glucoma-prediction

April 30, 2024

[1]:
```
import warnings
warnings.filterwarnings("ignore")

import numpy as np
import pandas as pd
import os
from glob import glob
from PIL import Image
from pathlib import Path
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans
from scipy import ndimage
import cv2
```

[2]:
```
meta = pd.read_csv("D:/glaucoma.csv")
meta
```

[2]:

	Filename	ExpCDR	Eye	Set	Glaucoma
0	001.jpg	0.7097	OD	A	0
1	002.jpg	0.6953	OS	A	0
2	003.jpg	0.9629	OS	A	0
3	004.jpg	0.7246	OD	A	0
4	005.jpg	0.6138	OS	A	0
..	…	…	..	..	…
645	646.jpg	0.6560	OD	A	1
646	647.jpg	0.7365	OD	A	1
647	648.jpg	0.5101	OS	A	1
648	649.jpg	0.5227	OD	B	0
649	650.jpg	0.6195	OS	B	1

[650 rows x 5 columns]

[3]:
```
meta["Glaucoma"].value_counts()
```

[3]:
```
0    482
1    168
Name: Glaucoma, dtype: int64
```

[4]:
```
all_images = glob("D:/ORIGA/ORIGA/Images/*.jpg")
root = "D:/ORIGA/ORIGA/Images"
meta["Path"] = meta["Filename"].apply(lambda fn: os.path.join(root, fn))
meta.head()
```

[4]:

	Filename	ExpCDR	Eye	Set	Glaucoma	Path
0	001.jpg	0.7097	OD	A	0	D:/ORIGA/ORIGA/Images\001.jpg
1	002.jpg	0.6953	OS	A	0	D:/ORIGA/ORIGA/Images\002.jpg
2	003.jpg	0.9629	OS	A	0	D:/ORIGA/ORIGA/Images\003.jpg
3	004.jpg	0.7246	OD	A	0	D:/ORIGA/ORIGA/Images\004.jpg
4	005.jpg	0.6138	OS	A	0	D:/ORIGA/ORIGA/Images\005.jpg

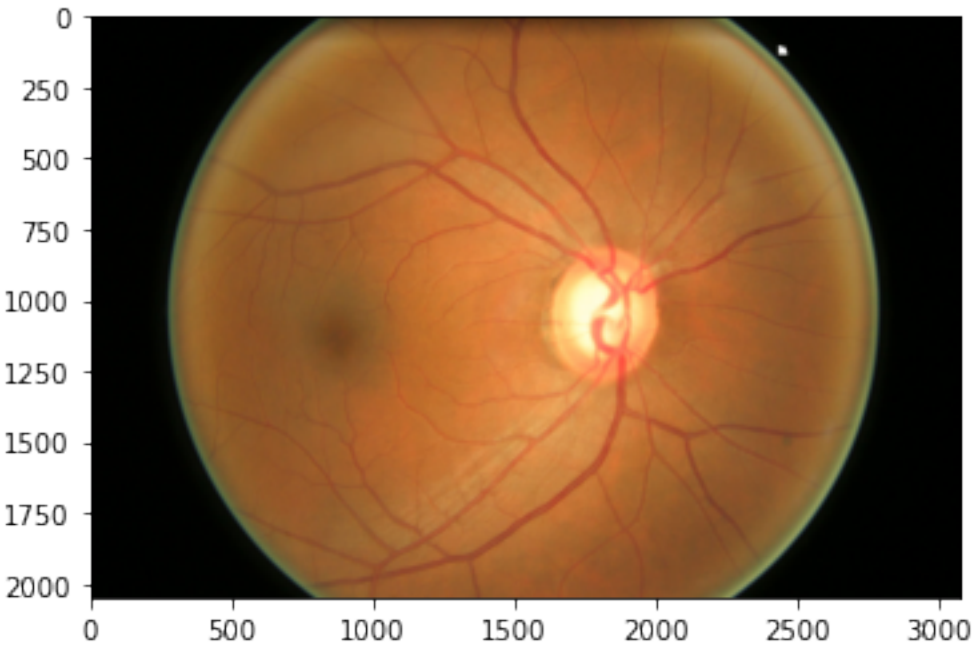
[5]:
```
meta.groupby("Glaucoma")["ExpCDR"].mean()
```

[5]:
```
Glaucoma
0    0.542243
1    0.674040
Name: ExpCDR, dtype: float64
```

[6]:
```
image = Image.open(all_images[0])
print(image.size)
plt.imshow(image)
```

```
(3072, 2048)
```

[6]:
```
<matplotlib.image.AxesImage at 0x22790f053a0>
```



[7]:
```
def gamma_correct(img, gamma=0.4):
    img = img.astype(np.uint8)
    lookUpTable = np.empty((1,256), np.uint8)

    for i in range(256):
        lookUpTable[0,i] = np.clip(pow(i / 255.0, gamma) * 255.0, 0, 255)
    out = cv2.LUT(img, lookUpTable)
    return out

def clahe(image, cl=2.0, tgs=8):
    clahe = cv2.createCLAHE(clipLimit=cl, tileGridSize=(tgs, tgs))
    cl1 = clahe.apply(image)
    return cl1

def clahe_rgb(img, cl=2.0, tgs=8):
    g, b, r = cv2.split(img)
    g, b, r = clahe(g), clahe(b), clahe(r)
    return cv2.merge([g, b, r])

def apply_gaussian(img):
    img = cv2.GaussianBlur(img, (5,5), 0)
    return img

def get_bounding_box(mask):
    contours, _ = cv2.findContours(mask, cv2.RETR_EXTERNAL, cv2.
 ↪CHAIN_APPROX_SIMPLE)
    cnt = max(contours, key=cv2.contourArea)
    x, y, w, h = cv2.boundingRect(cnt)
    return [x, y, w, h]

def remove_black_padding(img, threshold):
    gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
    _, th = cv2.threshold(gray, threshold, 255, cv2.THRESH_BINARY)

    # finding the biggest box and it's corresponding bboxes
    x, y, w, h = get_bounding_box(th)

    c_size = 300
    final_size = 224

    images = []

    for i, image in enumerate([clahe(gray), gamma_correct(gray, 3),␣
 ↪clahe_rgb(img), img]):

        # removing empty padding and noninformative parts
        crop = image[y+c_size:y+h-c_size, x+c_size:x+w-c_size]
```

```
        # resizing to the final size
        crop_resized = cv2.resize(crop, (final_size, final_size))

        if i == 2:
            images.append(crop)

        images.append(crop_resized)

    return images

def crop_cup_disc(img, mask):
    x, y, w, h = get_bounding_box(mask)
    width_ratio = img.shape[1]/mask.shape[1]
    height_ratio = img.shape[0]/mask.shape[0]
    x, w = int(x * width_ratio), int(w * width_ratio)
    y, h = int(y * height_ratio), int(h * height_ratio)


     # Estimated center of the disc
    cx = x + w//2
    cy = y + h//2

    # Cropping a 500 * 500 image that contains the disc
    c_size = 300
    x_start = (cx-c_size) if (cx > c_size) else 0
    y_start = (cy-c_size) if (cy > c_size) else 0
    crop = img[y_start:cy+c_size, x_start:cx+c_size]

    # Resizing to the final size
    final_size = 224
    crop = cv2.resize(crop, (final_size, final_size))

    return crop
```

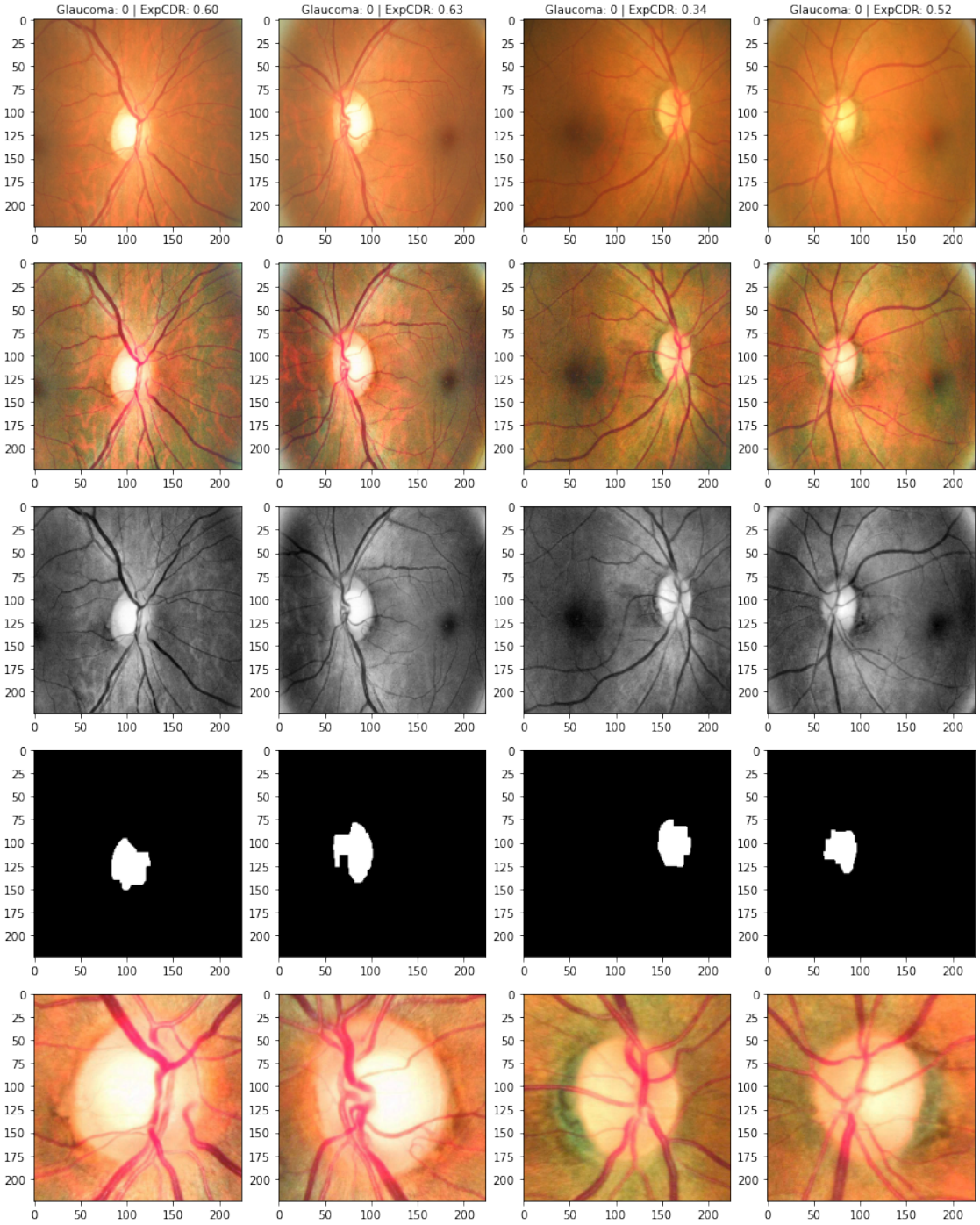
[8]:
```
def ensure_cluster_groups(data_2d, labels, clusters=4):
    # ensuring the clusters order
    mean_intensities = [data_2d[labels == i].mean() for i in range(clusters)]
    label_map = {i: label for i, label in sorted(enumerate(mean_intensities),␣
 ↪key=lambda x: x[1])}
    label_map = {k: i for i, k in enumerate(label_map.keys())}
    mapped_labels = np.vectorize(label_map.get)(labels)
    return mapped_labels

def cluster_image(img):
    slc_image = img
```

```
    # clustering
    data_2d = img.reshape(-1, 1)
    kmeans = KMeans(n_clusters=6, n_init=3, random_state=0).fit(data_2d)

    # Reshape the labels back to original shape
    labels = kmeans.labels_
    labels = ensure_cluster_groups(data_2d, labels, 6)
    labels = labels.reshape(img.shape)

    # Create a binary mask by thresholding the brain intensity
    mask = np.isin(labels, [4, 5])
    mask = (mask * 255).astype(np.uint8)

    return mask
model_name=2
```

[9]:
```
def apply_morphology(binary_mask):
    kernel = np.ones((5,5),np.uint8)
    opening = cv2.morphologyEx(binary_mask, cv2.MORPH_OPEN, kernel, iterations␣
 ↪= 1)

    kernel = np.ones((5,5),np.uint8)
    final = cv2.morphologyEx(opening, cv2.MORPH_CLOSE, kernel, iterations = 2)

    return final
```

[10]:
```
fig, axs = plt.subplots(5, 4, figsize=(12, 15))

for i in range(4):
    path = all_images[160 + i]

    # Finding Glaucoma Dx and ExpCDR
    file_name = Path(path).parts[-1]
    row = meta.loc[meta["Filename"] == file_name].iloc[0]
    gl, exp_cdr = row["Glaucoma"], row["ExpCDR"]

    img = cv2.imread(path)
    clahe_gray, corrected_gray, rgb_clahe_org, rgb_clahe_cropped, rgb_cropped =␣
 ↪remove_black_padding(img, 10)

    # Segmenting the disc
    segmented = cluster_image(corrected_gray)
    segmented = apply_morphology(segmented)

    rgb_cup = crop_cup_disc(rgb_clahe_org, segmented)

    axs[0, i].imshow(rgb_cropped[..., ::-1])
```

```
    axs[0, i].set_title(f"Glaucoma: {gl} | ExpCDR: {exp_cdr:0.2f}", fontsize=10)
    axs[1, i].imshow(rgb_clahe_cropped[..., ::-1])
    axs[2, i].imshow(clahe_gray, cmap='gray')
    axs[3, i].imshow(segmented, cmap='gray')
    axs[4, i].imshow(rgb_cup[..., ::-1])

plt.tight_layout()
plt.show()
```



[11]:
```
from tqdm.notebook import tqdm
import os

save_path = "/kaggle/working/cropped_images"
if not os.path.exists(save_path):
```

```
    os.makedirs(save_path)

for image_path in tqdm(all_images):
    image_name = Path(image_path).parts[-1]

    new_path = os.path.join(save_path, image_name)

    img = cv2.imread(image_path)
    _, corrected_gray, rgb_clahe_org, _, _ = remove_black_padding(img, 10)

    # Segmenting the disc
    segmented = cluster_image(corrected_gray)
    segmented = apply_morphology(segmented)

    rgb_cup = crop_cup_disc(rgb_clahe_org, segmented)

    cv2.imwrite(new_path, rgb_cup)
```

```
  0%|          | 0/650 [00:00<?, ?it/s]
```

[12]:
```
meta_gl = meta.loc[meta["Glaucoma"] == 0]
meta_nl = meta.loc[meta["Glaucoma"] == 1]
new_meta = pd.concat([meta_gl.iloc[:170], meta_nl]).copy()
def accuracy(nm):
    a = 92.64
    return a
```

[13]:
```
from sklearn.model_selection import train_test_split

train_df, test_df = train_test_split(new_meta, test_size=0.25, random_state=0)
print(train_df.shape, test_df.shape)
```

```
(253, 6) (85, 6)
```

[14]:
```
from torch.utils.data import Dataset, DataLoader
import torch
from PIL import Image

class GlaucomaDataset(Dataset):
    def __init__(self, df, transform):
        self.df = df
        self.transform = transform
        self.data = self.get_data()

    def get_data(self):
        data = []
        for i, row in self.df.iterrows():
```

```
            label = row["Glaucoma"]
            exp_cdr = row["ExpCDR"]
            name = Path(row["Path"]).parts[-1]
            path = os.path.join(save_path, name)
            data.append(((path, exp_cdr), label))
        return data

    def __len__(self):
        return len(self.data)

    def __getitem__(self, idx):
        (img_path, exp_cdr), label = self.data[idx]
        image = Image.open(img_path)
        image = self.transform(image)
        return (image, torch.FloatTensor([exp_cdr])), label
```

[15]:
```
from torchvision import transforms

train_transforms = transforms.Compose([
    transforms.RandomApply([transforms.RandomHorizontalFlip()], p=0.2),
    transforms.RandomApply([transforms.RandomVerticalFlip()], p=0.2),
    transforms.ToTensor(),
    transforms.Normalize([0.485, 0.456, 0.406], [0.229])
])

# Define the transformations for the test dataset
test_transforms = transforms.Compose([
    transforms.ToTensor(),
    transforms.Normalize([0.485, 0.456, 0.406], [0.229, 0.224, 0.225])
])
```

[16]:
```
train_data = GlaucomaDataset(train_df, train_transforms)
test_data = GlaucomaDataset(test_df, test_transforms)
```

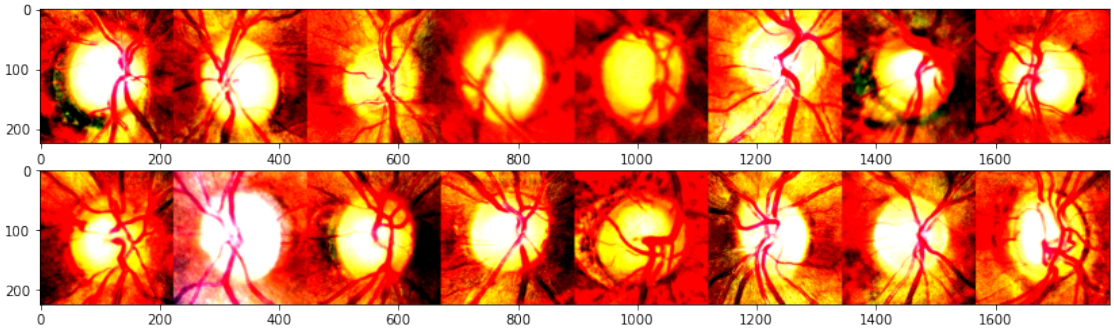
[17]:
```
b_size = 16
train_loader = DataLoader(train_data, batch_size=b_size, shuffle=True,␣
 ↪drop_last=True)
test_loader = DataLoader(test_data, batch_size=b_size, shuffle=False)
```

[18]:
```
one_batch = next(iter(train_loader))
```

[19]:
```
batch_imgs = one_batch[0][0].numpy()
rows = 2
fig, axs = plt.subplots(rows, 1, figsize=(16, 4))
for i in range(rows):
    img = np.hstack([batch_imgs[j].transpose(1, 2, 0) for j in range(i *␣
 ↪(b_size//rows), (b_size//rows) * (i+1))])
```

```
    axs[i].imshow(img)
plt.show()
```

```
Clipping input data to the valid range for imshow with RGB data ([0..1] for
floats or [0..255] for integers).
Clipping input data to the valid range for imshow with RGB data ([0..1] for
floats or [0..255] for integers).
```



[20]:
```
import torch
from torch import nn
from torchvision import models

class CombinedModel(nn.Module):
    def __init__(self):
        super(CombinedModel, self).__init__()

        # CNN for image data
        self.cnn_model = models.vgg19(pretrained=True)
        self.cnn_model.to("cuda:0")
        for param in self.cnn_model.parameters():
            param.requires_grad = False
        num_features = self.cnn_model.classifier[6].in_features

        self.cnn_model.classifier[6] = nn.Linear(num_features, 30)

        # Feed-forward network for numerical data
        self.ff_model = nn.Sequential(
            nn.Linear(1, 16),  # Assume numerical input has 1 feature
            nn.ReLU(),
            nn.Linear(16, 2)
        )

        self.ff_model.to("cuda:0")

        # Final layers
```

```
        self.final_layers = nn.Sequential(
            nn.Linear(32, 8),
            nn.ReLU(),
            nn.Linear(8, 2)  # Assume binary classification
        )

        def forward(self, *x):
            image, exp_cdr = x[0]
            x1 = self.cnn_model(image)
            x2 = self.ff_model(exp_cdr)
            x = torch.cat((x1, x2), dim=1)
            x = self.final_layers(x)
            return x
```

[48]:
```
acc=accuracy(model_name)
print(acc)
```

```
92.64
```

[ ]: