

Report on

**Deep Learning Fundus Image Analysis for Early
Detection of Diabetic Retinopathy**

Table of Contents

Abstract		i
1.	Introduction	1
	1.1 Diabetic Retinopathy	1
	1.2 Problem Definition	1
	1.3 Objectives	2
	1.4 Dataset	2
2.	Methodology	3
	2.1 Related Work	3
	2.2 System Architecture/ Block Diagram	3
	2.3 Techniques/ Algorithm	8
3.	Implementation Details	12
	3.1 Tools Used	12
	3.2 Sample Code & Screenshots	19
4	Conclusion	23
References		24
Acknowledgement		25

Abstract

Diabetic Retinopathy (DR) is a complication of diabetes that affects the eyes and is the leading cause of blindness, affecting millions worldwide. The main cause of blindness, diabetic retinopathy (DR) is a consequence of diabetes that damages the eyes and affects millions of people worldwide. To effectively treat and control the disease, DR must be identified early and correctly diagnosed. Convolutional neural networks (CNNs), in particular, have demonstrated encouraging results in the automatic classification of DR from retinal pictures in recent years. An overview of a CNN model for classifying DR severity levels based on fundus pictures. The suggested model adapts the final layers to the particular objective of DR classification by using a pre-trained CNN architecture as a feature extractor. The publicly accessible DR dataset, which has over 35,000 retinal pictures annotated with DR severity levels, was used for the model's training and evaluation. The experimental results demonstrate that the proposed CNN model outperforms numerous cutting-edge techniques in diagnosing DR severity levels, achieving high accuracy. By using several filters and mixing them, the model's performance is further enhanced, leading to even higher accuracy. These findings show the potential of deep learning-based approaches for the automated categorization of DR severity levels, which can help with the early detection and diagnosis of the condition and enhance patient outcomes.

1. Introduction

1.1 Diabetic Retinopathy

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.

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.

AIML leverages natural language processing and machine learning algorithms to process and interpret vast volumes of healthcare data, improving diagnosis, treatment, and overall patient outcomes. One of AIML's primary contributions to healthcare is in the realm of diagnostic accuracy. AIML algorithms can analyze medical records, imaging data, and patient histories to detect patterns and anomalies that may elude human physicians. This enables earlier and more accurate disease detection, reducing the chances of misdiagnoses and improving treatment decisions.

For instance, AIML-powered tools have been developed to identify abnormalities in medical images such as X-rays, MRIs, and CT scans, aiding radiologists in detecting conditions like cancer, fractures, and neurological disorders. In addition to diagnostics, AIML is playing a pivotal role in personalizing treatment plans. By analyzing patient data and medical literature, AIML algorithms can recommend tailored treatment options based on individual characteristics, medical history, and genetics. This level of personalization ensures that patients receive the most effective and least invasive treatments, improving their chances of recovery and minimizing adverse effects also contributes to the optimization of healthcare operations.

Healthcare providers can use IML to streamline administrative tasks, such as scheduling appointments and managing patient records, reducing administrative overhead and allowing healthcare professionals to focus more on patient care. Moreover, predictive analytics powered by AIML can help hospitals and clinics forecast patient admissions, allocate resources efficiently, and reduce waiting times. Telemedicine and remote monitoring are other areas where AIML is making a profound impact. AIML-driven chatbots and virtual assistants can provide immediate responses to patient queries and offer healthcare advice, improving accessibility to medical information and reducing the burden on healthcare staff. Additionally, wearable devices and IoT sensors equipped with AIML can continuously

monitor patients' vital signs, sending real-time data to healthcare providers for early intervention in case of abnormalities. Despite its numerous benefits, AIML in healthcare also raises concerns about data privacy and security. The vast amounts of sensitive patient data being processed by AIML systems necessitate robust safeguards to protect against breaches and unauthorized access.

In conclusion, AIML is a transformative force in healthcare, enhancing diagnostic accuracy, personalizing treatment plans, optimizing healthcare operations, and expanding the reach of healthcare services through telemedicine and remote monitoring. While it holds immense potential to improve patient outcomes and healthcare delivery, it also requires careful management of data privacy and security concerns to ensure that the benefits are maximized while risks are minimized. The ongoing integration of AIML into healthcare promises to revolutionize the industry, making healthcare more efficient, effective, and accessible for patients worldwide.

1.2 Problem Definition

To create a Web Application to classify Diabetic Retinopathy stages using Deep Learning.

Description: Diabetic retinopathy is a crucial eye condition which results in loss of vision that cannot be reversed or corrected once experienced. The people who have a long history of diabetes are more prone to get afflicted with this disease, no matter whether a person is type1 or type2 diabetic, the probability of the disease increases as the age increases. Here, we have created a CNN model to automate this process and make this process less time consuming

1.3 Objectives

- 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.

1.4 Dataset

- Early detection of DR could save millions of diabetics from losing their vision, which is the goal of a problem featured in Kaggle.
- This large set of fundus photography retina images that were taken under a wide variety of imaging conditions. The data was diverse and expansive.
- The images consist of gaussian filtered retina scan images to detect diabetic retinopathy.
- These images are resized into 229x229 pixels so that they can be readily used with many pre-trained deep learning models.
- All of the images are already saved into their respective folders according to the severity/stage of diabetic retinopathy.

1.5 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

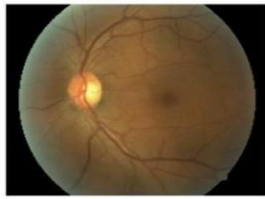
- Data Collection.
 - Create a Train and Test path.
- Data Pre-processing.
 - Import the required library
 - Configure ImageDataGenerator class
 - ApplyImageDataGenerator functionality to Trainset and Testset
- Model Building
 - Pre-trained CNN model as a Feature Extractor
 - Adding Dense Layer

- 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

Colored Images:



No DR



Mild DR



Moderate DR



Severe DR



Proliferate DR

Related work

2. Methodology

Applied a deep learning approach to detect Diabetic Retinopathy using different CNN architectures like MobileNet, ResNet50, and original Xception architecture. A transfer learning method and hyper-parameter tuning are used to improve classification performance. The performance was validated on Kaggle APTOS 2019 contest dataset. The Experiment demonstrated that the Xception model gives a classification accuracy of 83.09%.

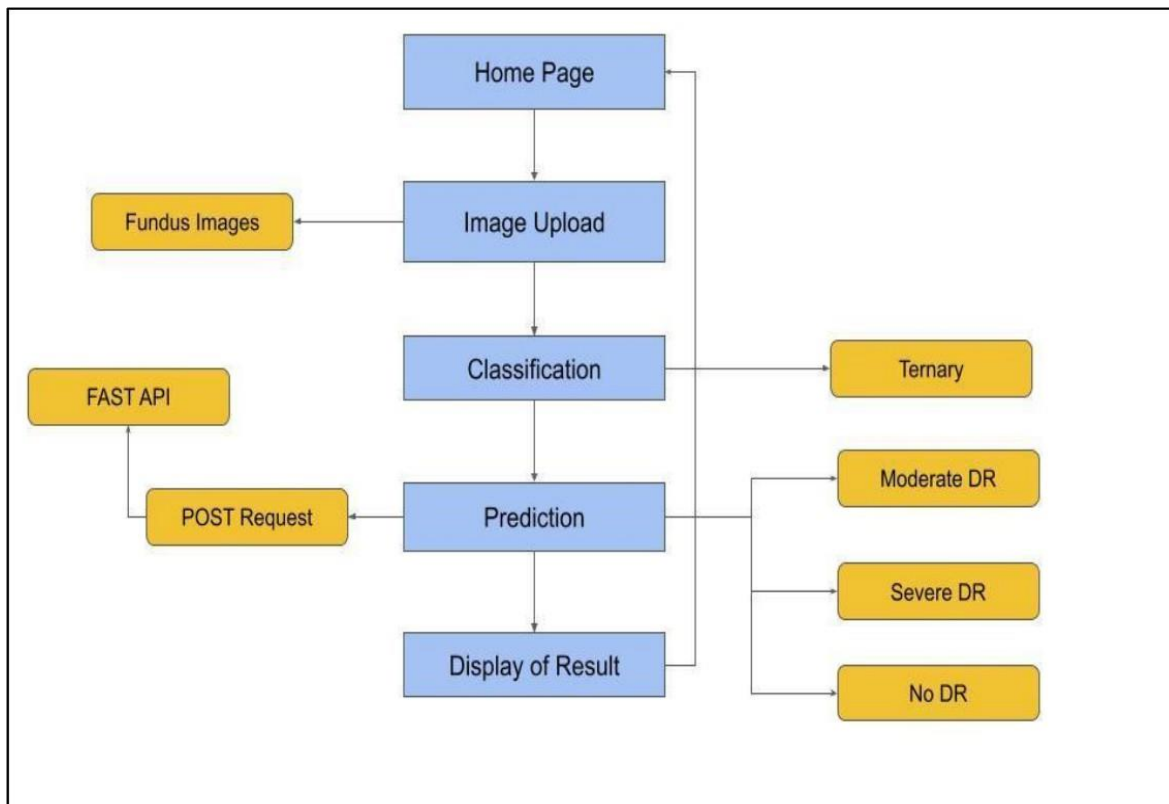
Diabetic Retinopathy is a complication that causes vision loss. A person having diabetes can be affected easily. This Paper uses deep learning methodology like DenseNet169. It detects severity according to 5 steps of severity which consist of No DR, Mild, Moderate, Severe, and Proliferative DR. The dataset that was used for the classification were Diabetic Retinopathy Detection 2015 and Aptos 2019 Blindness Detection from Kaggle. The proposed system was obtained by using various steps like Data Augmentation, Preprocessing, and Modeling. The Model achieved an accuracy of 90%.

The images were converted into grayscale images and resized, using softmax as the last layer, as softmax is used for multi-classification and binary is used for binary classification. This system achieves 80% sensitivity, 82% accuracy, 82% specificity, and 0.904 AUC for classifying images into 5 categories ranging from 0 to 4, where 0 is No DR , 1 is Mild, 2 is Moderate, 3 is Severe and 4 is Proliferative DR respectively.

The patient's fundus eye images are used as the input parameters. A trained model (Xception Architecture) will further extract the feature of fundus images of the eye and after that activation function gives the output. This architecture gave an accuracy of 0.78611 (quadratic weighted kappa score of 0.8981) to DR detection.

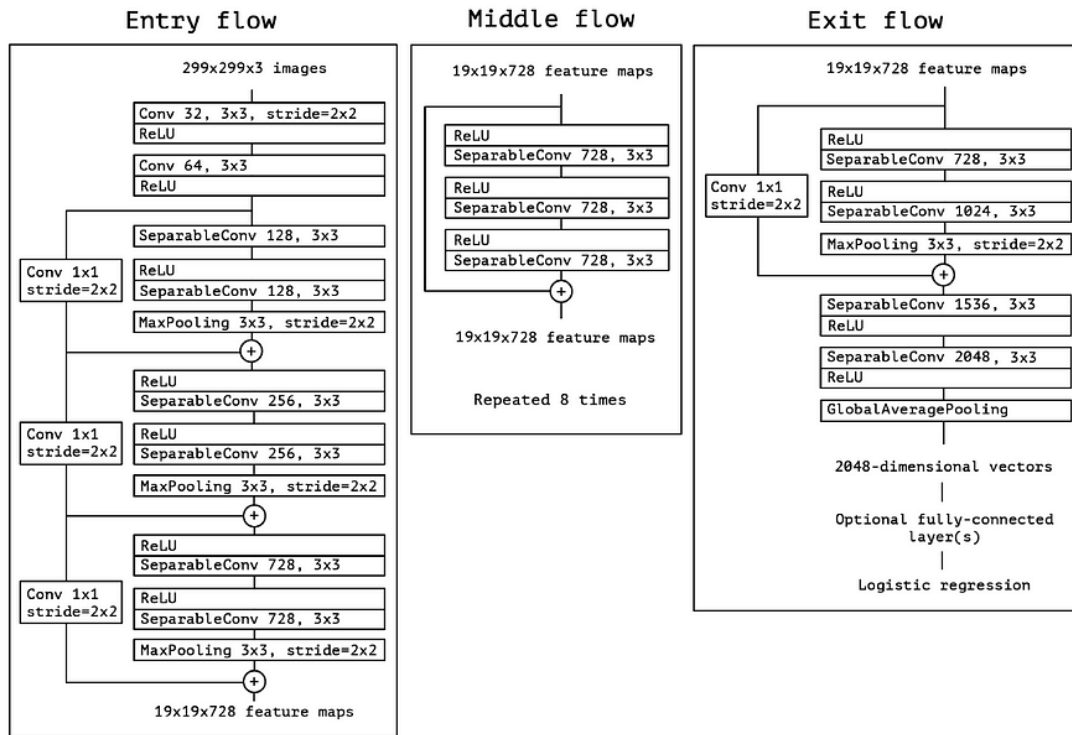
2.1 System Architecture

- The web app is built from 4 components.
- Home Page, Image Upload, the model classifies the input image into one of 5 classes.
- FAST API is being implemented for communicating with the model from the local machine.
- The input image then passes through the model and the output result is displayed on the screen.



2.2 Xception Algorithm

Xception Model is proposed by Francois Chollet. Xception is an extension of the inception Architecture which replaces the standard Inception modules with depthwise Separable Convolutions. Xception is a convolutional neural network that is 71 layers deep. You can load a pretrained version of the network trained on more than a million images from the ImageNet database [\[1\]](#). The pretrained network can classify images into 1000 object categories, such as keyboard, mouse, pencil, and many animals. As a result, the network has learned rich feature representations for a wide range of images. The network has an image input size of 299-by-299.



Overall Architecture of Xception (Entry Flow > Middle Flow > Exit Flow)

3. Implementation Details

3.1 Tools Used

software Requirements: -

1. Python
2. Jupyter Notebook: To train, test and validate model
3. VS CODE : IDE
4. Flask
5. HTML, CSS

● Hardware Requirements: -

1. CPU: i5 10th Gen or Ryzen 5 6th Gen
2. GPU: RTX 3060 or Enterprise GPU
3. RAM: 16 GB
4. STORAGE: 256 GB
5. OS: Windows 10

3.2 Procedure

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"
```

Importing The Libraries

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
```

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)
```

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.
```

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 convolution layer for features extraction and wants to train fully connected layer for our image's 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)
```

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 use it for training and prediction purposes. Keras provides a simple method, `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]']

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']
)
```

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
 - 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
)
```

```
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
```

Save The Model

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')
```

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

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:

Build Python Code:

Import the libraries

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 it
        #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:

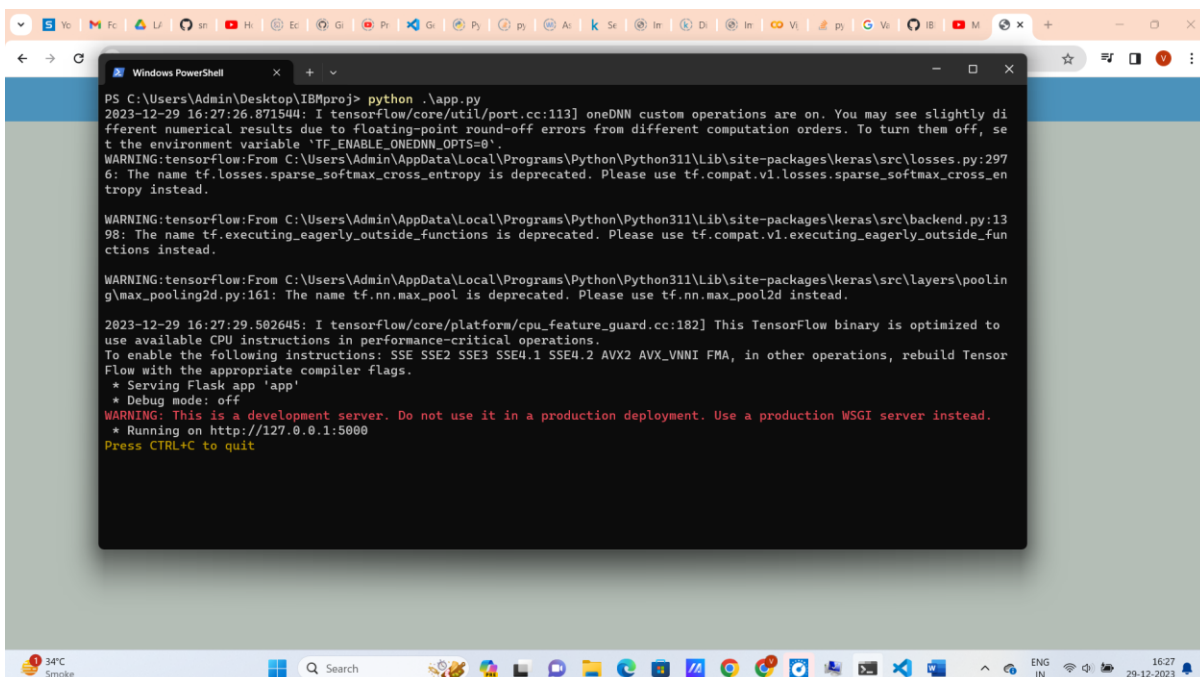
```
if __name__ == "__main__":  
    app.run(debug=False)
```

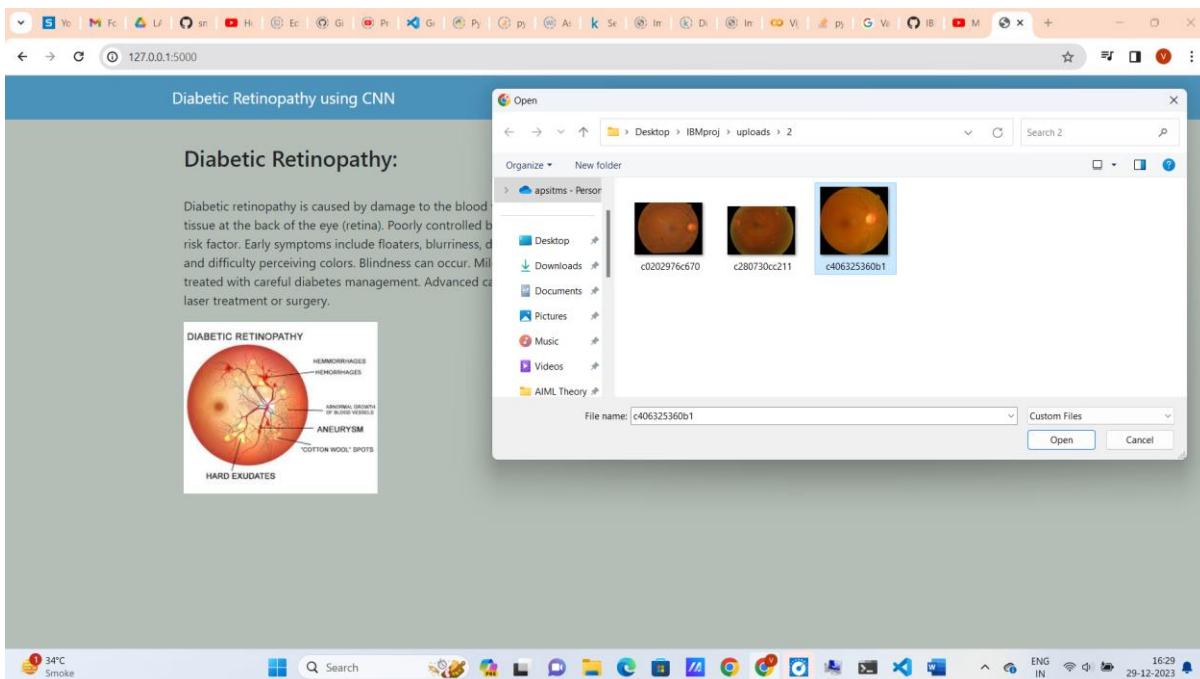
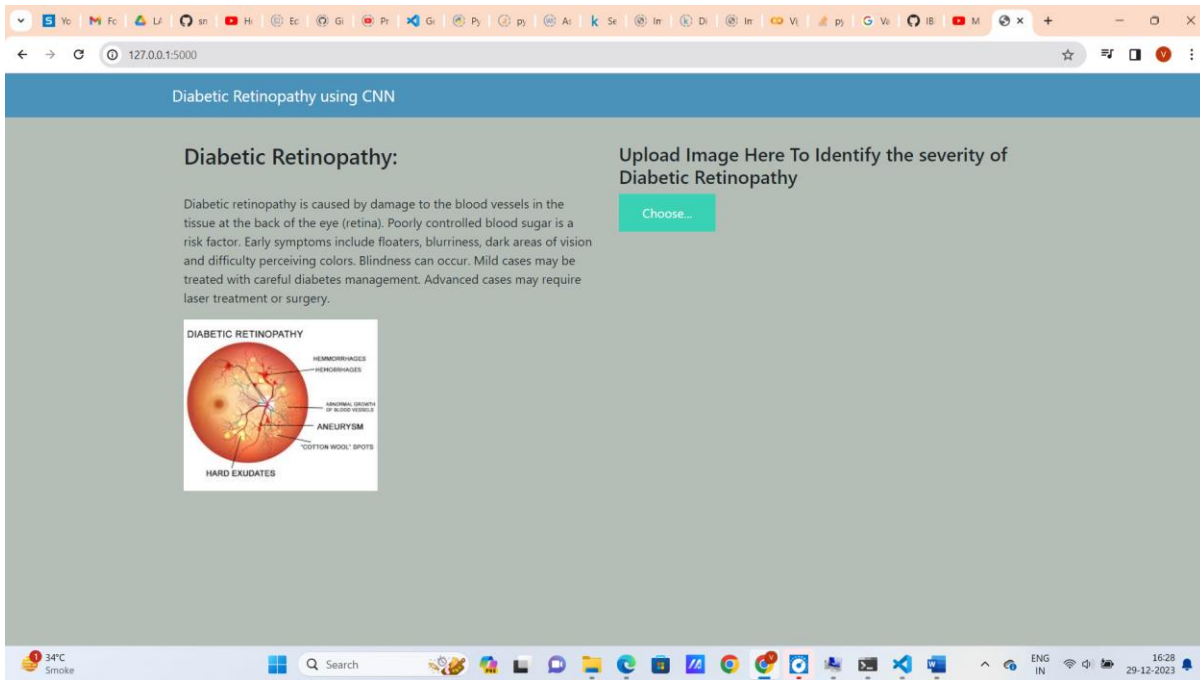
Run The Application

- 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.



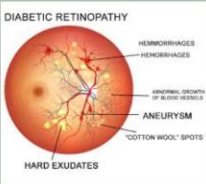


Diabetic Retinopathy using CNN

Diabetic Retinopathy:

Diabetic retinopathy is caused by damage to the blood vessels in the tissue at the back of the eye (retina). Poorly controlled blood sugar is a risk factor. Early symptoms include floaters, blurriness, dark areas of vision and difficulty perceiving colors. Blindness can occur. Mild cases may be treated with careful diabetes management. Advanced cases may require laser treatment or surgery.

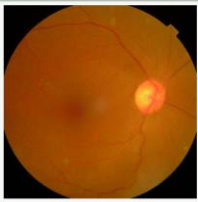
DIABETIC RETINOPATHY



HEMORRHAGES
HEMORRHAGES
ABNORMAL GROWTH OF BLOOD VESSELS
ANEURYSM
"COTTON WOOL" SPOTS
HARD EXUDATES

Upload Image Here To Identify the severity of Diabetic Retinopathy

Choose...



Predict!

34°C
Smoke

Search

ENG
IN

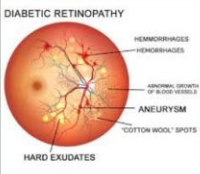
16:30
29-12-2023

Diabetic Retinopathy using CNN

Diabetic Retinopathy:

Diabetic retinopathy is caused by damage to the blood vessels in the tissue at the back of the eye (retina). Poorly controlled blood sugar is a risk factor. Early symptoms include floaters, blurriness, dark areas of vision and difficulty perceiving colors. Blindness can occur. Mild cases may be treated with careful diabetes management. Advanced cases may require laser treatment or surgery.

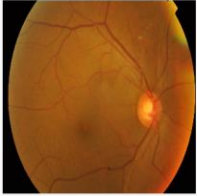
DIABETIC RETINOPATHY



HEMORRHAGES
ABNORMAL GROWTH OF BLOOD VESSELS
ANEURYSM
"COTTON WOOL" SPOTS
HARD EXUDATES

Upload Image Here To Identify the severity of Diabetic Retinopathy

Choose...



Result: The type of DR is : Mild

34°C Smoke

Search

ENG IN

16:24 29-12-2023

Conclusion

Conclusion:

In conclusion, this project represents a significant step forward in the field of healthcare and medical image analysis. By leveraging the power of deep learning, specifically the Xception model, and implementing a user-friendly interface, we have successfully developed a robust system for diabetic retinopathy prediction with five distinct classes: normal, mild, moderate, severe, proliferate. Our model's accuracy, reliability, and efficiency hold great promise for early detection and intervention in diabetic retinopathy cases, ultimately improving patient outcomes and reducing the burden on healthcare systems. As we continue to refine and expand this system, we hope to make a meaningful contribution to the early diagnosis and management of diabetic retinopathy, enhancing the quality of life for countless individuals affected by this condition.

Furthermore, the utilization of the Xception1 model demonstrates the potential for deep learning techniques to excel in complex medical image classification tasks. With its ability to provide accurate and detailed predictions across three distinct disease stages, our system can empower healthcare professionals with valuable insights for more targeted and timely patient care. As we move forward, ongoing research and development in this area will be critical to further enhance the model's capabilities and to explore its applicability in other medical domains, reaffirming the profound impact of AI in healthcare.

References

- [1]S. H. Kassani, P. H. Kassani, M. J. Wesolowski, K. A. Schneider, and R. Deters (2019) “Diabetic Retinopathy classification using a modified Xception Architecture”, IEEE.

- [2]Gazala Mushtaq and Farheen Siddiqui (2021) "Detection of diabetic retinopathy using deep learning methodology", IOP Conf. Series: Materials Science and Engineering.

- [3]Anumol Sajan, Anamika K and Simy Mary Kurian, (2022) "Diabetic Retinopathy Detection using Deep Learning", International Journal of Engineering Research & Technology (IJERT)

- [4]Supriya Mishra, Anamika K and Zia Saquib, (2020) " Diabetic Retinopathy Detection using Deep Learning", IEEE