**AD3002-Health Care Analytics**

**Mini Project**

**GitHub Link:** https://github.com/mukeshsundar23/Health-care-analytics

**Title:** Eye Disease Prediction

**Task:**

Eye disease classification is a research area that focuses on developing algorithms and models to accurately classify different types of eye diseases based on medical imaging data. It plays a critical role in assisting ophthalmologists and healthcare professionals in effectively diagnosing and treating eye diseases.

The primary objective of eye disease classification is to leverage machine learning and computer vision techniques to analyse medical images and detect the four diseases: cataract, diabetic retinopathy, glaucoma, normal

**About the diseases:**

**Cataract:** Cataract is a common age-related eye condition characterized by the clouding of the lens, leading to blurry vision and visual impairment. It can be treated surgically by replacing the cloudy lens with an artificial one, restoring clear vision and improving quality of life.

**Diabetic Retinopathy:** Diabetic retinopathy is a complication of diabetes that affects the blood vessels in the retina. It can cause vision loss, including blurred or distorted vision, and in severe cases, lead to blindness. Early detection, regular eye exams, and proper management of diabetes are crucial for preventing and managing this condition.

**Glaucoma:** Glaucoma is a group of eye diseases that damage the optic nerve, often due to increased fluid pressure in the eye. It gradually leads to vision loss, starting with peripheral vision and potentially progressing to complete blindness. Timely diagnosis, treatment, and ongoing monitoring are vital for preserving vision and preventing irreversible damage.

**Use Case:**

Eye disease classification has several important use cases and applications:

**Screening and Early Detection:** Eye disease classification algorithms can serve as screening tools to identify individuals at risk of developing eye diseases. By analysing medical images, these models can detect early signs of diseases like diabetic retinopathy, age-related macular degeneration, glaucoma, and others. Early detection enables prompt intervention and treatment, potentially preventing vision loss.

**Diagnosis Support:** Eye disease classification models can assist healthcare professionals, especially those with limited ophthalmic expertise, in making accurate diagnoses. By providing additional insights and suggestions based on image analysis, these models act as decision support systems, enhancing the accuracy and efficiency of diagnoses.

**Treatment Planning and Monitoring:** Once an eye disease is diagnosed, classification algorithms can aid in treatment planning and monitoring. By analysing sequential imaging data, these models can track disease progression, assess the effectiveness of treatments, and guide adjustments in treatment plans as required.

**Techniques used:** Transfer Learning and Deep Learning.

**Code:**

**Importing Libraries**

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

import seaborn as sea

import os

from tqdm import tqdm

import cv2 as op

import torch

from torchsummary import summary

import torchmetrics

np.\_\_version\_\_

device = 'cuda' if torch.cuda.is\_available() else 'cpu'

device



**Loading the data**

PATH = 'C:/Users/Mukesh/Desktop/Healhcare\_miniproject/dataset'

label2id = {}

for i, label in enumerate(os.listdir(PATH)):

label2id[label] = i

id2label = {key: value for (value, key) in label2id.items()}

filenames, outcome = [], []

for label in tqdm(os.listdir(PATH)):

for img in os.listdir(os.path.join(PATH, label)):

filenames.append(os.path.join(PATH, label, img))

outcome.append(label2id[label])



df = pd.DataFrame({

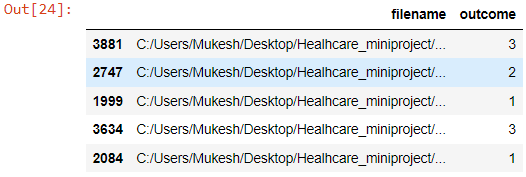
"filename": filenames,

"outcome": outcome

})

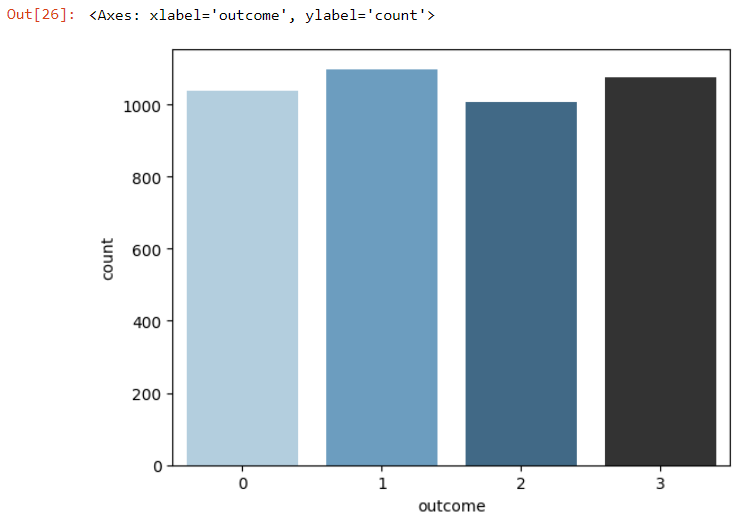
df = df.sample(frac=1)

df.head(5)



**Plotting the class distribution**

sea.countplot(x='outcome', hue='outcome', data=df, palette='Blues\_d', legend=False)



**Plotting the sample images**

def load\_image(path):

img = plt.imread(path)

img = (img - img.min())/img.max()

return img

counter = 0

plt.figure(figsize = (10, 12))

for i in range(4):

for path in df[df['outcome'] == i].sample(n = 3)['filename']:

plt.subplot(4, 3, counter + 1)

img = load\_image(path)

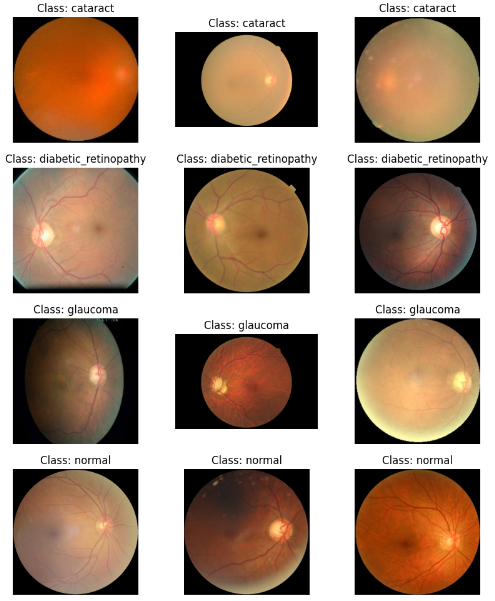
plt.imshow(img)

plt.axis('off')

plt.title('Class:' + " " + id2label[i])

counter += 1

plt.show()



**Building the dataset**

import torch

import torch.nn as nn

import torch.nn.functional as F

from torch.utils.data import Dataset, DataLoader

import torchvision

from torchvision import transforms

import matplotlib.pyplot as plt # Import matplotlib for plt.imread

train\_transform = transforms.Compose([

transforms.ToTensor(),

transforms.Resize((224, 224)),

transforms.RandomHorizontalFlip(p=0.5),

transforms.RandomVerticalFlip(p=0.5)

])

val\_transform = transforms.Compose([

transforms.ToTensor(),

transforms.Resize((224, 224))

])

class EyeDataset(Dataset):

def \_\_init\_\_(self, df, n\_classes, transform=None):

self.df = df

self.n\_samples = len(self.df)

self.n\_classes = n\_classes

self.transform = transform

def \_\_len\_\_(self):

return self.n\_samples

def \_\_getitem\_\_(self, index):

img = plt.imread(self.df.iloc[index, 0]) # Load the image with plt.imread

label = self.df.iloc[index, 1]

img = (img - img.min()) / (img.max() - img.min()) # Normalize the image

if self.transform:

img = self.transform(img)

z

return img.to(torch.float32), label

from sklearn.model\_selection import train\_test\_split

df\_train, df\_val = train\_test\_split(df, test\_size = 0.15, random\_state = 28)

df\_train.shape, df\_val.shape



NUM\_CLASSES = 4

BATCH\_SIZE = 128

train\_dataset = EyeDataset(df\_train, NUM\_CLASSES, train\_transform)

val\_dataset = EyeDataset(df\_val, NUM\_CLASSES, val\_transform)

train\_loader = DataLoader(train\_dataset, batch\_size = BATCH\_SIZE, shuffle = True)

val\_loader = DataLoader(val\_dataset, batch\_size = BATCH\_SIZE, shuffle = False)

train\_transform = transforms.Compose([

transforms.ToTensor(),

transforms.Resize((224, 224), antialias=True),

transforms.RandomHorizontalFlip(p=0.5),

transforms.RandomVerticalFlip(p=0.5)

])

val\_transform = transforms.Compose([

transforms.ToTensor(),

transforms.Resize((224, 224), antialias=True),

])

from math import ceil

class Net(nn.Module):

def \_\_init\_\_(self):

super().\_\_init\_\_()

self.base = torchvision.models.resnet18(pretrained = True)

for param in list(self.base.parameters())[:-15]:

param.requires\_grad = False

self.block = nn.Sequential(

nn.Linear(512, 128),

nn.ReLU(),

nn.Dropout(0.2),

nn.Linear(128, 4),

)

self.base.classifier = nn.Sequential()

self.base.fc = nn.Sequential()

def get\_optimizer(self):

return torch.optim.AdamW([

{'params' : self.base.parameters(), 'lr': 3e-5},

{'params' : self.block.parameters(), 'lr': 8e-4}

])

def forward(self, x):

x = self.base(x)

x = self.block(x)

return x

# 👁️ PyTorch: Eye Disease Classification| 92.7%

class Trainer(nn.Module):

def \_\_init\_\_(self, train\_loader, val\_loader, device):

super().\_\_init\_\_()

self.train\_loader = train\_loader

self.val\_loader= val\_loader

self.device = device

self.model = Net().to(self.device)

self.optimizer = self.model.get\_optimizer()

self.loss\_fxn = nn.CrossEntropyLoss()

self.accuracy = torchmetrics.Accuracy(task = "multiclass", num\_classes = NUM\_CLASSES).to(self.device)

self.history = {'train\_loss' : [], 'val\_loss': [], 'train\_acc': [], 'val\_acc': []}

def training\_step(self, x, y):

pred = self.model(x)

loss = self.loss\_fxn(pred, y)

acc = self.accuracy(pred, y)

self.optimizer.zero\_grad()

loss.backward()

self.optimizer.step()

return loss, acc

def val\_step(self, x, y):

with torch.no\_grad():

pred = self.model(x)

loss = self.loss\_fxn(pred, y)

acc = self.accuracy(pred, y)

return loss, acc

def step\_fxn(self, loader, step):

loss, acc = 0, 0

for X, y in tqdm(loader):

X, y = X.to(self.device), y.to(self.device)

l, a = step(X, y)

loss, acc = loss + l.item(), acc + a.item()

return loss/len(loader), acc/len(loader)

def train(self, epochs):

for epoch in tqdm(range(epochs)):

train\_loss, train\_acc = self.step\_fxn(self.train\_loader, self.training\_step)

val\_loss, val\_acc = self.step\_fxn(self.val\_loader, self.val\_step)

for item, value in zip(self.history.keys(), list([train\_loss, val\_loss, train\_acc, val\_acc])):

self.history[item].append(value)

print("[Epoch: {}] Train: [loss: {:.3f} acc: {:.3f}] Val: [loss: {:.3f} acc:{:.3f}]".format(epoch + 1, train\_loss, train\_acc, val\_loss, val\_acc))

trainer = Trainer(train\_loader, val\_loader, device)

**Summary of the model**

summary(trainer.model.base, (3, 224, 224))

----------------------------------------------------------------

Layer (type) Output Shape Param #

================================================================

Conv2d-1 [-1, 64, 112, 112] 9,408

BatchNorm2d-2 [-1, 64, 112, 112] 128

ReLU-3 [-1, 64, 112, 112] 0

MaxPool2d-4 [-1, 64, 56, 56] 0

Conv2d-5 [-1, 64, 56, 56] 36,864

BatchNorm2d-6 [-1, 64, 56, 56] 128

ReLU-7 [-1, 64, 56, 56] 0

Conv2d-8 [-1, 64, 56, 56] 36,864

BatchNorm2d-9 [-1, 64, 56, 56] 128

ReLU-10 [-1, 64, 56, 56] 0

BasicBlock-11 [-1, 64, 56, 56] 0

Conv2d-12 [-1, 64, 56, 56] 36,864

BatchNorm2d-13 [-1, 64, 56, 56] 128

ReLU-14 [-1, 64, 56, 56] 0

Conv2d-15 [-1, 64, 56, 56] 36,864

BatchNorm2d-16 [-1, 64, 56, 56] 128

ReLU-17 [-1, 64, 56, 56] 0

BasicBlock-18 [-1, 64, 56, 56] 0

Conv2d-19 [-1, 128, 28, 28] 73,728

BatchNorm2d-20 [-1, 128, 28, 28] 256

ReLU-21 [-1, 128, 28, 28] 0

Conv2d-22 [-1, 128, 28, 28] 147,456

BatchNorm2d-23 [-1, 128, 28, 28] 256

Conv2d-24 [-1, 128, 28, 28] 8,192

BatchNorm2d-25 [-1, 128, 28, 28] 256

ReLU-26 [-1, 128, 28, 28] 0

BasicBlock-27 [-1, 128, 28, 28] 0

Conv2d-28 [-1, 128, 28, 28] 147,456

BatchNorm2d-29 [-1, 128, 28, 28] 256

ReLU-30 [-1, 128, 28, 28] 0

Conv2d-31 [-1, 128, 28, 28] 147,456

BatchNorm2d-32 [-1, 128, 28, 28] 256

ReLU-33 [-1, 128, 28, 28] 0

BasicBlock-34 [-1, 128, 28, 28] 0

Conv2d-35 [-1, 256, 14, 14] 294,912

BatchNorm2d-36 [-1, 256, 14, 14] 512

ReLU-37 [-1, 256, 14, 14] 0

Conv2d-38 [-1, 256, 14, 14] 589,824

BatchNorm2d-39 [-1, 256, 14, 14] 512

Conv2d-40 [-1, 256, 14, 14] 32,768

BatchNorm2d-41 [-1, 256, 14, 14] 512

ReLU-42 [-1, 256, 14, 14] 0

BasicBlock-43 [-1, 256, 14, 14] 0

Conv2d-44 [-1, 256, 14, 14] 589,824

BatchNorm2d-45 [-1, 256, 14, 14] 512

ReLU-46 [-1, 256, 14, 14] 0

Conv2d-47 [-1, 256, 14, 14] 589,824

BatchNorm2d-48 [-1, 256, 14, 14] 512

ReLU-49 [-1, 256, 14, 14] 0

BasicBlock-50 [-1, 256, 14, 14] 0

Conv2d-51 [-1, 512, 7, 7] 1,179,648

BatchNorm2d-52 [-1, 512, 7, 7] 1,024

ReLU-53 [-1, 512, 7, 7] 0

Conv2d-54 [-1, 512, 7, 7] 2,359,296

BatchNorm2d-55 [-1, 512, 7, 7] 1,024

Conv2d-56 [-1, 512, 7, 7] 131,072

BatchNorm2d-57 [-1, 512, 7, 7] 1,024

ReLU-58 [-1, 512, 7, 7] 0

BasicBlock-59 [-1, 512, 7, 7] 0

Conv2d-60 [-1, 512, 7, 7] 2,359,296

BatchNorm2d-61 [-1, 512, 7, 7] 1,024

ReLU-62 [-1, 512, 7, 7] 0

Conv2d-63 [-1, 512, 7, 7] 2,359,296

BatchNorm2d-64 [-1, 512, 7, 7] 1,024

ReLU-65 [-1, 512, 7, 7] 0

BasicBlock-66 [-1, 512, 7, 7] 0

AdaptiveAvgPool2d-67 [-1, 512, 1, 1] 0

================================================================

Total params: 11,176,512

Trainable params: 7,213,056

Non-trainable params: 3,963,456

----------------------------------------------------------------

Input size (MB): 0.57

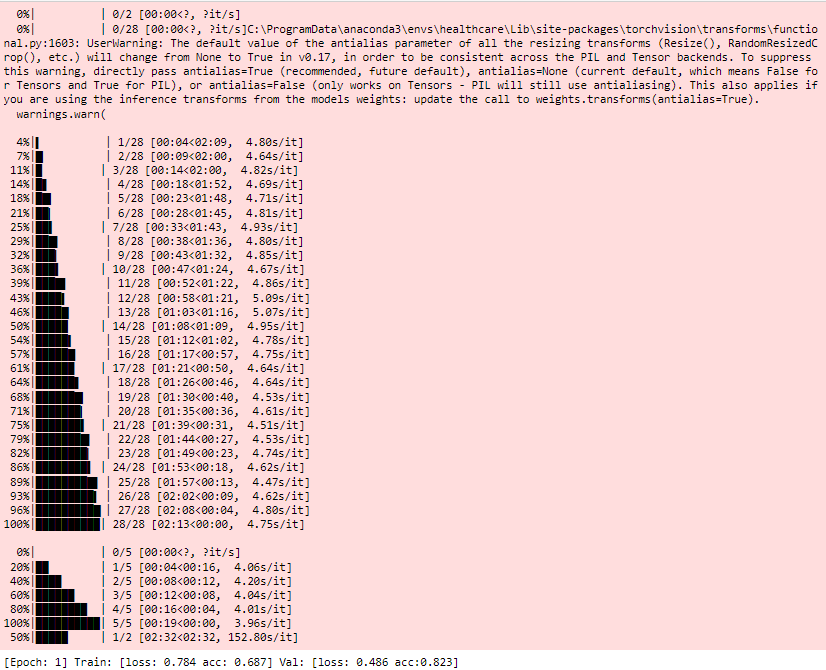
Forward/backward pass size (MB): 62.79

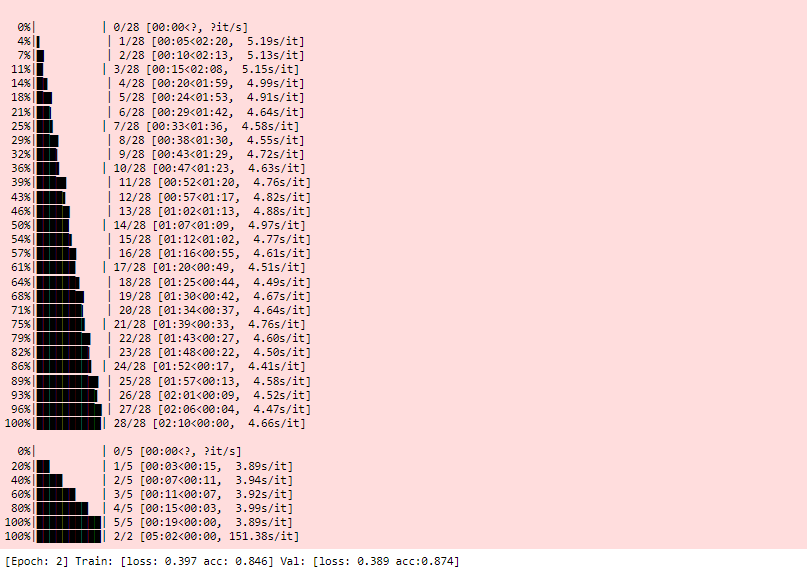
Params size (MB): 42.64

Estimated Total Size (MB): 105.99

**Training the model**

trainer.train(epochs = 2)





**Plotting Model Results**

plt.figure(figsize = (15, 4))

plt.subplot(1,2,1)

plt.title('Loss')

plt.plot(trainer.history['train\_loss'], label = 'Training')

plt.plot(trainer.history['val\_loss'], label = 'Validation')

plt.legend()

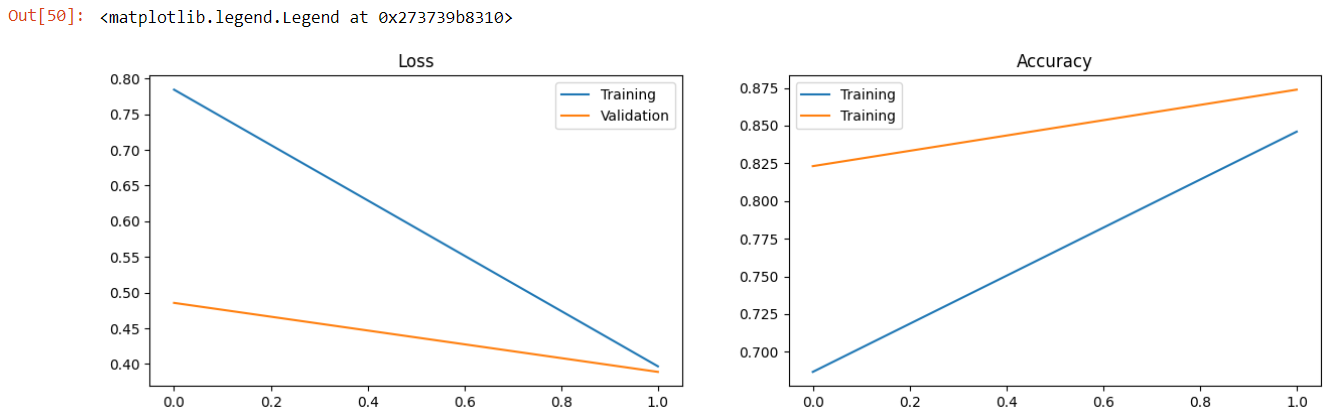
plt.subplot(1,2,2)

plt.title('Accuracy')

plt.plot(trainer.history['train\_acc'], label = 'Training')

plt.plot(trainer.history['val\_acc'], label = 'Training')

plt.legend()

****

**Model Predictions**

preds, true = [], []

with torch.no\_grad():

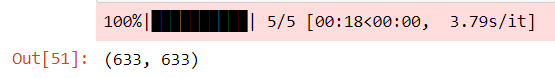
for x, y in tqdm(val\_loader):

pred = torch.argmax(trainer.model(x.to(device)), axis = 1).detach().cpu().numpy()

preds.extend(pred)

true.extend(y)

len(preds), len(true)

****

from sklearn.metrics import confusion\_matrix

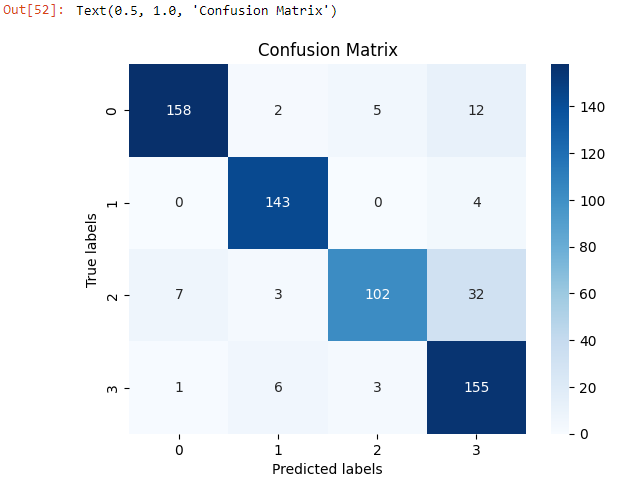
cm = confusion\_matrix(true, preds)

sea.heatmap(cm, annot=True, fmt='d', cmap='Blues', cbar=True)

plt.xlabel('Predicted labels')

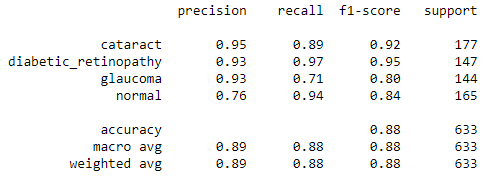
plt.ylabel('True labels')

plt.title('Confusion Matrix')

****

from sklearn.metrics import classification\_report

print(classification\_report(true, preds, target\_names = label2id.keys()))



**Testing the model with custom input**

import torch

import numpy as np

import matplotlib.pyplot as plt

# Provide the path to the image you want to test

image\_path = 'C:/Users/Mukesh/Desktop/Healhcare\_miniproject/dataset/cataract/\_19\_9976222.jpg' # Replace with the actual image path

# Ensure the model is in evaluation mode

trainer.model.eval()

# Load and preprocess the test image

img = plt.imread(image\_path)

img = (img - img.min()) / (img.max() - img.min())

# Convert the image to data type Double

img = img.astype(np.double)

# Apply the same transformations as used during training

img = val\_transform(img)

img = img.unsqueeze(0) # Add batch dimension

# Move the image to the device used for training (e.g., 'cuda' or 'cpu')

img = img.to(trainer.device)

# Convert the model to the same data type as the image (Double)

trainer.model = trainer.model.double()

# Perform the prediction

with torch.no\_grad():

predicted\_probs = trainer.model(img)

predicted\_class = torch.argmax(predicted\_probs, dim=1).item()

confidence = torch.max(predicted\_probs).item()

# Get the predicted label

predicted\_label = id2label[predicted\_class]

# Display the image

plt.imshow(plt.imread(image\_path))

plt.title(f'Predicted Class: {predicted\_class} - {predicted\_label}\nConfidence: {confidence:.2f}')

plt.axis('off')

plt.show()



**Conclusion:**

1. The "glaucoma" class has a precision of 0.90, recall of 0.83, and F1-score of 0.86. This suggests that

the model performs well in correctly identifying glaucoma cases, but there may be some false

negatives. The "normal" class has a precision of 0.85, recall of 0.90, and F1-score of 0.88. The model

performs well in both precision and recall for normal cases. The "diabetic\_retinopathy" class has high

precision, recall, and F1-score of 0.99. This indicates the model's excellent performance in correctly

identifying cases of diabetic retinopathy. The "cataract" class also has high precision, recall, and F1

score of 0.95 and above, indicating accurate identification of cataract cases.

2. The overall accuracy of the model is 0.92, indicating the percentage of correctly predicted

instances across all classes.

3. In summary, the model shows strong performance in correctly identifying cases of diabetic

retinopathy and cataract, while slightly lower precision and recall are observed for glaucoma.