Eye state detection (Image Classification) using Machine Learning Techniques

Submitted in partial fulfillment of the requirements of the degree

Post Graduate Diploma in Statistical Methods and Analytics

by

Tamanna Sharma DST 20/21 025
Vaibhav Kumar Gupta DST 20/21 026

Under the guidance of
Dr. Sanjit Maitra

INDIAN STATISTICAL INSTITUTE
NORTH EAST CENTRE
JUNE 2021

Contents

1 Introduction

Image detection and classification has become a major part of Machine learning and AI in recent decades. The simple reason for this is the many applications where it can be used to solve problems of the modern society. There have been many advancements in the field of medicine; like detecting early stages of deadly diseases like cancers by analyzing patients' X-rays, or detecting stages of a disease by looking at CT scan images.

In the times of Covid-19, we have seen new developments where AI and machine learning techniques have been deployed to prevent cheating during online exams, by being able to detect face and eye movements of participants.

One of major applications has also been in the Transportation sector, where traffic signals can now capture number plates of cars on the roads and detect any rule breaking while driving. Another area of interest is, deploying AI for the prevention of road accidents. An application that people have been working on in the past few years has been Driver Drowsiness detection.

In this, images of the face of drivers are captured in real-time by pre-installed software in the vehicle itself. The algorithm in the software tries to classify whether or not the eyes of the driver are fully open or not, so as to alert them accordingly if needed. Many high-end automobile companies like Audi, BMW, and Tesla are known to have been working on such car-safety technologies. This particular application is the area that we shall be focusing on for the purpose of our project.

1.1 Motivation

The US department of Transportation reported a total of six hundred and ninety-seven deaths from road accidents due to driver drowsiness in the year 2019 alone.[1] According to the CDC, an estimated 1 in 25 adult drivers report to having fallen asleep while driving.[2] When it comes to India, a report from July,2019 by the Indian Express newspaper told that 40% of highway accidents occur due to drivers falling asleep at the wheel. Hence, this is a major issue in not only our country, but in other countries as well and provides a strong incentive to solve this problem.

[1] https://www.nhtsa.gov/risky-driving/drowsy-driving

[2] https://www.cdc.gov/sleep/features/drowsy-driving.html

1.2 About the Dataset

MRL Eye Dataset is a large-scale dataset containing human eye images of 37 different individuals (33 men and 4 women). It is provided by the Media Research Lab (MRL), which is a team of people (teachers and students) at the Department of Computer Science, Faculty of Electrical Engineering and Computer Science, VSB - Technical University of Ostrava. There are a total of 84,898 images. They are infrared, with low and high resolutions, captured in various lighting conditions and using different devices. The dataset is already divided into several categories, which makes it easy to apply in various algorithms and is suitable for training and testing classifiers.

Images are classified with the following fields:

· **Subject ID**

· **Image ID**

· **Gender** (0 – man, 1 – woman)

· **Glasses** (0 – no, 1 – yes)

· **Eye State** (0 – closed, 1 – open)

· **Reflections** (0 – none, 1 – small, 2 – big)

· **Lighting conditions** (0 – bad, 1 – good)

· **Sensor ID**

The dataset is publicly available and can be downloaded from the MRL official website. http://mrl.cs.vsb.cz/eyedataset

2 Theory

2.1 Supervised Machine Learning

Supervised learning is a technique that uses training data to 'learn' and create a certain function that produces desired output for a given input. Training data includes feature vector as well as the corresponding labels attached with each combination of the different features. A feature is any particular characteristic of the data, and can be numeric or categorical(ordinal or nominal). For instance, an image of an apple could have a feature vector describing its color, shape, i.e. "red","rounded". This is an example of a feature vector with 2 nominal features.

Overtime, the algorithm learns to recognize relationships between features and its corresponding labels, and is then used to predict output for new data, called "test" data. The algorithm tries to minimize a certain loss function in order to achieve this.

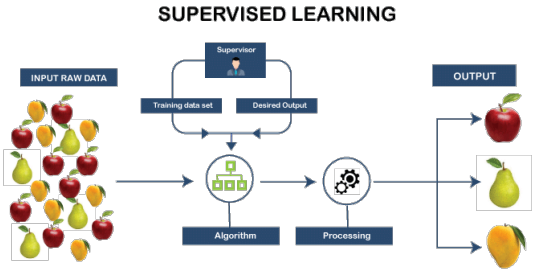


Figure 1: Source: https://www.tutorialandexample.com/

2.2 Unsupervised Machine Learning

Unsupervised learning is a machine learning technique that groups data solely based on the features of the input. It does not have any labeled output against the input. Hence, it recognizes similarities and patterns in the features of the input and classifies the data based on that. Unsupervised machine learning can be further divided into clustering and association.

2.3 Distance Metric

The distance metric is a measure that tells us how 'far' or 'close' two data points are. It is frequently used in many classification and clustering machine learning algorithms. It can be calculated for any n-dimensional input vector.

Minkowski Distance -

$$dist(x_i, x_j) = [(a_i - a_j)^h + (b_i - b_j)^h + ... + (n_i - n_j)^h]^{1/h} \quad (1)$$

where a,b,etc. are the different features of input x.

Some commonly used distance metrics are -

1. Manhattan Distance:

$$[|a_i - a_j| + |b_i - b_j| + ... + |n_i - n_j|] \quad (2)$$

2. Euclidean Distance:

$$\sqrt{(a_i - a_j)^2 + (b_i - b_j)^2 + ... + (n_i - n_j)^2} \quad (3)$$

3. Squared Euclidean Distance:

$$(a_i - a_j)^2 + (b_i - b_j)^2 + ... + (n_i - n_j)^2 \quad (4)$$

3 Exploratory Data Analysis

Before we start to apply any machine learning techniques on our dataset,it is important to get a basic understanding of what our data looks like. For this purpose, we need to understand how images are stored digitally, and what their structure looks like. Images are made up of small building blocks called pixels. They are in the shape of tiny squares, and when put together in the form a grid, give us a coherent image. Each image contains a varying number of such pixels. More the number of pixels, clearer is the image, and higher is its resolution.

Each pixel has a pixel intensity, which indicates how dark or bright it is. The scale ranges from 0 to 255, where 0 indicates darkest intensity and 255 indicates the brightest.

Since we are working with grayscale images, we used the pixel intensities to calculate the mean and variance of Closed and Open Eye images.

As we can see in Figure 2 above, image of an open eye shows the pupil of the individual. This region contributes to a larger no. of dark pixels, and hence we expect the mean of the open eye images to be lower than that of closed eyes, since darker pixel intensities have a lower numeric value. We also calculate variance to see how much spread there is in pixel intensities for each of the two categories.

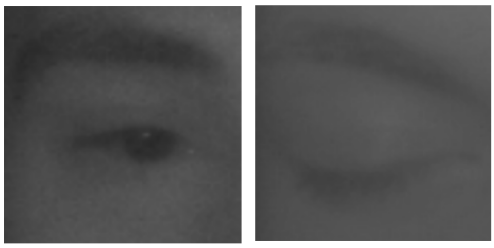


Figure 2: Image of a closed and open eye

Overall mean and variance:

· **Mean of Closed eye dataset -** 102.3293

· **Variance of Closed eye dataset -** 864.3478

· **Mean of Open eye dataset -** 83.5472

· **Variance of Open eye dataset -** 302.3102

As expected, the mean for our Closed eye images is greater than the mean of Open eye images. Both the datasets have a large variation in pixel intensities as well.

For each individual image, we get -

Figure 3: Boxplot of pixel intensities

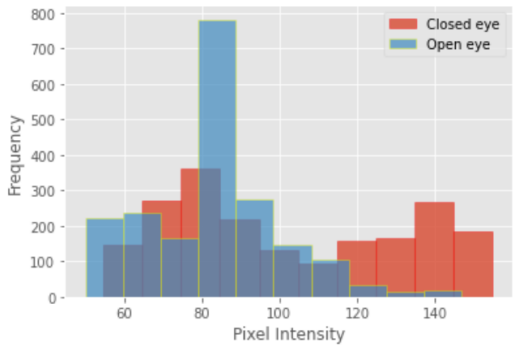


Figure 4: Histogram of pixel intensities

We can see that there is an overlap in pixel intensity values for both the

categories. Even though the overall means are different, pixel intensity values are spread over a common range of 30 - 150.

4 K-means Clustering

K-means is one of the simplest and most widely used unsupervised machine learning algorithms. It creates cluster centroids, and groups the input data according to their proximity with each of the cluster centroids, by using a certain distance metric.

Initially, K cluster centroids are randomly selected. Each data point is assigned to a particular cluster, and the centroids of each cluster are re-calculated based on the average of all the data points $\vec{x} \in c_j \ \forall$ K clusters. The process of re-assignment runs until minimum re-assignment occurs, or cluster centroids move by only minute distances.

4.1 Choosing Optimal value of K

Elbow Method - In this method, we simply plot the cost function (J) against K (no. of clusters)

$$J = \frac{1}{m} \sum_{j=1}^{k} \sum_{x_i \in c_j} dist(x_i, \mu_j)^2 \tag{5}$$

where m is the total no. of data points,
μ_j is the cluster centroid for the j^{th} cluster

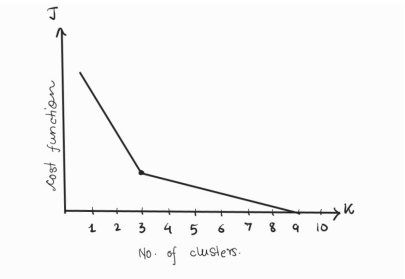


Figure 5: Illustration of Elbow Method for optimal K

4.2 Results

The results after running K means clustering algorithm on our dataset were as follows -

· Optimal value of K was found to be 2 using the Elbow method.

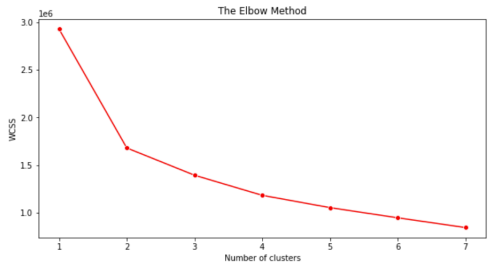


Figure 6: Elbow occurring at K = 2

· Out of the 800 images from the test dataset, 283 were not found in the same cluster as other images with similar features[3]. This indicates and reinforces the fact that there are a lot of overlapping images, in terms of their pixel intensities.

This overlap could be due to a number of reasons. Variation in light sources - Some images are taken directly in front of a light source where as in other images, light is coming from the side.
This also results in glare of images of individuals wearing spectacles. Also, some images are much brighter than others. Lastly, images of people with more prominent features such as long eyelashes or thick eyebrows tend to bring in the overlap of pixel intensities.

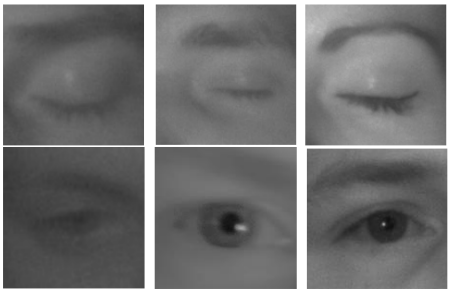


Figure 7: The above figure shows how images with different levels of brightness and light sources look like

[3]For the purpose of our project, K means algorithm has been used to verify and see the extent of overlap in our dataset. Since there are only 2 clusters being formed, we can interpret these as 0 and 1 "labels", and then check these values against the true labels to find the extent of overlap.

5 Image Processing

5.1 Cropping the images

To resolve the problem of overlap, one of the fastest solutions is to crop all the images, so that each image focuses only on the pupil region. Hence, variation due to features like eyebrows and eyelashes should be reduced.

Since all images are pre-processed to be of the same shape and size by our model, the process of cropping simply involved creating a function which gets rid of the extra portion.

Initial size of images - $224 * 224$
After cropping - $110 * 120$

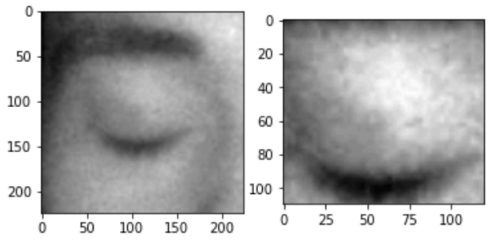


Figure 8: Example of an image before and after cropping

Results after cropping -

On applying K means clustering algorithm, results were only slightly improved, and out of 800 total images, 234 were not found in the same cluster as other images with similar features.

Hence, cropping only improved the results slightly, and there is still a need to get rid of the brightness and light variation.

5.2 Histogram Equilization

This method is used in the field of image processing to adjust the contrast of a particular image. It takes the pixel intensity of the image to create a frequency distribution (pdf) and uses it to form a linear Cummulative Distribution, which mathematically corresponds to a uniform distribution.

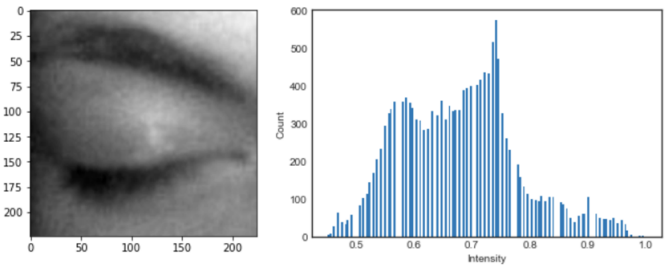


Figure 9: Representation of an image along with its normalized histogram

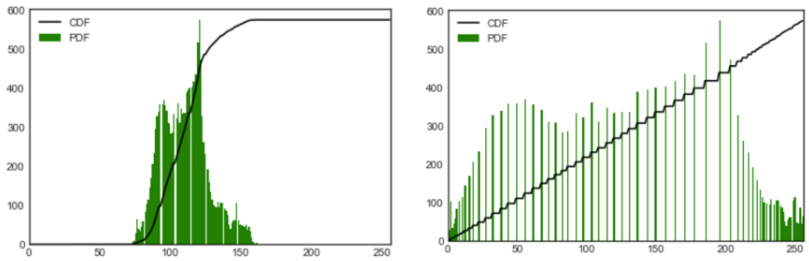


Figure 10: The above graph on the left shows CDF as well as PDF of the above image before performing histogram equilization. We see that on the scale of 0-255, pixels only take values ranging from 70-160. This shows that the contrast of the image is poor. On the right is the CDF as well as PDF of the same image after applying histogram equilization. The PDF is now spread over the entire range of possible values, and CDF is a straight linearly increasing function, indicating uniform distribution of values.

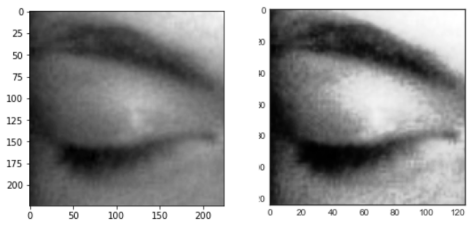


Figure 11: A comparison of the image before and after contrast adjustment

Results after Histogram Equilization -

· K means clustering algorithm was again performed after the images were cropped and adjusted for contrast.
· The results showed that out of 800 images, only 64 were not found in the same cluster as other images with similar features.This is an improvement of 21.25% from the previous runs of K means.

6 K-nearest neighbors (KNN)

K nearest neighbors is supervised machine learning classifier. As the name suggests, it considers K no. of nearest data points to predict the class of the test data point that is inputted as a query point.This is also known as lazy learning because here the model does not learn from the training data, and the learning process is postponed till each query point in taken as an input.

Following steps are used in the KNN algorithm -

· **Step 1** Load the dataset and pick a value of K.

· **Step 2** For each query point -

  · First, distance of each point from the query point is calculated using a distance metric.
  · The distances are stored in an array, in ascending order.
  · The top K distances are picked, and corresponding data points stored as the K nearest neighbours.

· **Step 3** For classification purposes, the mode of the K nearest neighbors is taken as the label for the query point.

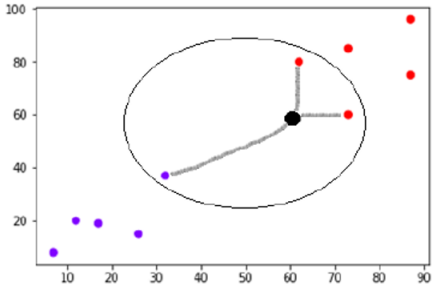


Figure 12: Source - *https : www.tutorialspoint.com/knn_algorithm.jpg*

Results after K Nearest Neighbors -

Optimal value of was found by plotting a graph of Error rate versus K

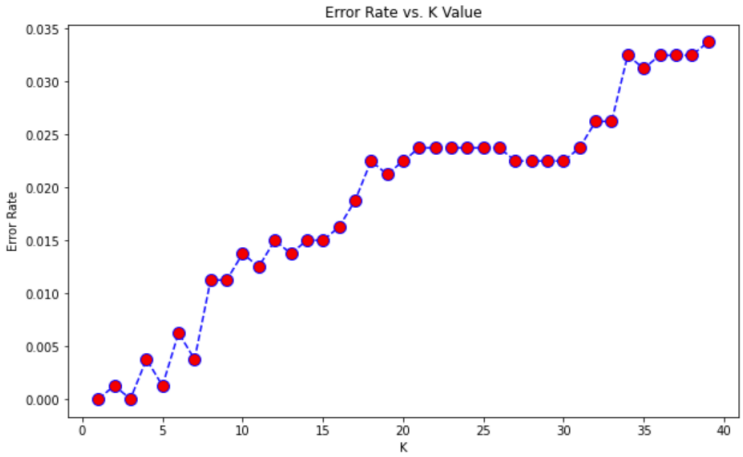


Figure 13: Minimum error occurs at K = 3

On performing KNN classification algorithm on raw data -

```
from sklearn.metrics import classification_report,confusion_matrix
print(confusion_matrix(Y_test,pred))
```

```
[[372   0]
 [  3 425]]
```

```
print(classification_report(Y_test,pred))
```

```
              precision    recall  f1-score   support

           0       0.99      1.00      1.00       372
           1       1.00      0.99      1.00       428

    accuracy                           1.00       800
   macro avg       1.00      1.00      1.00       800
weighted avg       1.00      1.00      1.00       800
```

Figure 14: Classification Report and Confusion matrix of KNN

We are getting extremely accurate results for KNN classification on our raw data. However, on testing the model on a new dataset,[4] we saw an accuracy of only 65%.
This could be because - The new randomly selected images come from a completely different distribution, in the sense that their resolution is higher, and a lot of variation coming from different lighting conditions. Since our model is built on raw input, and no cropping and contrast adjustment has taken place, this is to be expected.

[4]Downloaded 40 random images of closed and open eyes from Google to create this new test dataset.

```
from sklearn.metrics import classification_report,confusion_matrix
print(confusion_matrix(Y_t,pred_test))
```

```
[[19  1]
 [13  7]]
```

Figure 15: Classification Report and Confusion matrix of KNN

KNN after images cropping and Histogram Equilization -

```
from sklearn.metrics import classification_report,confusion_matrix
print(confusion_matrix(Y_test,pred))
```

```
[[404   0]
 [  0 396]]
```

```
print(classification_report(Y_test,pred))
```

```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00       404
           1       1.00      1.00      1.00       396

    accuracy                           1.00       800
   macro avg       1.00      1.00      1.00       800
weighted avg       1.00      1.00      1.00       800
```

Figure 16: Classification Report after

On performing KNN after cropping and histogram equilization, it predicts class of the images with 100% accuracy. However, on testing the model on a new dataset,we see an accuracy of 75%.

This is higher than the accuracy from the previous model. However, there is still room for improvement. This hints towards the fact that we need a technique to transform our feature space. Using image processing, and pixel intensity as the features is not giving very accurate predictions for a generalized set of images.

7 Edge Detection

Edge detection is a technique in image processing, used to detect sharp changes in pixel intensity, indicating an edge; the boundary of an object. These edges are the regions in the image with a high gradient. If we

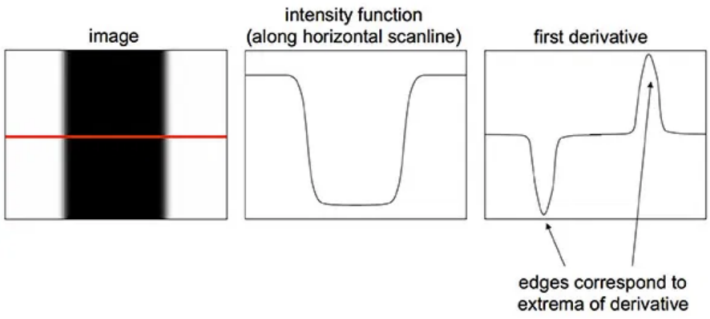


Figure 17: Source: http://stanford.edu/

calculate gradients in a region with uniform intensity, we would get the 0 vector since there is no change in pixel intensity, hence there is likely to be no edge.

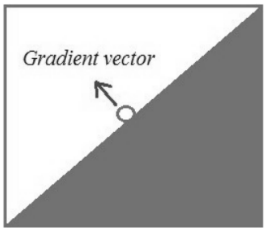


Figure 18: The gradient of an edge pixel pointing towards the rate of fastest change

There are many different types of edge detectors used in the field of image processing. We shall study and apply the following -

7.0.1 Sobel Edge Detector

The Sobel edge detector is an operator that calculates the 2D spatial gradient on images. It uses a pair of 3 X 3 kernels, one calculating the gradient in the x-direction (Gx) , and the other calculating gradient in the y-direction (Gy). These kernels individually detect horizontal and vertical edges, and then combines them together to give the resulting edge.

The resulting magnitude is given by -

$$|G| = \sqrt{(G_x)^2 + (G_y)^2} \tag{6}$$

The angle of orientation of the edge (relative to the pixel grid) which gives rise to the gradient is given by -

$$\theta = \tan^{-1}(G_y/G_x) \tag{7}$$

An angle of 0 means that the direction of maximum contrast, from black to white runs from left to right on the image, and other angles are measured anti-clockwise from this.

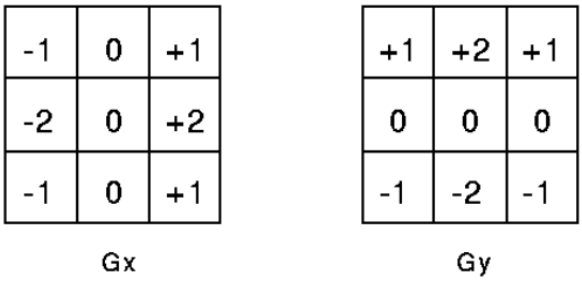
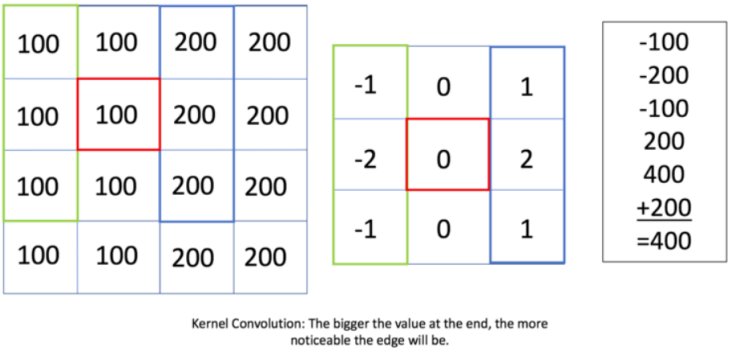
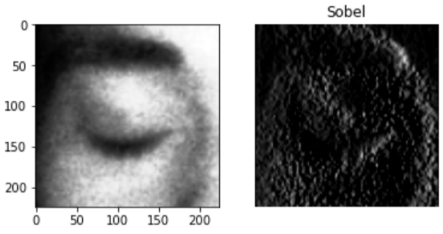


Figure 19: Gx and Gy kernels for Sobel Edge detection



Applying sobel filter on one of the images from the dataset, we get -



7.0.2 Gaussian Smoothing

The gaussian operator is used as a filter to 'smooth' the image by reducing noise and details from it. The kernel comes from a discrete approximation of the gaussian function.

In 2-D, gaussian function is of the form -

$$G(x, y) = \frac{1}{2\pi\sigma^2} * \exp^{\frac{-(x^2+y^2)}{2\sigma^2}} \tag{8}$$

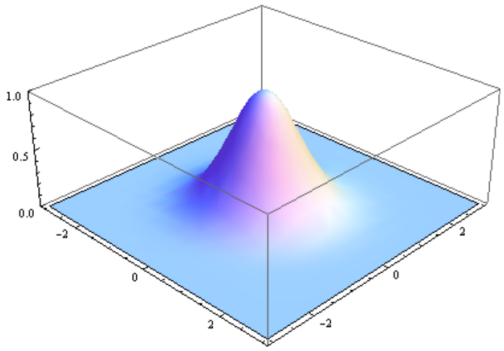


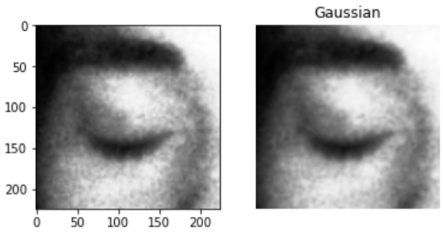
Figure 20: Source: https://www.pixelstech.net/article/images/gb-08.png

$$\frac{1}{273}\begin{bmatrix} 1 & 4 & 7 & 4 & 1 \\ 4 & 16 & 26 & 16 & 4 \\ 7 & 26 & 41 & 26 & 7 \\ 4 & 16 & 26 & 16 & 4 \\ 1 & 4 & 7 & 4 & 1 \end{bmatrix}$$

Figure 21: The discrete approximation of Gaussian function with $\sigma = 1.0$
Source: https://homepages.inf.ed.ac.uk/rbf/HIPR2/gsmooth.htm

In some sense, from the above figures, we can see that the gaussian filter takes an average of all the pixel values it is convolving over.

Applying gaussian filter on one of the images from the dataset, we get -



7.0.3 Laplace Edge Detector

The laplace edge detector calculates the 2nd derivative of an image, and looks for points which cut through the x-axis (zero-crossing). Such points are taken as edges, i.e. where the intensity is changing rapidly. However, it is susceptible to noise, and is often, in practice, used in combination with the Gaussian smoothning filer (also known as Laplace of Gaussian).

$$L(x, y) = \frac{\partial^2 I}{\partial x^2} + \frac{\partial^2 I}{\partial y^2} \tag{9}$$

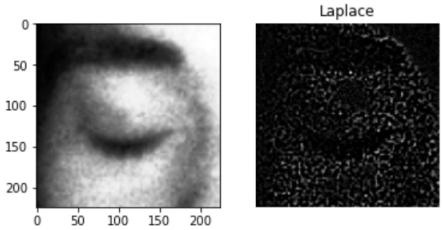
The discrete approximation of the above formula in the form of a kernel is given by -

0	−1	0
−1	4	−1
0	−1	0

−1	−1	−1
−1	8	−1
−1	−1	−1

Figure 22: The 2 most commonly used Laplacian filters

Applying Laplace of Gaussian filter on one of the images from the dataset, we get -



7.0.4 Canny Edge Detector

Canny edge detector uses various concepts from the edge detectors discussed above, and involves a multi-stage algorithm.

· **Reducing Noise** It uses a gaussian filter to blur the noise.

· **Gradient calculation** Second, it uses sobel filters to calculate gradient of the image at each pixel.

· **Non-maximum suppression** Third, it calculates the non-maximum suppression. As the name suggests, for each pixel, it checks whether or not the pixel is a local maxima in the direction of the gradient. If yes, then the value for that pixel becomes 1, otherwise 0.

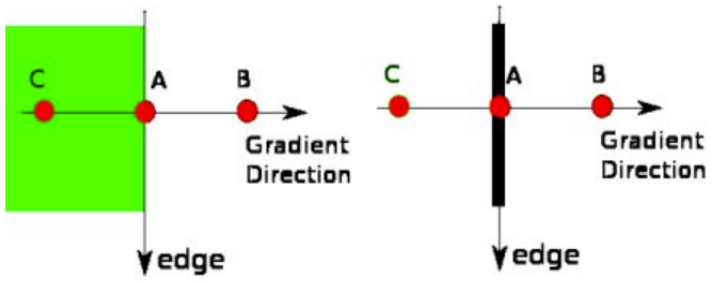


Figure 23: Non-maximum suppression in Canny edge detection

· **Max-Min Threshold** In this step, pixel values from the above steps are categorised into 3 groups; Strong, weak and non-stationary. If a certain pixel value is above max threshold, it is categorised as Strong. Between max and min threshold are weak. Below min. threshold value are non-stationary, and are discarded as "non-edges".

· **Hysteresis Threshold** The weak edge pixels are then further categorised as edges or non-edges, depending upon whether or not they are surrounded by any strong pixels or not.

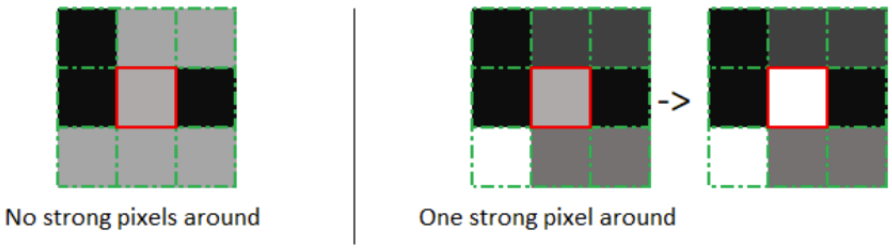
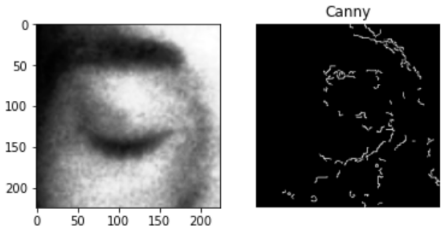


Figure 24: Hysteresis threshold in Canny edge detection.
Source: https://miro.medium.com

Applying canny filter on one of the images from the dataset, we get -



7.1 Using Edge Detection with KNN

7.1.1 Laplace of Gaussian

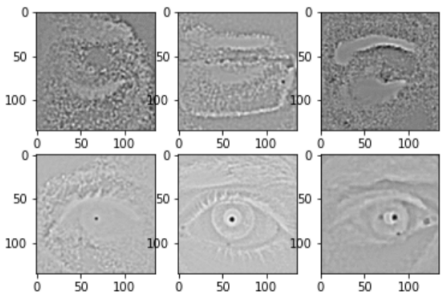


Figure 25: Images in the training set after applying LoG filter. Top: Closed Bottom: Open

```
from sklearn.metrics import classification_report,confusion_matrix
print(confusion_matrix(Y_test,pred))
```

```
[[  0 413]
 [  0 387]]
```

```
print(classification_report(Y_test,pred))
```

```
              precision    recall  f1-score   support

           0       0.00      0.00      0.00       413
           1       0.48      1.00      0.65       387

    accuracy                           0.48       800
   macro avg       0.24      0.50      0.33       800
weighted avg       0.23      0.48      0.32       800
```

Figure 26: Results after applying KNN with LoG filter.

As we can see, LoG filter gives the poorest result so far. As we can see from the above figure, LoG is able to capture the sharp edges around the pupil in open eye images, however it is not at all capturing edges in the closed eye images. This could be due to the fact that after H.E. and smoothning, which helps us in getting uniformity in all the images, it also introduces some shadows. This could also be the reason why there is so much noise in the case of closed eyes. It may also be the case that the training data is simply not enough for this filter to work properly.

The above result is produced in the case where image size = 224 during pre-processing stage. However, on doing some experimentation, it was found that reducing the image size improves accuracy of the LoG-KNN model.

```
[[360  41]
 [ 13 386]]
              precision    recall  f1-score   support

           0       0.97      0.90      0.93       401
           1       0.90      0.97      0.93       399

    accuracy                           0.93       800
   macro avg       0.93      0.93      0.93       800
weighted avg       0.93      0.93      0.93       800
```

Figure 27: Results for LoG-KNN when image size $= 86 * 86$.

The changes in result could be due to the fact that when we work with a large sized image, the proportion of smoothning due to gaussian filter is higher, and perhaps edges are not as sharply formed. When we reduce the size, the smoothning effect is also reduced, and edges are slightly sharper, hence information gain is more in this case.

7.1.2 Sobel

```
<matplotlib.image.AxesImage at 0x1abfb6fa850>
```

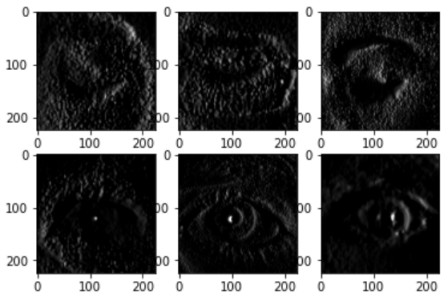


Figure 28: Images from training set after applying KNN with Sobel filter

```
from sklearn.metrics import classification_report,confusion_matrix
print(confusion_matrix(Y_test,pred))
```

```
[[279 121]
 [  5 395]]
```

```
print(classification_report(Y_test,pred))
```

```
              precision    recall  f1-score   support

           0       0.98      0.70      0.82       400
           1       0.77      0.99      0.86       400

    accuracy                           0.84       800
   macro avg       0.87      0.84      0.84       800
weighted avg       0.87      0.84      0.84       800
```

Figure 29: Results after applying KNN with Canny filter

Since Sobel filter works by calculating gradient in the x and y direction, it is able to capture edges around a closed or an open eyelid, and also the edges arising from the round shape of the pupil, which can be approximated using x and y coordinates. It is doing a much better job than LoG, and there is a clear distinction between closed and open eye images, as we can see above.

```
<matplotlib.image.AxesImage at 0x1ab88790880>
```

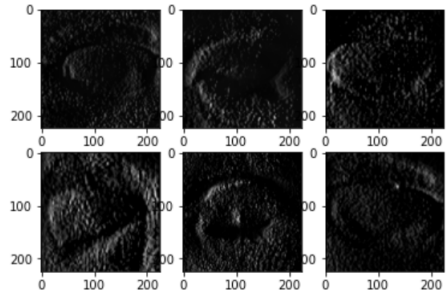


Figure 30: Example of images that were mis-classified

7.1.3 Canny

```
<matplotlib.image.AxesImage at 0x280cd1cb880>
```

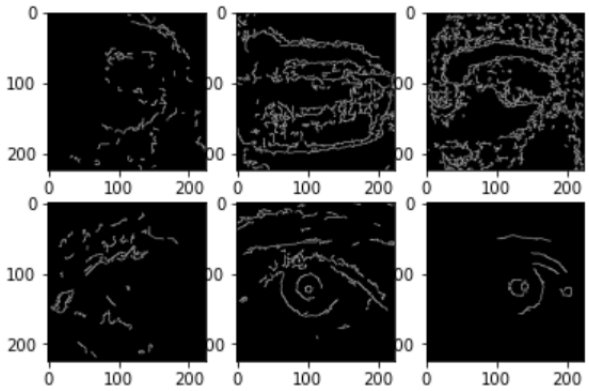


Figure 31: Images from training set after applying KNN with Canny filter

```
from sklearn.metrics import classification_report,confusion_matrix
print(confusion_matrix(Y_test,pred))
```

```
[[360  22]
 [413   5]]
```

```
print(classification_report(Y_test,pred))
```

```
              precision    recall  f1-score   support

           0       0.47      0.94      0.62       382
           1       0.19      0.01      0.02       418

    accuracy                           0.46       800
   macro avg       0.33      0.48      0.32       800
weighted avg       0.32      0.46      0.31       800
```

Figure 32: Results after applying KNN with Canny filter

As we can see, Canny is susceptible to a lot of noise, and it is detecting a lot of it in the image as edges. The top right corner image above shows perfectly how it is capturing too many edges, and there is no proper distinction between open and closed eye images. This could be happening due to the Hysteresis threshold step, which takes into account neighboring pixels to classify an edge pixel. After histogram equilization, some shadows are introduced due to contrast adjustment, and is perhaps causing the Canny edge detector to take into account a lot of extra noise as edges. Hence, it is giving such poor results.

7.1.4 Gaussian Blur

```
<matplotlib.image.AxesImage at 0x252a9783dc0>
```

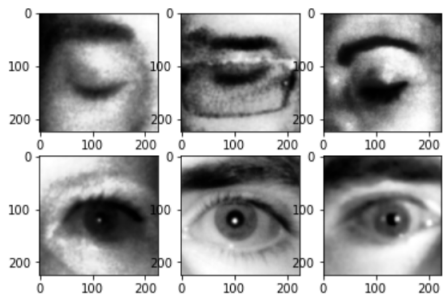


Figure 33: Images from training set after applying KNN with Gaussian filter

```
from sklearn.metrics import classification_report,confusion_matrix
print(confusion_matrix(Y_test,pred))
```

```
[[389   0]
 [  0 411]]
```

```
print(classification_report(Y_test,pred))
```

```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00       389
           1       1.00      1.00      1.00       411

    accuracy                           1.00       800
   macro avg       1.00      1.00      1.00       800
weighted avg       1.00      1.00      1.00       800
```

Figure 34: Results after applying KNN with Gaussian filter

As we can see, Gaussian blur is giving the best result. This could be because, this filter simply reduces noise and sharpens the edges in the image. It preserves the rest of the features, and this works best for our purpose, since the pixel intensities are also important for classification.

When tested on a new completely new dataset, it is also giving good results. However, since these new images may be coming from a completely different distribution, the accuracy is reduced. However, this could be improved upon by simply increasing the size of the training dataset.

```
from sklearn.metrics import classification_report,confusion_matrix
print(confusion_matrix(Y_t,pred_test))
```

```
[[13  7]
 [ 6 14]]
```

```
print(classification_report(Y_t,pred_test))
```

```
              precision    recall  f1-score   support

           0       0.68      0.65      0.67        20
           1       0.67      0.70      0.68        20

    accuracy                           0.68        40
   macro avg       0.68      0.68      0.67        40
weighted avg       0.68      0.68      0.67        40
```

Figure 35: Results on test set after applying KNN with Gaussian filter

7.2 Conclusion

We can see from the above results that for the purpose of our project, edge detection alone does not fulfill our requirements. Comparing the results of Gaussian filter with the rest, we see that it is essential to preserve information in the form of pixel intensity as well, as simply using edge detection leads to significant information loss.

8 Future Work

In the case of KNN with edge detection, there is room for improvement if we use parameter tuning; using K-fold cross validation process to get optimal value of K. This will be the immediate next step for our project.

After doing image processing, applying Machine learning algorithms, and edge detection, the next step in our project will be to apply simple Artificial Neural Networks; to figure out optimal values of input parameters, as well as hidden layers.

9 References

- "A Comparison of various Edge Detection Techniques used in Image Processing" by G.T. Shrivakshan, Bharathiar University, Coimbatore and Dr.C. Chandrasekar, Associate Professor, Periyar University Salem, Tamilnadu.
- "Edge Detection Operators: Peak Signal to Noise Ratio Based Comparison" D. Poobathy Research Scholar, Dr. Mahalingam Centre for Research and Development, NGM College, Pollachi, and Dr. R. Manicka Chezian, Associate Professor, Dr. Mahalingam Centre for Research and Development, NGM College, Pollachi.
- "A Descriptive Algorithm for Sobel Image Edge Detection" by O. R. Vincent, Clausthal University of Technology, Germany and University of Agriculture, Abeokuta, Nigeria and O. Folorunso, Department of Computer Science, University of Agriculture, Abeokuta, Nigeria.
- A course in Machine Learning by Hal Daume
- The Data Science Handbook by Field Cady
- Digital Image Processing 3rd ed. - R. Gonzalez, R. Woods
- Feature Detectors - Zero Crossing Detector
- Spatial Filters - Laplacian/Laplacian of Gaussian
- In Depth: k-Means Clustering — Python Data Science Handbook
- MRL Eye Dataset — MRL
- TOP Machine Learning Algorithms -Beginner | Kaggle
- Comprehensive data exploration with Python | Kaggle
- Eye blink detection for different driver states in conditionally automated driving and manual driving using EOG and a driver camera — SpringerLink
- The Sobel and Laplacian Edge Detectors - AI Shack