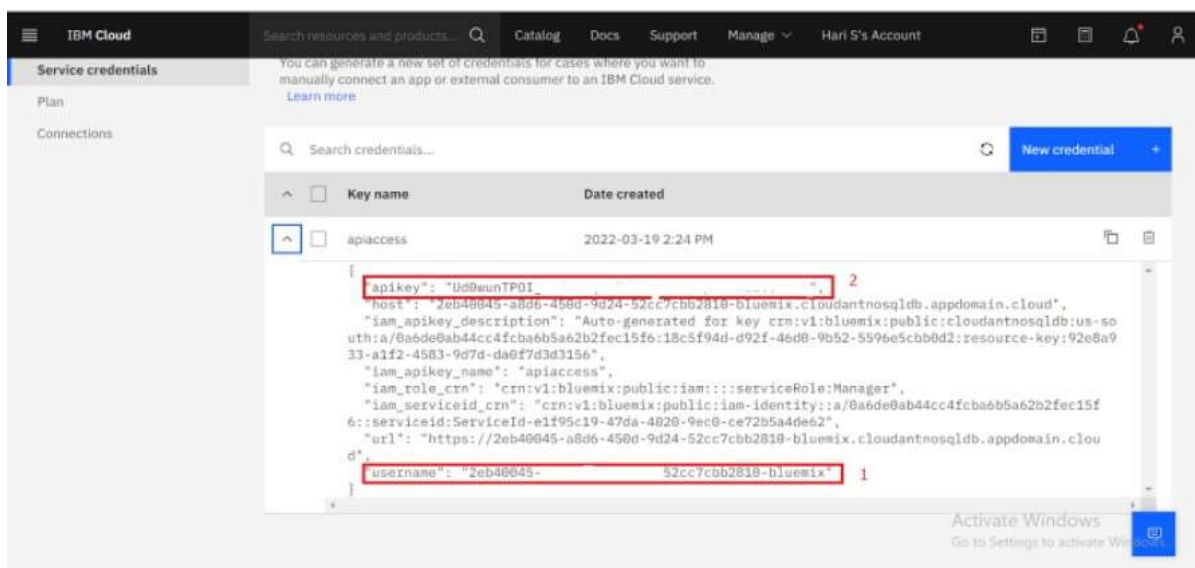


Fig. 5.9. Access the Username and API key

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

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

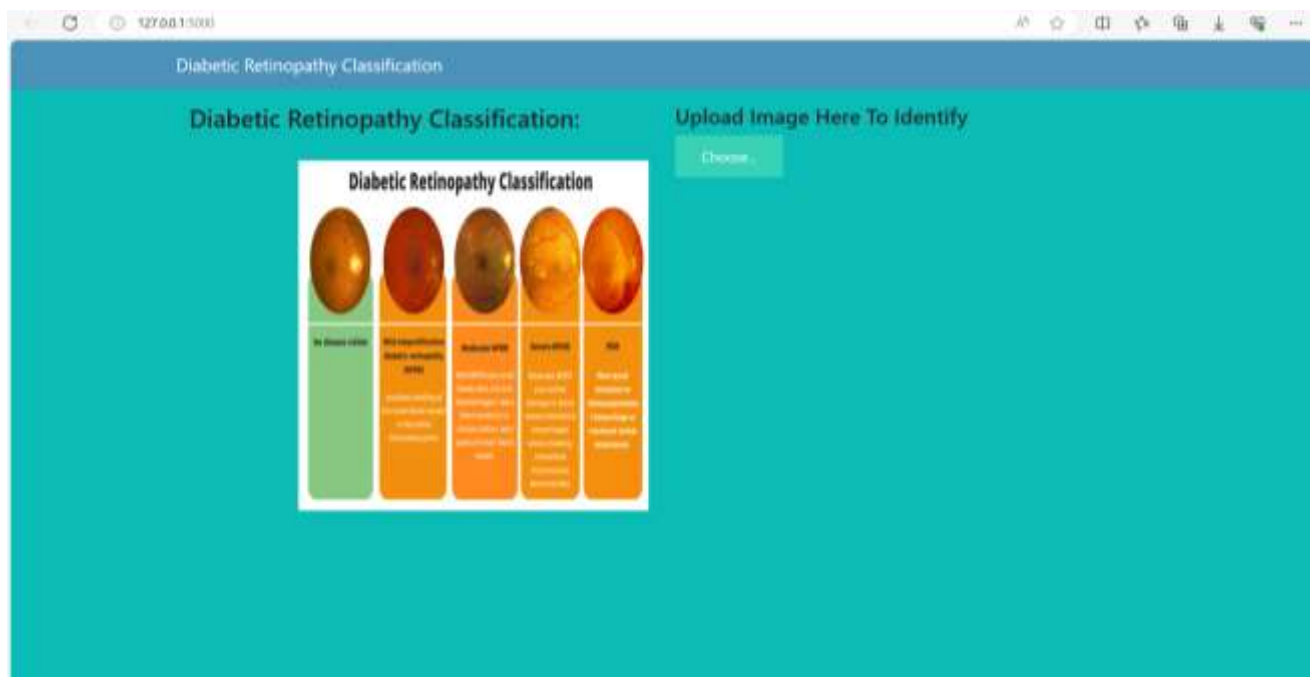


Fig. 6.1. Home page of Diabetic Retinopathy Classification Webpage

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

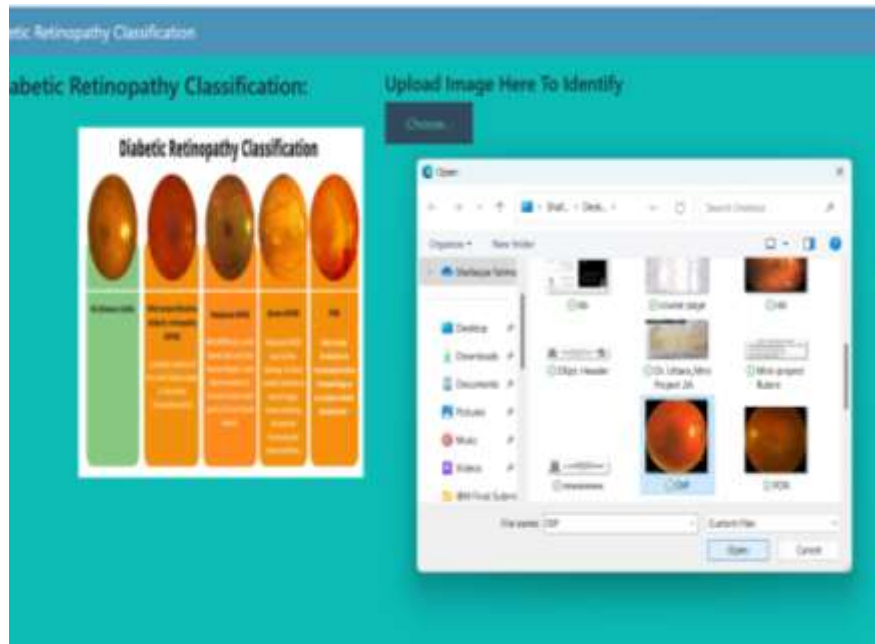


Fig. 6.2 Select an image to predict the type of diabetic condition.



After the successful upload of the image, press the submit 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.

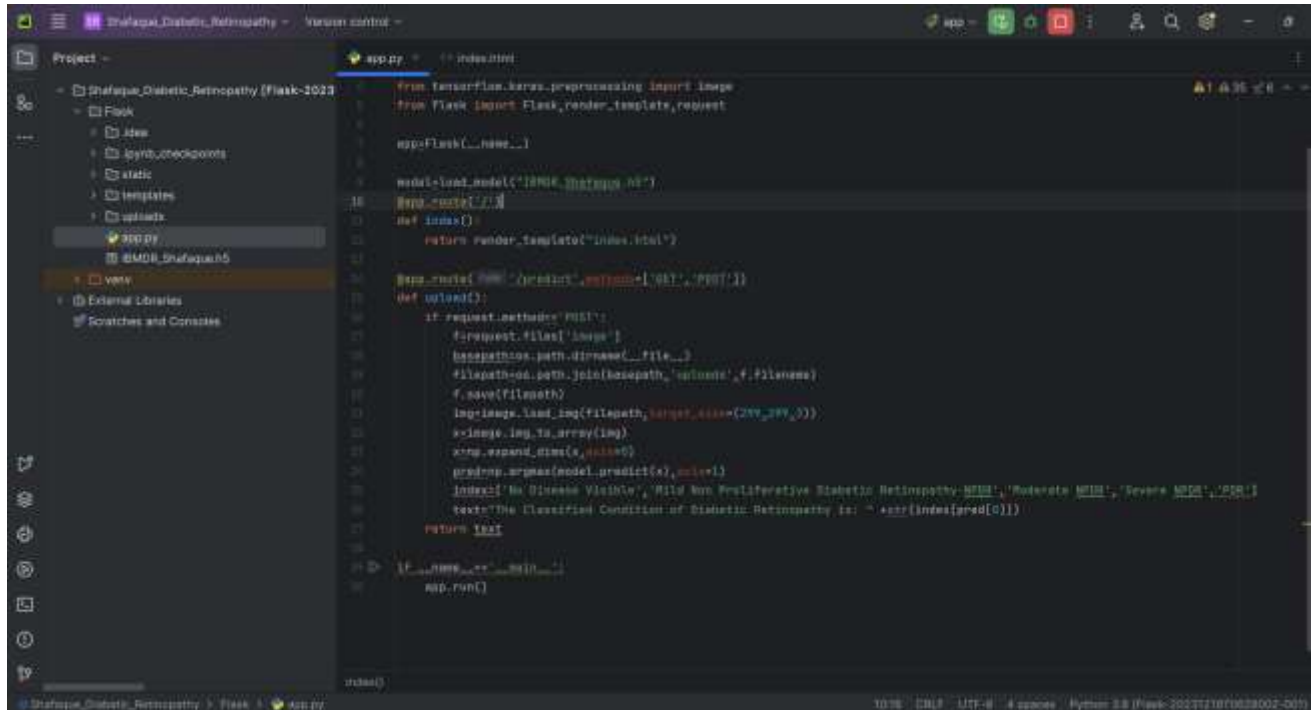


Fig. 6.4. Building python code in PyCharm

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:

```

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

```

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

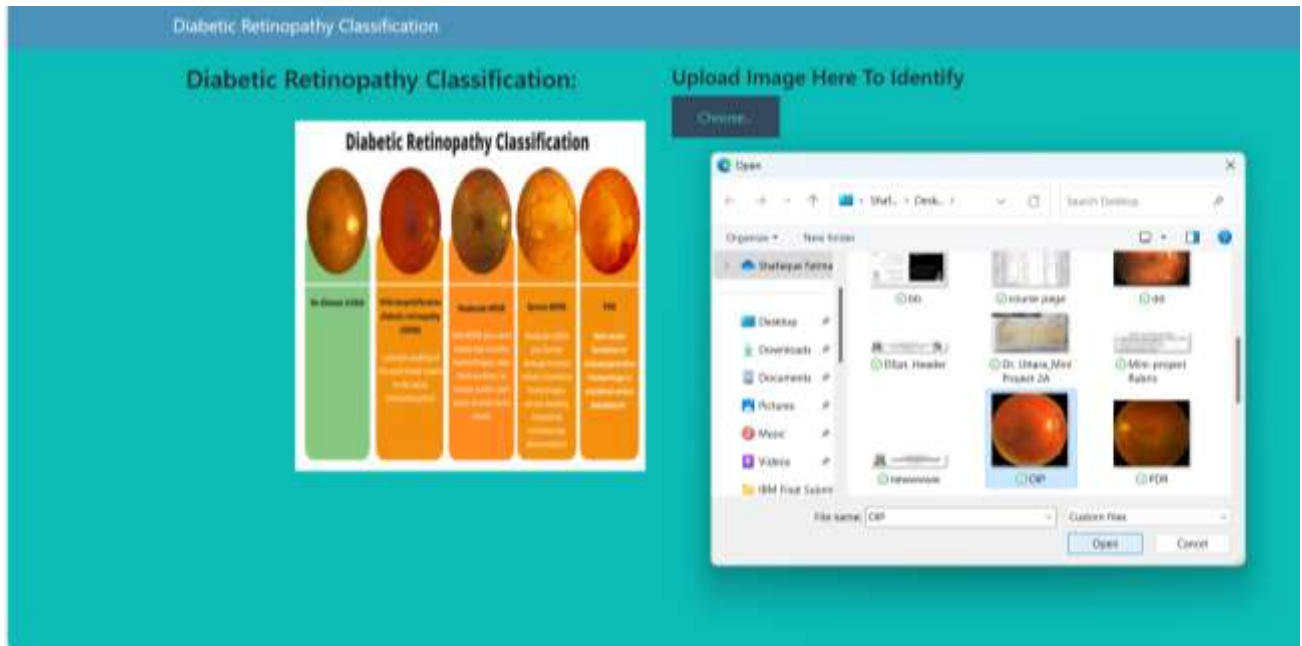


Fig. 7.2. Select an image to predict the class of DR.

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.

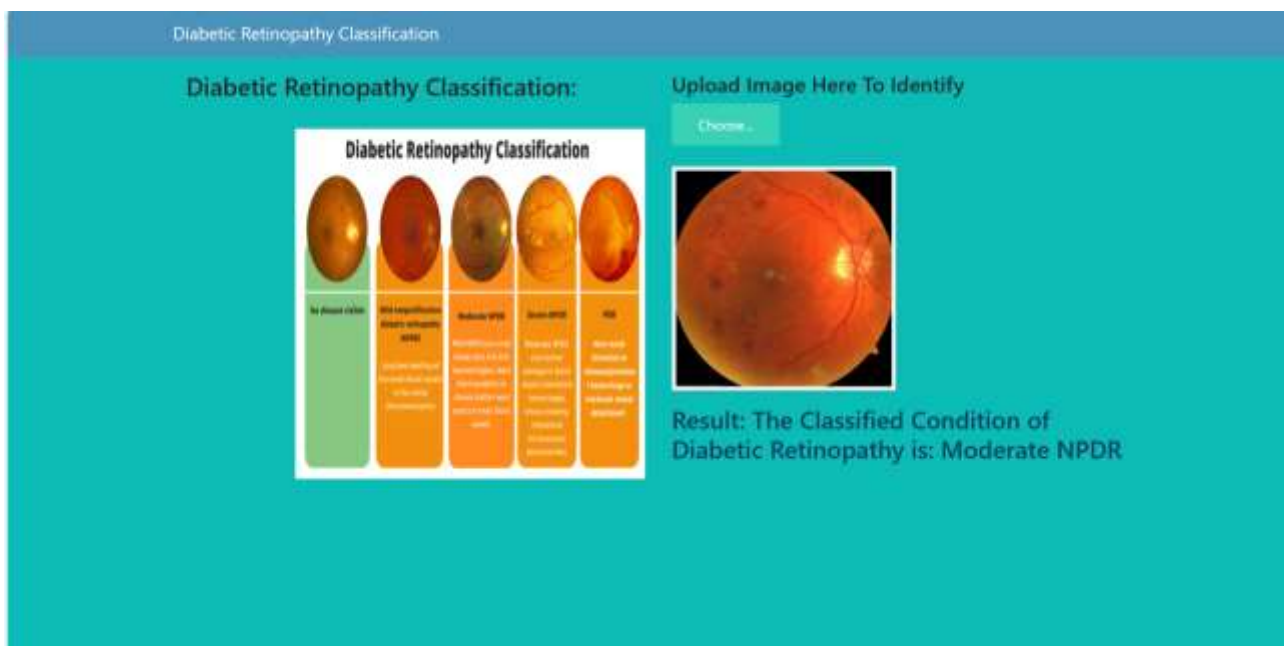


Fig. 7.3 Predict the outcome of selected image

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.

AutoML processing of CNNs: In autoML assistants learn and process the model is updated automatically and predict more 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

BIBLIOGRAPHY

1. www.embibe.com
2. www.slidehare.com
3. www.researchgate.net
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