

What is Diabetic Retinopathy?

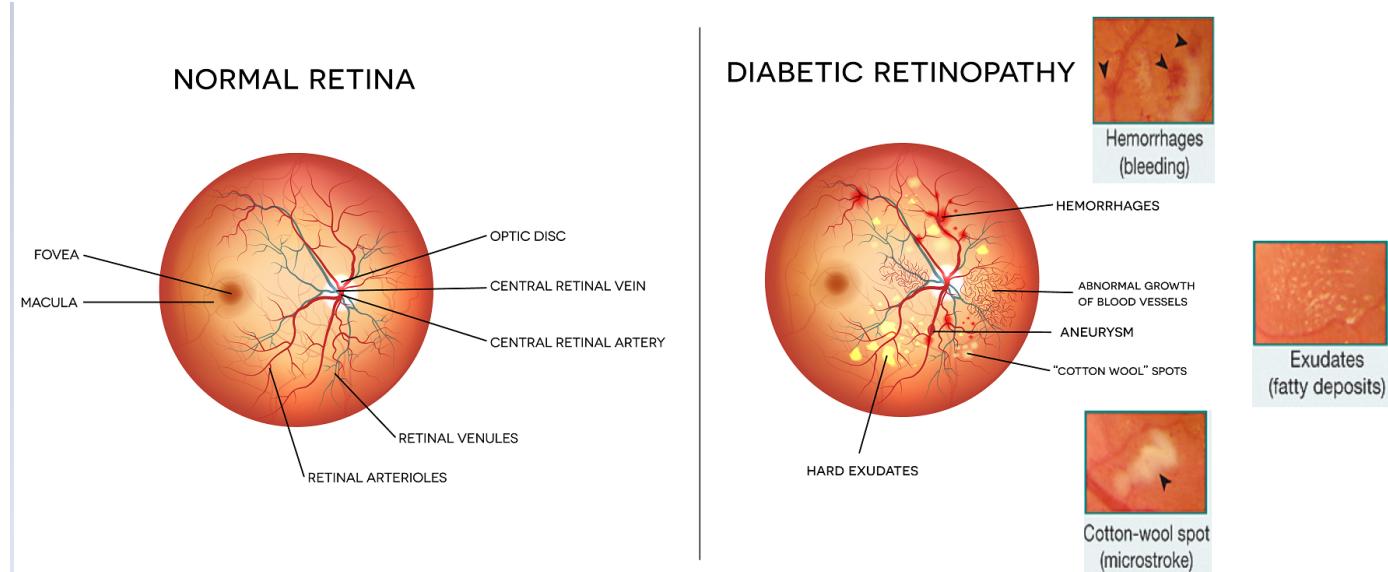
Diabetic retinopathy is the most common form of diabetic eye disease. Diabetic retinopathy usually only affects people who have had diabetes for a significant number of years.

Retinopathy can affect all diabetics and becomes particularly dangerous, increasing the risk of blindness, if it is left untreated. The condition is easiest to treat in the early stages, which is why it is important to undergo routine eye exams.

The risk of developing diabetic retinopathy is known to increase with age as well with less well controlled blood sugar and blood pressure level and occurs when the damaged blood vessels leak blood and other fluids into your retina, causing swelling and blurry vision. The blood vessels can become blocked, scar tissue can develop, and retinal detachment can eventually occur.

Hence all people with diabetes should have a dilated eye examination at least once every year to check for diabetic retinopathy.

Difference Between Normal retina and Diabetic retina:



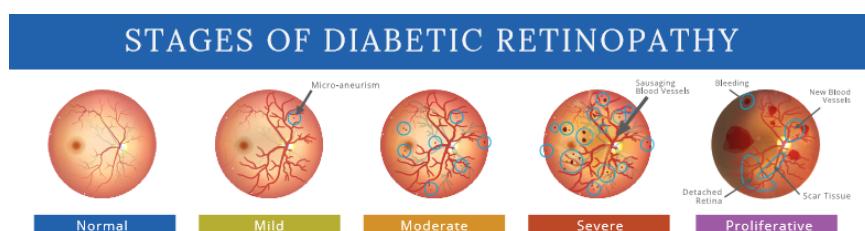
Problem Statement:

APTO2019 has provided with a large set of retina images taken using fundus photography under a variety of imaging conditions where we need to classify the severity of disease on a scale of 0 to 4. Some of the images are overexposed and some are underexposed and taken under different lightning conditions.

They want us to make an automation of the prediction of severity of disease retinopathy basing upon retinal scanned images from a clinic.

A clinician has rated each image for the severity of diabetic retinopathy on a scale of 0 to 4:

It is a multi-class classification problem with 5 target classes [0,1,2,3,4]



0 - No DR

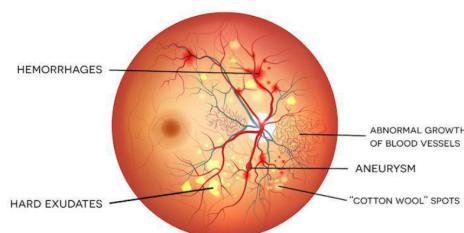
1 - Mild

2 - Moderate

3 - Severe

4 - Proliferative DR

These are the 5 things to spot on the eye ball to know whether the patient have diabetic retinopathy or not.



Read more about Disease Retinopathy [here](https://www.eyeops.com/contents/our-services/eye-diseases/diabetic-retinopathy).

Importing Dataset:

Mounting Google Drive:

```
In [ ]: from google.colab import drive  
drive.mount('/content/drive')
```

Downloading Data Using Curl/WGet:

```
In [1]: !wget --header="Host: storage.googleapis.com" --header="User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/85.0.4183.121 Safari
--2020-10-06 11:27:50-- https://storage.googleapis.com/kaggle-competitions-data/kaggle-v2/14774/875431/bundle/archive.zip?GoogleAccessId=web-data@kaggle-161607.iam.gserviceaccount.com&Expires=1602242834&Signature=Dkduy28EwVTgBz51wyUiRiQMaAzbdHjuo2FmJ4kskDK32zCwbANE0VNLTuzjS%2BA37Lhy%2BcD21KFz8oCAZm9cY8xoWjCb6YoFrt515zDvuScP2y1Z2f%2Byo9v17odKiYy0sxOPonostErAbphi0VpLo97Esf91F3MkMLFT81YgCu2BtfqyZnKReTHPr1MkquGnhMct%2B3Zw0zQ0Ck1ZVhcrFp91kQ4qjYW5fwQr8MS05uBNHjTWUFZS02gvV4Sn%2FrSQGgw90EK6kJslIezhBuMp7NRanCOAJWIBj1Nq248NNK06urt5Oyp5LbnSv%2Bht3q7E4z%2BN%2Fz2Pw%3D%3D&response-content-disposition=attachment%3B+filename%3Daptos2019-blindness-detection.zip" (https://storage.googleapis.com/kaggle-competitions-data/kaggle-v2/14774/875431/bundle/archive.zip?GoogleAccessId=web-data@kaggle-161607.iam.gserviceaccount.com&Expires=1602242834&Signature=Dkduy28EwVTgBz51wyUiRiQMaAzbdHjuo2FmJ4kskDK32zCwbANE0VNLTuzjS%2BA37Lhy%2BcD21KFz8oCAZm9cY8xoWjCb6YoFrt515zDvuScP2y1Z2f%2Byo9v17odKiYy0sxOPonostErAbphi0VpLo97Esf91F3MkMLFT81YgCu2BtfqyZnKReTHPr1MkquGnhMct%2B3Zw0zQ0Ck1ZVhcrFp91kQ4qjYW5fwQr8MS05uBNHjTWUFZS02gvV4Sn%2FrSQGgw90EK6kJslIezhBuMp7NRanCOAJWIBj1Nq248NNK06urt5Oyp5LbnSv%2Bht3q7E4z%2BN%2Fz2Pw%3D%3D&response-content-disposition=attachment%3B+filename%3Daptos2019-blindness-detection.zip)
Resolving storage.googleapis.com (storage.googleapis.com)... 108.177.125.128, 74.125.204.128, 64.233.189.128, ...
Connecting to storage.googleapis.com (storage.googleapis.com)|108.177.125.128|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 10215289875 (9.5G) [application/zip]
Saving to: 'aptos2019-blindness-detection.zip'

aptos2019-blindness 100%[=====] 9.51G 53.6MB/s in 3m 31s

2020-10-06 11:31:22 (46.3 MB/s) - 'aptos2019-blindness-detection.zip' saved [10215289875/10215289875]
```

- The files in this competition occupies 9.52GB of Space.
- And it will take nearly 36 minutes for downloading the data on a normal local machine.
- Hence I am using CUrlWget (chrome extension) for faster downloading of data.
- It gives **aptos2019-blindness-detection.zip** file in 2-3 minutes. However we can extract all the files by unzipping that file using the following command in 4-5 minutes.

Unzipping Folders in .zip file:

```
In [2]: !unzip aptos2019-blindness-detection.zip
```

```
In [3]: import os
os.listdir("./content")
```

```
Out[3]: ['.config',
'aptos2019-blindness-detection.zip',
'test_images',
'train_images',
'train.csv',
'sample_submission.csv',
'drive',
'test.csv',
'sample_data']
```

Loading Dependencies:

```
In [4]: # for system settings
import os
os.chdir('/content/drive/My Drive/')
import warnings
from tqdm import tqdm
warnings.filterwarnings('ignore')
import time
# for excessing or creating tabular data
import pandas as pd
import pandas.util.testing as tm
# for matrix manipulation
import numpy as np
# for visualization
import matplotlib.pyplot as plt
import seaborn as sns
import cv2
# for splitting the data
from sklearn.model_selection import train_test_split
```

```
In [7]: train_data = pd.read_csv("/content/train.csv")
print("train.csv:")
print("Number of Training images: {}".format(train_data.shape[0]))
print(train_data.head(2),"\n")
print("-"*100)
test_data = pd.read_csv("/content/test.csv")
print("test.csv: ")
print("Number of Testing images: {}".format(test_data.shape[0]))
print(test_data.head(2))
print("\n","-"*100)
print("sample_submission.csv:")
sample_submission = pd.read_csv("/content/sample_submission.csv")
print("The format of submitting the final predictions on testing images: ")
print(sample_submission.head(5))
```

train.csv:
Number of Training images: 3662

	id_code	diagnosis
0	000c1434d8d7	2
1	001639a390f0	4

test.csv:
Number of Testing images: 1928

	id_code
0	0005cf8af6
1	003f0afcd15

sample_submission.csv:
The format of submitting the final predictions on testing images:

	id_code	diagnosis
0	0005cf8af6	0
1	003f0afcd15	0
2	006efc72b638	0
3	00836aaacf06	0
4	009245722fa4	0

The competition has given 5 files:

1. train.csv -> which contains two columns id_code (path of the training images) and diagnosis (target class)
2. test.csv -> which contains one column id_code (path of the testing images) which we need to predict the class labels.

3. train_images -> Folder which contains the training images.
4. test_images -> Folder which contains the testing images.
5. sample_submission.csv -> which contains the template of final submission file to the competition.
6. apots2019-blindness-detection.zip -> zip Folder using which we can extract all the above four files.

In [8]: `# Initialization of variables which are useful for the later tasks.`

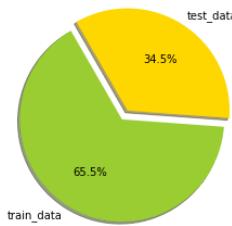
```
img_width = 512
img_height = 512
no_channels = 3
split_size = 0.15
class_labels = {0: 'No DR[0]', 1: 'Mild[1]', 2: 'Moderate[2]', 3: 'Severe[3]', 4: 'Proliferative DR[4]}'
```

Exploratory Data Analysis:

In []: `data = [len(train_data), len(test_data)]`
`print("Number of Images in train dataset: ", data[0])`
`print("Number of Images in test dataset: ", data[1])`
`labels = ['train_data', 'test_data']`
`plt.pie(data, explode = [0,0.1], labels= labels, shadow = True, colors = ['yellowgreen','gold'], autopct='%.1f%%', startangle = 120)`
`plt.title('Pie Chart Analysis of size of train and test datasets')`
`plt.axis('equal')`
`plt.show()`

Number of Images in train dataset: 3662
Number of Images in test dataset: 1928

Pie Chart Analysis of size of train and test datasets



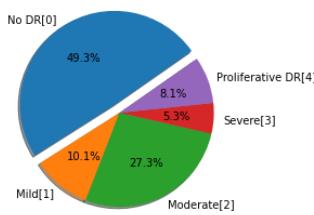
- Both the training and test folders are not too large. Which may not be sufficient for model training. Since the model training is with few images, gets overfitted too easily. So we need to come up with different data augmentation techniques and make use of pretrained models(Transfer Learning).
- Train data set is **three times** larger than test data set.

In [9]: `class_labels_ = list(set(train_data['diagnosis']))`
`print("Number of target classes: {}".format(class_labels_))`

Number of target classes: [0, 1, 2, 3, 4]

In []: `class_sizes = []`
`for i in range(0,5):`
 `class_sizes.append(list(train_data['diagnosis']).count(i))`
`labels = class_labels_.values()`
`colors = ['gold', 'yellowgreen', 'lightcoral', 'lightskyblue', 'darkgreen']`
`plt.pie(class_sizes, explode = [0.1,0,0,0,0], labels= labels, shadow = True, autopct='%.1f%%', startangle = 35)`
`plt.title('Pie Chart Analysis of Number of Images on each target label:')`
`plt.show()`

Pie Chart Analysis of Number of Images on each target label:



- We can see that the data is heavily imbalanced. There are ten times more images with No DR condition than the images with Severe condition.
- Adding class_weights may solve our problem.

In [10]: `def image_analysis(dataframe, path):`
 `width_range = []`
 `height_range = []`
 `for i in range(dataframe.shape[0]):`
 `img = cv2.imread(path+dataframe.iloc[i]['id_code']+'.png')`
 `height, width, _ = img.shape`
 `width_range.append(width)`
 `height_range.append(height)`
 `return width_range, height_range`

```
width_range, height_range = image_analysis(train_data, '/content/train_images/')
```

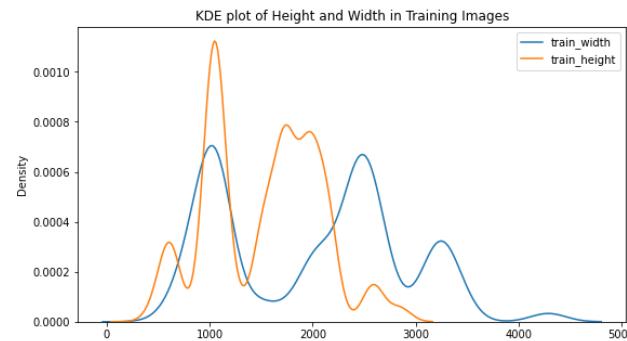
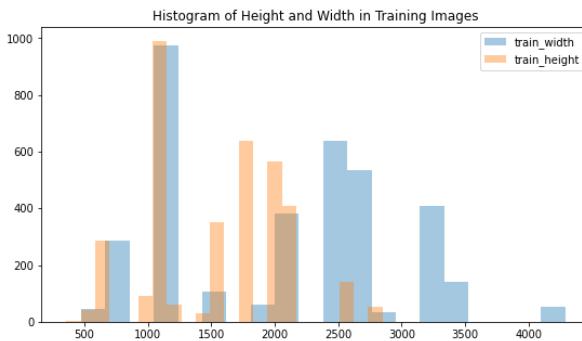
```
In [ ]:
avg_width = sum(width_range)/len(width_range)
avg_height = sum(height_range)/len(height_range)
max_width = max(width_range)
max_height = max(height_range)
min_width = min(width_range)
min_height = min(height_range)
print("Average width of images in training set: {}".format(int(avg_width)))
print("Average height of images in training set: {}".format(int(avg_height)))
print(" "*100)
print("Maximum width of images in training set: {}".format(max_width))
print("Maximum height of images in training set: {}".format(max_height))
print(" "*100)
print("Minimum width of images in training set: {}".format(min_width))
print("Minimum height of images in training set: {}".format(min_height))
```

Average width of images in training set: 2015
 Average height of images in training set: 1526

 Maximum width of images in training set: 4288
 Maximum height of images in training set: 2848

 Minimum width of images in training set: 474
 Minimum height of images in training set: 358

```
In [ ]:
plt.figure(figsize = (20,5))
plt.subplot(1,2,1)
sns.distplot(width_range, kde = False, label = 'train_width')
sns.distplot(height_range, kde = False, label = 'train_height')
plt.legend()
plt.title("Histogram of Height and Width in Training Images")
plt.subplot(1,2,2)
sns.kdeplot(width_range, label = 'train_width')
sns.kdeplot(height_range, label = 'train_height')
plt.legend()
plt.title('KDE plot of Height and Width in Training Images')
plt.show()
```



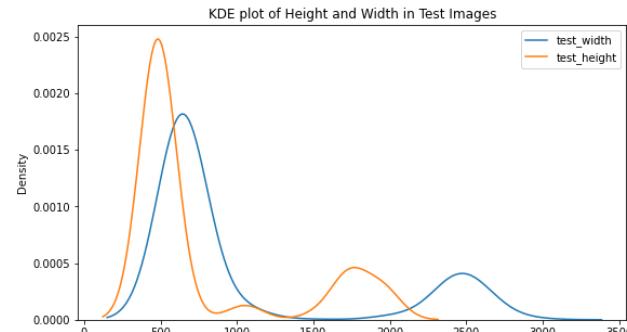
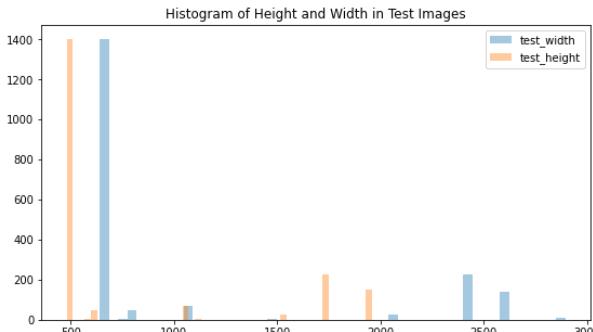
```
In [ ]:
width_range_test, height_range_test = image_analysis(test_data, '/content/test_images/')
avg_width = sum(width_range_test)/len(width_range_test)
avg_height = sum(height_range_test)/len(height_range_test)
max_width = max(width_range_test)
max_height = max(height_range_test)
min_width = min(width_range_test)
min_height = min(height_range_test)
print("Average width of images in training set: {}".format(int(avg_width)))
print("Average height of images in training set: {}".format(int(avg_height)))
print(" "*100)
print("Maximum width of images in test set: {}".format(max_width))
print("Maximum height of images in test set: {}".format(max_height))
print(" "*100)
print("Minimum width of images in test set: {}".format(min_width))
print("Minimum height of images in test set: {}".format(min_height))
```

Average width of images in training set: 1043
 Average height of images in training set: 783

 Maximum width of images in test set: 2896
 Maximum height of images in test set: 1958

 Minimum width of images in test set: 640
 Minimum height of images in test set: 480

```
In [ ]:
plt.figure(figsize = (20,5))
plt.subplot(1,2,1)
sns.distplot(width_range_test, kde = False, label = 'test_width')
sns.distplot(height_range_test, kde = False, label = 'test_height')
plt.legend()
plt.title("Histogram of Height and Width in Test Images")
plt.subplot(1,2,2)
sns.kdeplot(width_range_test, label = 'test_width')
sns.kdeplot(height_range_test, label = 'test_height')
plt.legend()
plt.title('KDE plot of Height and Width in Test Images')
plt.show()
```



- Their is no standard shape for images in both train and test datasets.
- We can see that their are different distribution in (train_width,test_width) and (train_height, test_height).
- We need to bring all the images into standard resolution.

Splitting the Dataset:

```
In [ ]: def splitting_data(train_data, size, is_split = True):
    """
        This function splits the given data into train and validation sets basing on size for validation.
        Args : df - (dataframe) through which splitting is performed
        size - (Integer) test_size -> percentage of data for validation set
        is_split = (boolean) returns train and validation if it is True , otherwise it simply returns the train data
        Outputs : (Series Object) train and validation sets of data
    """
    try:
        if is_split:
            data = train_data['id_code']
            labels = train_data['diagnosis']
            train_x, validation_x, train_labels, validation_labels = train_test_split(data, labels, stratify=labels, shuffle=True, test_size=size)
            print("Training data: {} {}".format(train_x.shape, train_labels.shape))
            print("Validation data: {} {}".format(validation_x.shape, validation_labels.shape))
            return train_x, train_labels, validation_x, validation_labels
        else:
            return train_data['id_code'], train_data['diagnosis'], [], []
    except:
        print("Error: Invalid file format, Function argument requires .csv file!!!")
```

```
In [ ]: train_x, train_labels, validation_x, validation_labels = splitting_data(train_data, split_size) # function calling
<
Training data: (3112,) (3112,)
Validation data: (550,) (550,)
```

Saving files into folders for future use:

```
In [ ]: train = pd.DataFrame(train_x, columns = ['id_code'])
train['diagnosis'] = train_labels
train.to_csv('/content/drive/My Drive/training.csv', index = False)
validation = pd.DataFrame(validation_x, columns = ['id_code'])
validation['diagnosis'] = validation_labels
validation.to_csv('/content/drive/My Drive/validation.csv', index = False)
```

```
In [ ]: test = pd.read_csv('/content/test.csv')
test.to_csv('/content/drive/My Drive/test.csv', index = False)
```

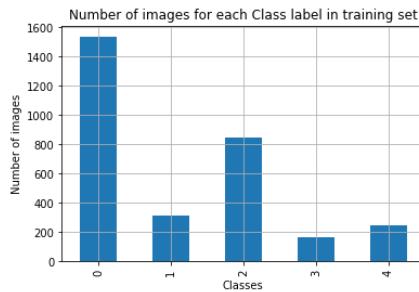
```
In [11]: train = pd.read_csv("/content/drive/My Drive/training.csv")
validation = pd.read_csv("/content/drive/My Drive/validation.csv")
train_x = train['id_code']
train_labels = train['diagnosis']
validation_x = validation['id_code']
validation_labels = validation['diagnosis']
```

Target Labels Analysis on Train and validation sets:

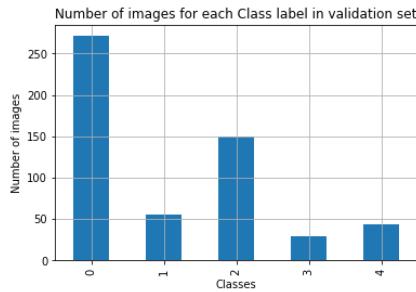
```
In [12]: def class_analysis(labels, d_set):
    """
        This function plots the histogram of class labels for given set of labels.
        Args : labels - (Series object) which contains the class_labels of train or validation sets.
        d_set - (String) which helps to known whether it is a train or validation set.
        Output : None - this function doesn't return anything
    """
    if d_set == 'training': print("-"*100,'\\n')
    counter = labels.value_counts().sort_index()
    counter.plot(kind = 'bar')
    plt.title('Number of images for each Class label in {} set'.format(d_set))
    plt.xlabel('Classes')
    plt.ylabel('Number of images')
    plt.grid()
    plt.show()
    iter=0

    for i in list(set(labels)):
        percentage = list(labels).count(i)/len(list(labels))
        print("Number of images in class - {} ({}), nearly {} % of total data".format(i,class_labels[i],np.round(percentage*100,4)))
        iter+=1
    if d_set == 'training':
        print("\\n","-*100,"\\n")
    if d_set == 'validation': print("-"*100,'\\n')
```

```
In [13]: class_analysis(train_labels,'training')
class_analysis(validation_labels,'validation')
```



Number of images in class - 0 (No DR[0]) , nearly 49.2931 % of total data
Number of images in class - 1 (Mild[1]) , nearly 10.09 % of total data
Number of images in class - 2 (Moderate[2]) , nearly 27.2815 % of total data
Number of images in class - 3 (Severe[3]) , nearly 5.2699 % of total data
Number of images in class - 4 (Proliferative DR[4]) , nearly 8.0656 % of total data



Number of images in class - 0 (No DR[0]) , nearly 49.2727 % of total data
Number of images in class - 1 (Mild[1]) , nearly 10.1818 % of total data
Number of images in class - 2 (Moderate[2]) , nearly 27.2727 % of total data
Number of images in class - 3 (Severe[3]) , nearly 5.2727 % of total data
Number of images in class - 4 (Proliferative DR[4]) , nearly 8.0 % of total data

Image Preprocessing:

Reference: <https://www.kaggle.com/ratthachat/aptos-eye-preprocessing-in-diabetic-retinopathy>.

```
In [14]: class ImageProcessing:
    def __init__(self, img_height, img_width, no_channels, tol=7, sigmaX=8):
        ''' Initialization of variables'''

        self.img_height = img_height
        self.img_width = img_width
        self.no_channels = no_channels
        self.tol = tol
        self.sigmaX = sigmaX

    def cropping_2D(self, img, is_cropping = False):
        '''This function is used for Cropping the extra dark part of the GRAY images'''

        mask = img>self.tol
        return img[np.ix_(mask.any(1),mask.any(0))]

    def cropping_3D(self, img, is_cropping = False):
        '''This function is used for Cropping the extra dark part of the RGB images'''

        gray_img = cv2.cvtColor(img, cv2.COLOR_RGB2GRAY)
        mask = gray_img>self.tol

        check_shape = img[:, :, 0][np.ix_(mask.any(1),mask.any(0))].shape[0]
        if (check_shape == 0): # if image is too dark we return the image
            return img
        else:
            img1 = img[:, :, 0][np.ix_(mask.any(1),mask.any(0))] #for channel_1 (R)
            img2 = img[:, :, 1][np.ix_(mask.any(1),mask.any(0))] #for channel_2 (G)
            img3 = img[:, :, 2][np.ix_(mask.any(1),mask.any(0))] #for channel_3 (B)
            img = np.stack([img1,img2,img3],axis=-1)
        return img

    def Gaussian_blur(self, img, is_gaussianblur = False):
        '''This function is used for adding Gaussian blur (image smoothing technique) which helps in reducing noise in the image.'''

        img = cv2.addWeighted(img,4,cv2.GaussianBlur(img,(0,0),self.sigmaX),-4,128)
        return img

    def draw_circle(self,img, is_drawcircle = True):
        '''This function is used for drawing a circle from the center of the image.'''

        x = int(self.img_width/2)
        y = int(self.img_height/2)
        r = np.amin((x,y)) # finding radius to draw a circle from the center of the image
        circle_img = np.zeros((img_height, img_width), np.uint8)
        cv2.circle(circle_img, (x,y), int(r), 1, thickness=-1)
        img = cv2.bitwise_and(img, img, mask=circle_img)
        return img

    def image_preprocessing(self, img, is_cropping = True, is_gaussianblur = True):
        """
        This function takes an image -> crops the extra dark part, resizes, draw a circle on it, and finally adds a gaussian blur to the images
        Args : image - (numpy.ndarray) an image which we need to process
               cropping - (boolean) whether to perform cropping of extra part(True by Default) or not(False)
               gaussian.blur - (boolean) whether to apply gaussian blur to an image(True by Default) or not(False)
        Output : (numpy.ndarray) preprocessed image
        """

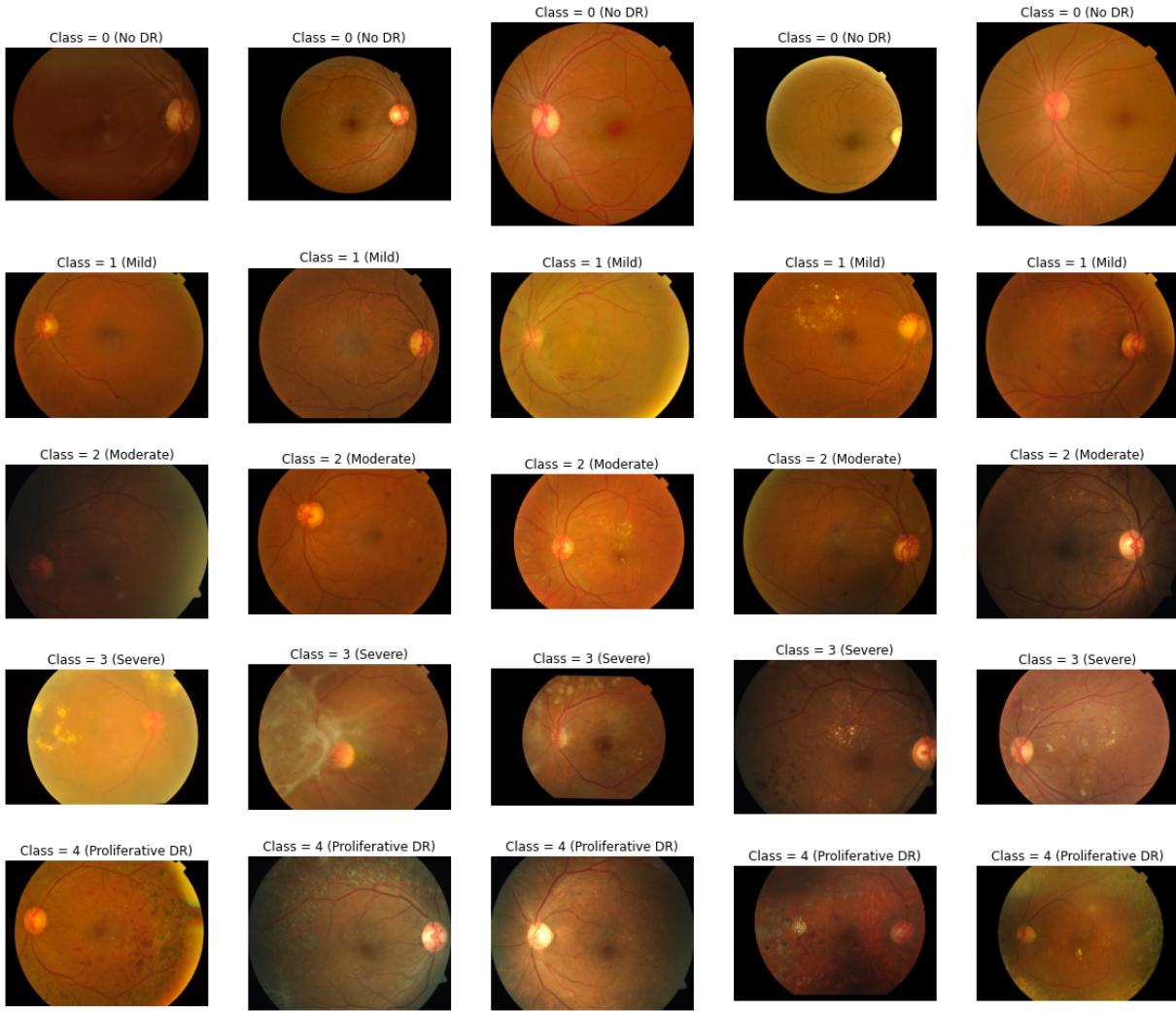
        if img.ndim == 2:
            img = self.cropping_2D(img, is_cropping) #calling cropping_2D for a GRAY image
        else:
            img = self.cropping_3D(img, is_cropping) #calling cropping_3D for a RGB image
        img = cv2.resize(img, (self.img_height, self.img_width)) # resizing the image with specified values
        img = self.draw_circle(img) #calling draw_circle
        img = self.Gaussian_blur(img, is_gaussianblur) #calling Gaussian_blur
        return img
```

```
In [15]: def images_per_class(class_labels,num, data_x , is_preprocess = False):

    """
    This function plots "num" number of images per each class
    Args : class_labels - (Series Object) which contains the class_labels of train or validation sets.
           num - (Integer) sample number of images to be plot per each class
           data_x - (Series Object) which contains the id_code of each point in train or validation sets.
           is_preprocess - (boolean) whether to perform image processing(True) on image or not(False by Default)
    Output : None - this function doesn't return anything.
    """

    # class_labels num data_x data_y
    labels = list(set(class_labels))
    classes = ['No DR', 'Mild', 'Moderate', 'Severe', 'Proliferative DR']
    iter=0
    for i in labels:
        j=1
        plt.figure(figsize=(20,5))
        for row in range(len(data_x)):
            if class_labels.iloc[row] == i:
                if is_preprocess == False:plt.subplot(1,num,j)
                else: plt.subplot(1,num*2,j)
                img = cv2.imread('/content/train_images/*data_x.iloc[row]+.png')
                img1 = cv2.cvtColor(img, cv2.COLOR_RGB2BGR)
                plt.imshow(img1)
                plt.axis('off')
                plt.title("Class = {} ({})".format(class_labels.iloc[row],classes[iter]))
                j+=1
                if is_preprocess == True:
                    obj = ImageProcessing(img_width,img_height,no_channels,sigmaX=14)
                    image = obj.image_preprocessing(img)
                    plt.subplot(1,num*2,j)
                    plt.imshow(image)
                    plt.axis('off')
                    plt.title('=> After Image Processing')
                    j+=1
                if is_preprocess == False and j>num: break
                elif is_preprocess == True and j>num*2: break
        iter+=1
    plt.show()
```

In []: `images_per_class(train_labels,5,train_x, False) #printing 5 random images per each class.`



Exploring the Dataset:

Each row depicts the severity level of the disease retinopathy.

There are four major problems in our dataset which will make our model difficult to spot the identities.

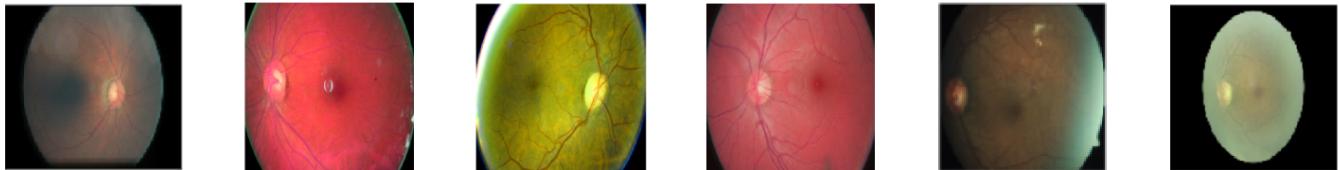
Images with dark color illumination:

Some of the images are very dark which will mislead our model towards incorrect classification.



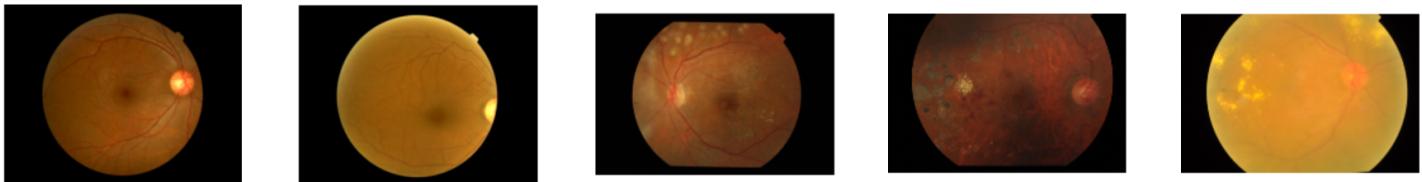
Images with different color inversion:

Here some images are different in color shades from the rest of the images like blue, pink, green.



Images with uninformative extra dark black pixels

This is the major issue because while resizing the image most of the informative will become small.



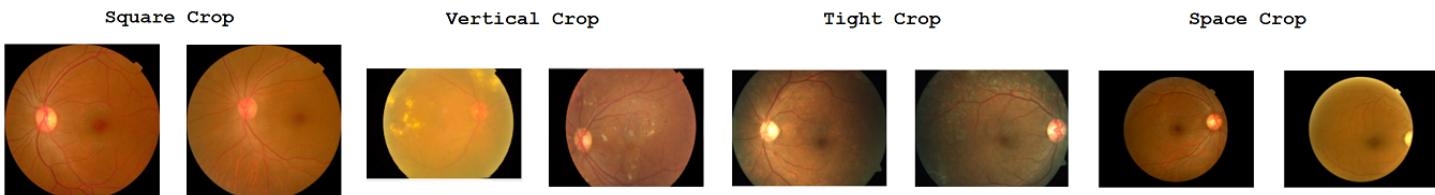
Images with irregular cropping

Their are four different types of cropping in the dataset
 i) Square crop: square image with No cropping needed

ii) vertical crop: rectangular image with lossy vertical cropping.

iii) tight crop: images with tight cropping.(lossy vertical and horizontal cropping)

iv) space crop: images with more uninformative pixels.



Most of the test images are type-iii while train images are type-iii and type-i

```
In [16]: def plotting(img, title,i):
    """
    This function is used for subplots
    Args: img (numpy.ndarray) - image we need to plot
          title(string) - title of the plot
          i (integer) - column number
    output: None - this function doesn't return anything.
    """
    plt.subplot(1,5,i)
    plt.imshow(img)
    plt.axis('off')
    plt.title(title)
```

```
In [ ]: obj1 = ImageProcessing(img_width,img_height, no_channels, sigmaX = 14)
img = '/content/train_images/201f882365d3.png' #random train image
img = cv2.imread(img)
img1 = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
plt.figure(figsize=(20,5))
plotting(img1,'Before Image Processing:',1)
img1 = obj1.cropping_3D(img1)
plotting(img1,'Step-1: (Cropping extra dark pixels)',2)
img1 = cv2.resize(img1, (img_height,img_width))
plotting(img1,'Step-2: (Resizing the image)',3)
img1 = obj1.draw_circle(img1)
plotting(img1,'Step-3: (Drawing circle)',4)
img = obj1.image_preprocessing(img, 'True')
plotting(img,'Step-4: (Adding gaussian blur)',5)
```



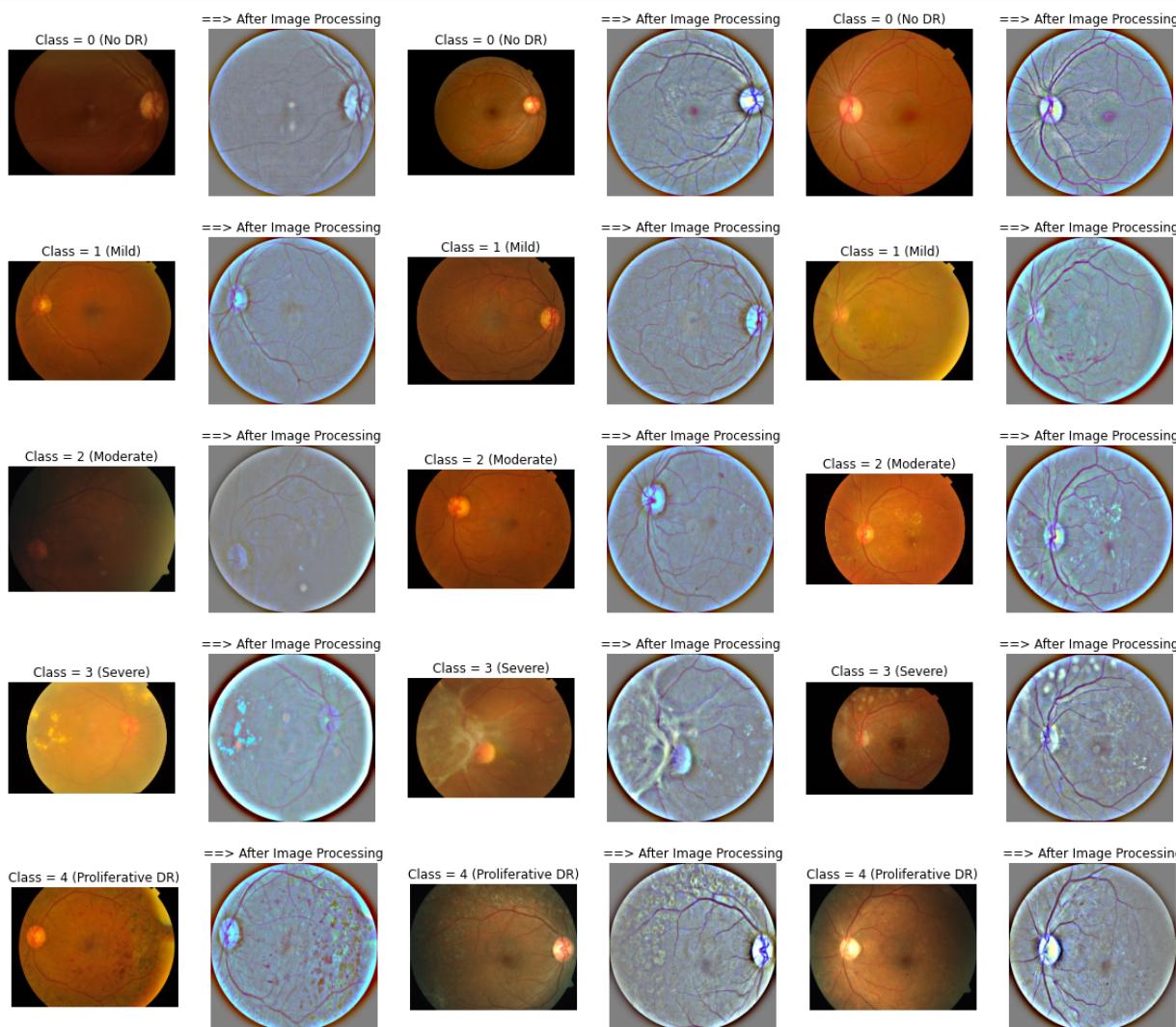
Steps of Image Preprocessing:

Above preprocessing of images is done in four steps.

- **Step-1:** All the images contains black background. And we can observe some of the images have extra dark pixels at the sides of the images. So in step-1 involves cropping of those extra dark pixels.
- **Step-2:** We can see that their is no fixed size for image width and image height. Hence we do resizing of an image in step-2.
- **Step-3:** Lot of images are shrinked and some images have circular shape and some are cropped at edges. Hence in step-3 we are drawing a circle from the center to give a similar shape to all the images.
- **Step-4:** As we can observe most of the images are taken in different resolutions. Where some are taken in lighting conditions and some in dark. we are adding a smoothing technique in order to remove the noise in images by using gaussian blur to the images. This gives our final preprocessed image.

Plotting sample images of each class before and after Image Preprocessing:

```
In [ ]: images_per_class(train_labels,3,train_x,True)
```



I tried using multiple values of sigmaX and the value = 14 works best for this case.

Converting Images to Vectors and storing it into files:

- Image preprocessing is a time taking process. Hence I am storing preprocessed images into a vector to save time on preprocessing images, every time running the program.

Image Conversion:

```
In [17]: def image_2_vector(data, sep):  
    """  
    This function is used for Converting an images into a vector and storing it in a file (.npy) format.  
    Input: data (Series Object) - which contains the path of the images  
          sep (String) - used in file creation  
    Output: None - This function doesn't return anything.  
    """  
    start_time = time.time() # storing timestamp  
    image_vector = np.empty([len(data),img_width, img_height, no_channels], dtype = np.uint8)  
    image_processing = ImageProcessing(img_width, img_height, no_channels, sigmaX) # Object creation  
    if sep !='test':  
        c = '/content/train_images/'  
    else:  
        c = '/content/test_images/'  
    for iter,row in enumerate(tqdm(data)):  
        img_path = c+data.iloc[iter]+'.png'  
        img = cv2.imread(img_path)  
        img = image_processing.image_preprocessing(img) #calling image_processing  
        image_vector[iter,:,:,:]= img  
  
    if sep == 'training': print("\nShape of the vector:",image_vector.shape)  
    else: print("\n\nShape of the vector:",image_vector.shape)  
    print("Time taken to process the {} images: {} seconds".format(sep,np.round(time.time()-start_time,5)))  
    path = '/content/drive/My Drive/processed_images'  
    print("... Saving image_vector to {}".format(path+'/*sep))  
  
    if sep == 'training':  
        print("\n","-*100,""\n")  
        if not os.path.exists(path):  
            os.makedirs(path)  
        np.save(path+'/'+sep+'.npy', image_vector) #saving file
```

```
In [18]: sigmaX = 14
image_2_vector(train_x, "training") # function calling
image_2_vector(validation_x, "validation") #function calling
```

```
100%|██████████| 3112/3112 [09:43<00:00,  5.33it/s]
```

```
Shape of the vector: (3112, 512, 512, 3)
Time taken to process the training images: 583.81414 seconds
... Saving image_vector to /content/drive/My Drive/processed_images/training
```

```
100%|██████████| 550/550 [01:55<00:00,  4.75it/s]
```

```
Shape of the vector: (550, 512, 512, 3)
Time taken to process the validation images: 115.90104 seconds
... Saving image_vector to /content/drive/My Drive/processed_images/validation
```

```
In [ ]: test = pd.read_csv("/content/test.csv")
image_2_vector(test['id_code'], 'test') #function calling
```

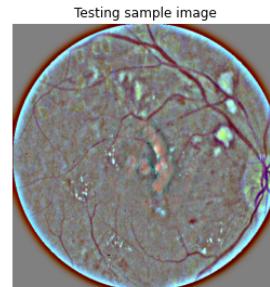
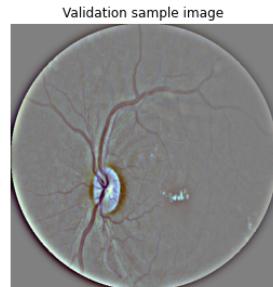
```
100%|██████████| 1928/1928 [02:39<00:00, 12.07it/s]
```

```
Shape of the vector: (1928, 512, 512, 3)
Time taken to process the test images: 159.69072 seconds
... Saving image_vector to /content/drive/My Drive/processed_images/test
```

```
In [ ]: train_x = np.load('/content/drive/My Drive/processed_images/training.npy') #training set
validation_x = np.load('/content/drive/My Drive/processed_images/validation.npy') #validation set
test_x = np.load('/content/drive/My Drive/processed_images/test.npy') #test set
```

Testing:

```
In [ ]: plt.figure(figsize=(15,5))
plt.subplot(131)
plt.imshow(train_x[8]) #random training example
plt.axis('off')
plt.title("Training sample image")
plt.subplot(132)
plt.imshow(validation_x[120]) #random validation example
plt.title("Validation sample image")
plt.axis('off')
plt.subplot(133)
plt.imshow(test_x[1200]) #random test example
plt.title("Testing sample image")
plt.axis('off')
plt.show()
```



Everything is perfect. Now we are already for model training.

```
In [ ]:
```