CHAPTER 1

INTRODUCTION

1.1 MACHINE LEARNING

Machine Learning (ML) can be explained as automating and improving the learning process of computers based on their experiences without being actually programmed i.e. without any human assistance. The process starts with feeding a good quality data and then training our machines (computers) by building machine learning models using the data and different algorithms. The choice of algorithms depends on what type of data do we have and what kind of task we are trying to automate.

1.1.1 METHODS OF MACHINE LEARNING

Supervised machine learning algorithms: can apply what has been learned in the past to new data using labeled examples to predict future events. Starting from the analysis of a known training dataset, the learning algorithm produces an inferred function to make predictions about the output values. The system is able to provide targets for any new input after sufficient training. The learning algorithm can also compare its output with the correct, intended output and find errors in order to modify the model accordingly.

Unsupervised machine learning algorithms: are used when the information used to train is neither classified nor labeled. Unsupervised learning studies how systems can infer a function to describe a hidden structure from unlabeled data. The system doesn't figure out the right output, but it explores the data and can draw inferences from datasets to describe hidden structures from unlabeled data.

Semi-supervised machine learning algorithms from it. Otherwise, acquiring unlabeled data generally doesn't require additional fall somewhere in between supervised and unsupervised learning, since they use both labeled and unlabeled data for training – typically a small amount of labeled data and a large amount of unlabeled data. The systems that use this method are able to considerably improve learning accuracy. Usually, semi-supervised learning is chosen when the acquired labeled data requires skilled and relevant resources in order to train it / learn resources.

Reinforcement machine learning algorithms: is a learning method that interacts with its environment by producing actions and discovers errors or rewards. Trial and error search and delayed reward are the most relevant characteristics of reinforcement learning. This method allows machines and software agents to automatically determine the ideal behavior within a specific context in order to maximize its performance. Simple reward feedback is required for the agent to learn which action is best; this is known as the reinforcement signal.

1.2 IMAGE PROCESSING

Image processing is a method to convert an image into digital form and perform some operations on it, in order to get an enhanced image or to extract some useful information from it. It is a type of signal dispensation in which input is image, like video frame or photograph and output may be image or characteristics associated with that image. Usually Image Processing system includes treating images as two dimensional signals while applying already set signal processing methods to them.

1.2.1 PURPOSE OF IMAGE PROCESSING

The purpose of image processing is divided into 5 groups. They are:

1. Visualization - Observe the objects that are not visible.
2. Image sharpening and restoration - To create a better image.
3. Image retrieval - Seek for the image of interest.
4. Measurement of pattern – Measures various objects in an image.
5. Image Recognition – Distinguish the objects in an image.

1.3 CLASSIFICATION

Classification is the problem of identifying to which of a set of categories (sub-populations) a new observation belongs, on the basis of a training set of data containing observations (or instances) whose category membership is known. Examples are assigning a given email to the "spam" or "non-spam" class, and assigning a diagnosis to a given patient based on observed characteristics of the patient (sex, blood pressure, presence or absence of certain symptoms, etc.). Classification is an example of pattern recognition.

1.3.1 ANN- ARTIFICIAL NEURAL NETWORK

Artificial neural networks are one of the main tools used in machine Learning. As the “neural” part of their name suggests, they are brain-inspired systems which are intended to replicate the way that we humans learn. Neural networks consist of input and output layers, as well as (in most cases) a hidden layer consisting of units that transform the input into something that the output layer can use. They are excellent tools for finding patterns which are far too complex or numerous for a human programmer to extract and teach the machine to recognize.

1.3.1.1 STRUCTURE OF ANN

ANNs are composed of multiple nodes, which imitate biological neurons of human brain. The neurons are connected by links and they

interact with each other. The nodes can take input data and perform simple operations on the data. The result of these operations is passed to other neurons. The output at each node is called its activation or node value. Each link is associated with weight. ANNs are capable of learning, which takes place by altering weight values.

TYPES OF ANN

Feed Forward ANN

In this ANN, the information flow is unidirectional. A unit sends information to other unit from which it does not receive any information. There are no feedback loops. They are used in pattern generation/recognition/classification. They have fixed inputs and outputs.

Feed Back ANN

Here, feedback loops are allowed. They are used in content addressable memories.

1.3.2 NAIVE BAYES CLASSIFIER

Naive Bayes classifiers are a collection of classification algorithms based on Bayes' Theorem. It is not a single algorithm but a family of algorithms where all of them share a common principle, i.e. every pair of features being classified is independent of each other.

BAYES' THEOREM

$$P(A|B) = P(B|A)\, P(A) / P(B)$$

To find probability of event A, given the event B is true. Event B is also termed as evidence. P(A) is the priori of A (the prior probability, i.e. Probability of event before evidence is seen). The evidence is an attribute value of an unknown instance

(here, it is event B). P(A|B) is a posteriori probability of B, i.e. probability of event after evidence is seen.

TYPES OF NAIVE BAYES CLASSIFIER

Multinomial Naive Bayes

This is mostly used for document classification problem, i.e whether a document belongs to the category of sports, politics, technology etc. The features/predictors used by the classifier are the frequency of the words present in the document.

Bernoullin Naive Bayes

This is similar to the multinomial naive bayes but the predictors are boolean variables. The parameters that we use to predict the class variable take up only values yes or no, for example if a word occurs in the text or not.

Gaussian Naïve Bayes

When the predictors take up a continuous value and are not discrete, we assume that these values are sampled from a gaussian distribution.

1.3.3 SVM-SUPPORT VECTOR MACHINE

The objective of the support vector machine algorithm is to find a hyperplane in an N-dimensional space(N-the number of features) that distinctly classifies the data points. To separate the two classes of data points, there are many possible hyperplanes that could be chosen. Our objective is to find a plane that has the maximum margin, i.e., the maximum distance between data points of both classes. Maximizing the margin distance provides some reinforcement so that future data points can be classified with more confidence.

Support vectors are data points that are closer to the hyperplane and influence the position and orientation of the hyperplane. Using these support vectors, we

maximize the margin of the classifier. Deleting the support vectors will change the position of the hyperplane.

1.4 EDGE DETECTION

Edge Detection is simply a case of trying to find the regions in an image where we have a sharp change in intensity or a sharp change in color, a high value indicates a steep change and a low value indicates a shallow change.

1.4.1 SOBEL OPERATOR

Sobel operator is a derivate mask and is used for edge detection.

Sobel operator is also used to detect two kinds of edges in an image.

- Vertical direction
- Horizontal direction

1.4.2 CANNY OPERATOR

The canny edge detector is a multistage edge detection algorithm. The steps are:

1. Preprocessing
2. Calculating gradients
3. Nonmaximum suppression
4. Thresholding with hysterysis

The two key parameters of the algorithm are an upper threshold and a lower threshold. The upper threshold is used to mark edges that are definitely edges. The lower threshold is to find faint pixels that are actually a part of an edge.

Preprocessing

Edge detectors are prone to noise. A bit of smoothing with a Gaussian blur helps. 5x5 Gaussian filter with standard deviation = 1.4

Calculating gradients

Gradient magnitudes and directions are calculated at every single point in the image. The magnitude of the gradient at a point determines if it possibly lies on an edge or not. A high gradient magnitude means the colors are changing rapidly - implying an edge. A low gradient implies no substantial changes. So it's not an edge. The magnitude of gradient is $m = \sqrt{G_x^2 + G_y^2}$ The direction of gradient $\theta = \arctan(G_{x,}/G_y)$ .Here, G_x and , G_y are the X and Y derivatives at the point being considered.

Nonmaximum supression

For each image pixel in the direction matrix, if corresponding pixel of the magnitude image is less than its diagonals, vertical or horizontal pixels we make that pixel 0.

Thresholding With Hysterysis

Hyterysis is the lagging of an effect is a kind of inertia. In the context of thresholding, it means that areas above some low threshold are considered to be above the threshold if they are also connected to areas above a higher, more stringent, threshold. They can thus be seen as continuations of these high-confidence areas.

1.4.3 PREWITT OPERATOR

Prewitt operator is used for edge detection in an image. It detects two types of edges Horizontal edges, Vertical Edges. Edges are calculated by using

difference between corresponding pixel intensities of an image. All the masks that are used for edge detection are also known as derivative masks. Because as we have stated many times before in this series of tutorials that image is also a signal so changes in a signal can only be calculated using differentiation. So that's why these operators are also called as derivative operators or derivative masks.

- Opposite sign should be present in the mask.
- Sum of mask should be equal to zero.
- More weight means more edge detection.

1.5 GAUSSIAN FILTER

A Gaussian filter is a linear filter. It's usually used to blur the image or to reduce noise. If you use two of them and subtract, you can use them for "unsharp masking" (edge detection). The Gaussian filter alone will blur edges and reduce contrast. One advantage a Gaussian filter has over a median filter is that it's faster because multiplying and adding is probably faster than sorting. It defines a probability distribution for noise or data. It is a smoothing operator. It is used in mathematics.

$$G(x,y) = \frac{1}{2\pi\sigma^2} e^{-\frac{x^2+y^2}{2\sigma^2}}$$

Where σ is the standard deviation of the distribution. The distribution is assumed to have a mean of 0.

MEDIAN FILTER

The Median Filter is a non-linear digital filtering technique, often used to remove noise from an image or signal. Such noise reduction is a typical pre-processing step to improve the results of later processing (for example, edge

detection on an image). Median filtering is very widely used in digital image processing because, under certain conditions, it preserves edges while removing noise (but see discussion below), also having applications in signal processing. Median filtering is a nonlinear process useful in reducing impulsive, or salt-and-pepper noise. It is also useful in preserving edges in an image while reducing random noise. Impulsive or salt-and pepper noise can occur due to a random bit error in a communication channel. In a median filter, a window slides along the image, and the median intensity value of the pixels within the window becomes the output intensity of the pixel being processed. For example, suppose the pixel values within a window are 5, 6, 55, 10 and 15 and the pixel being processed has a value of 55. The output of the median filter the current pixel location is 10, which is the median of the five values.

CHAPTER 2
LITERATURE SURVEY

Iris recognition is a biometrical based technology for personal identification and verification. Iris recognizes a person from his/her iris prints and analyses the features that have the colored tissue surrounding the pupil. Old identification system uses password, token cards or pins. However, these security methods are not efficient enough due to forgotten, stolen and borrowed. Therefore, in this paper 1, the authors proposed artificial neural network for iris recognition because of iris uniqueness and stability. MATLAB was used as the development tool for iris recognition. The performance of their work in iris recognition was found to be more than 80%.

Iris recognition is a proven, accurate means to identify people. In paper 2, the authors included the preprocessing, segmentation, feature extraction and recognition techniques. Especially it focused on image segmentation and statistical feature extraction for iris recognition process. The performance of iris recognition system highly depends on segmentation. For instance, even an effective feature extraction method would not be able to obtain useful information from an iris image that is not segmented properly. The authors used the sobel detector to determine an automated global threshold and the pupil center.

Biometric technologies such as fingerprint, facial recognition, and iris recognition are deployed for verification and/or identification in applications such as access control, border management, and Identification systems. In paper 3, system performed with perfect recognition on a set of 40 eye images. The recognition scheme presented by the authors was found to be more reliable and accurate biometric technology by using Hamming Distance method.

The authors suggested Sobel filter for edge detection and then ROIs were segmented. Then segmented images were denoised by applying Mean filter. The pixel values changed between 0 to 255 in gray scale for inputs of neural network. And these values were to 0 to 1 and so the training and test data sets were formed. After these processes, for classification, multilayer perceptron neural network (MLPNN) employing back normalized propagation algorithm was used. In paper 4, the correct classification rate was found to be 95%.

In paper 5, the authors suggested five Artificial Neural Network (ANN) models separately for iris recognition system: feed forward (FFBPNN), cascade forward (CFBPNN), function fitting (FitNet), pattern recognition (PatternNet) and learning vector quantization (LVQNet). For each ANN model, two architectures were constructed separately; 4 layers and 7 layers, each with different numbers of hidden layer units (5, 10 and 15). The authors used ten different ANN optimization training algorithms (LM, BFG, BR, CGF, GD, GDM, GDA, GDX, OSS and RP) to train each model separately. Many experiments were conducted for each one of the five models. Each model used two different architectures, a different number of hidden layer neurons and ten different training algorithms. The performance results of the

models were compared according to mean square error to identify the best ANN model. The results showed that the PatternNet model was the best model used. Finally, comparisons between the ten training algorithms were performed through training the PatternNet model. Comparison results showed that TrainLM was the best training algorithm for the iris recognition system

In paper 6, the authors used Daugman's model for Iris recognition. The Daugman's work divided into four main parts: Such as segmentation, normalization, feature extraction and matching. The authors used 2D Gabor phase coefficients and wavelets to decompose the images. The authors used the combination of DCT and wavelet transform for obtaining more efficiency in iris recognition system. So the cost of designing and manufacturing system was very high.

CHAPTER 3

SYSTEM ANALYSIS

3.1 EXISTING SYSTEM

- The Existing System of Iris Recognition for Password Authentication Techniques were implemented in MATLAB Programming Language.
- It could not match, if the given eye image was blurred and of poor quality.
- It used traditional method of Decision making techniques named Hamming Distance, SVM methods.
- The result of the HD, SVM techniques were gave the poor accuracy rate.
- It removes less noise in the image by using mean blur filtering.
- It found all contour values by applying adaptive threshold to remove unimportant region in the image.
- It identified only boundary box of the iris image.
- The major disadvantages of Existing system was high cost.

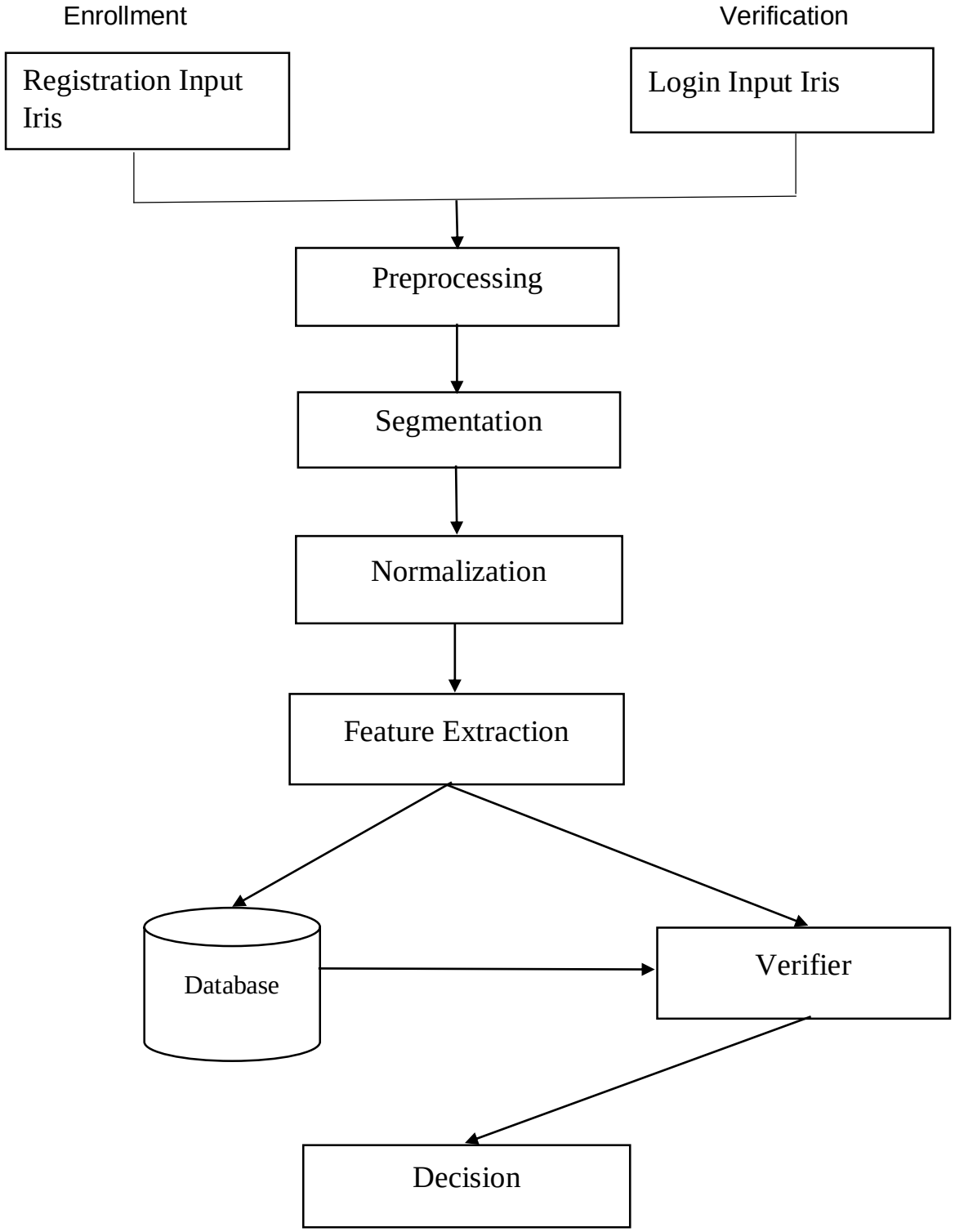
3.2 PROPOSED SYSTEM

Iris recognition for password authentication is one of the computer vision application. The objective is to design an accurate identification by using human's Iris. This system first captures the human eye image. The modules used in this systems are Preprocessing, Segmentation, Normalization, Feature extraction and Decision making. In preprocessing, the noise (or) unnecessary features like eyelids, eyeshades are removed. Segmentation is used to separate the region of the iris from the other parts of the eye. The canny edge detector was used to detect the region of the iris. Iris normalization is mainly used to resize the iris image. The use of normalization module is to get accurate iris detection. Then finally the threshold value is generated in the feature extraction module. The generated threshold values are stored in database. In testing phase, when the user logs into the system, some threshold value is generated currently for that user. Then the system compares that generated threshold value with the already stored threshold value in the database. If it matches, the user is identified as the authorized user and the system welcomes him. Otherwise, it displays error.

CHAPTER 4

SYSTEM DESIGN

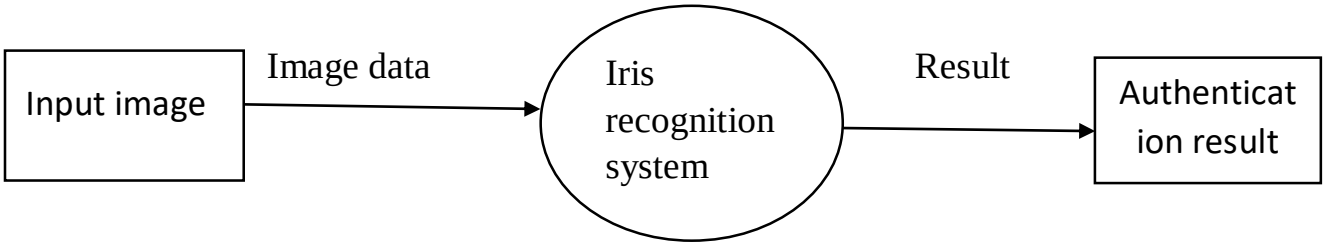
4.1 SYSTEM ARCHITECTURE



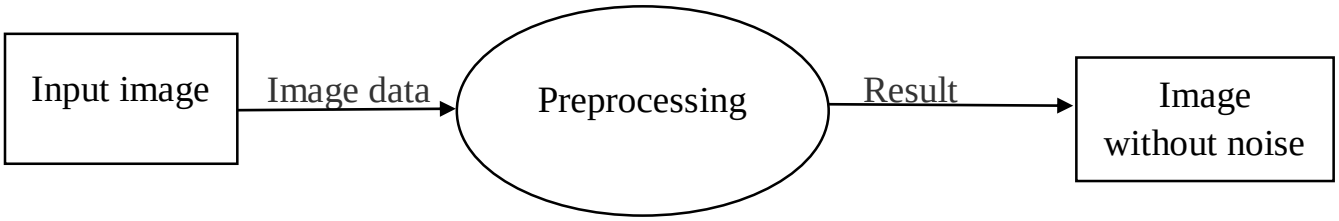
4.2 DATAFLOW DIAGRAM

A data-flow diagram (DFD) is a way of representing a flow of a data of a process or a system (usually an information system) The DFD also provides information about the outputs and inputs of each entity and the process itself. A data-flow diagram has no control flow, there are no decision rules and no loops. Specific operations based on the data can be represented by a flowchart.

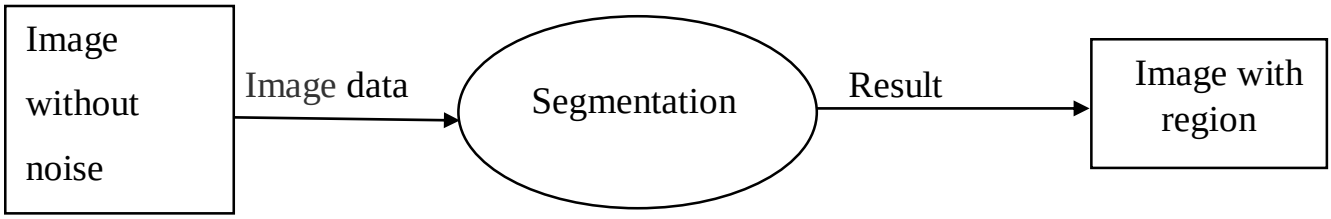
LEVEL 0:



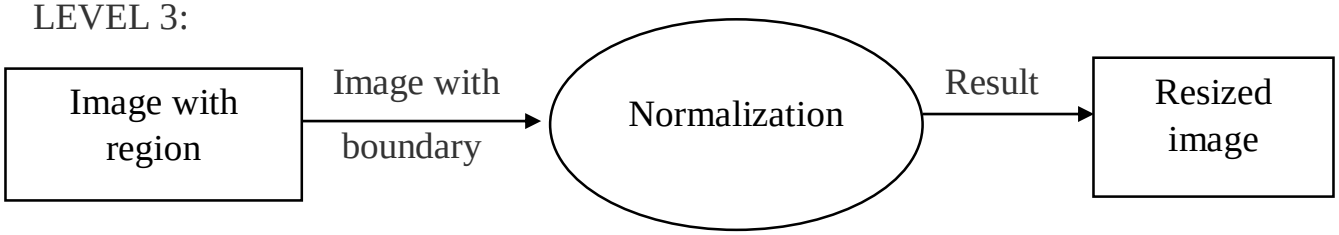
LEVEL 1:



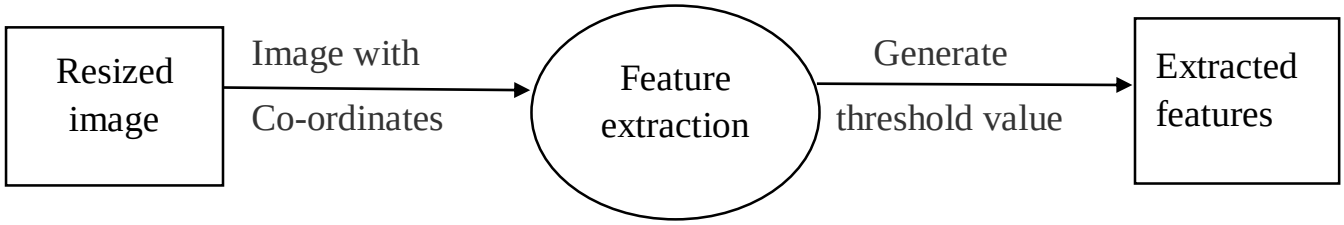
LEVEL 2:



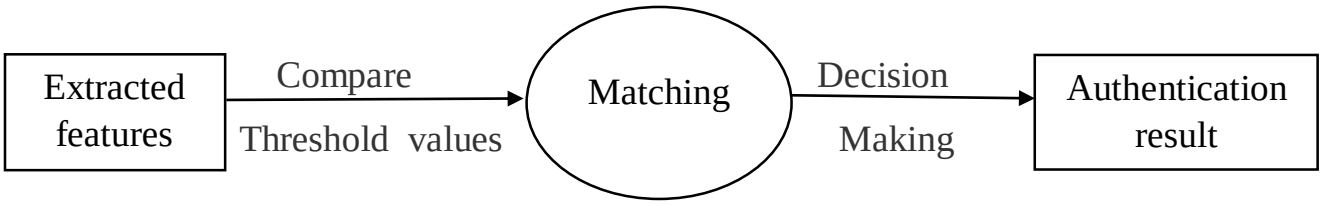
LEVEL 3:



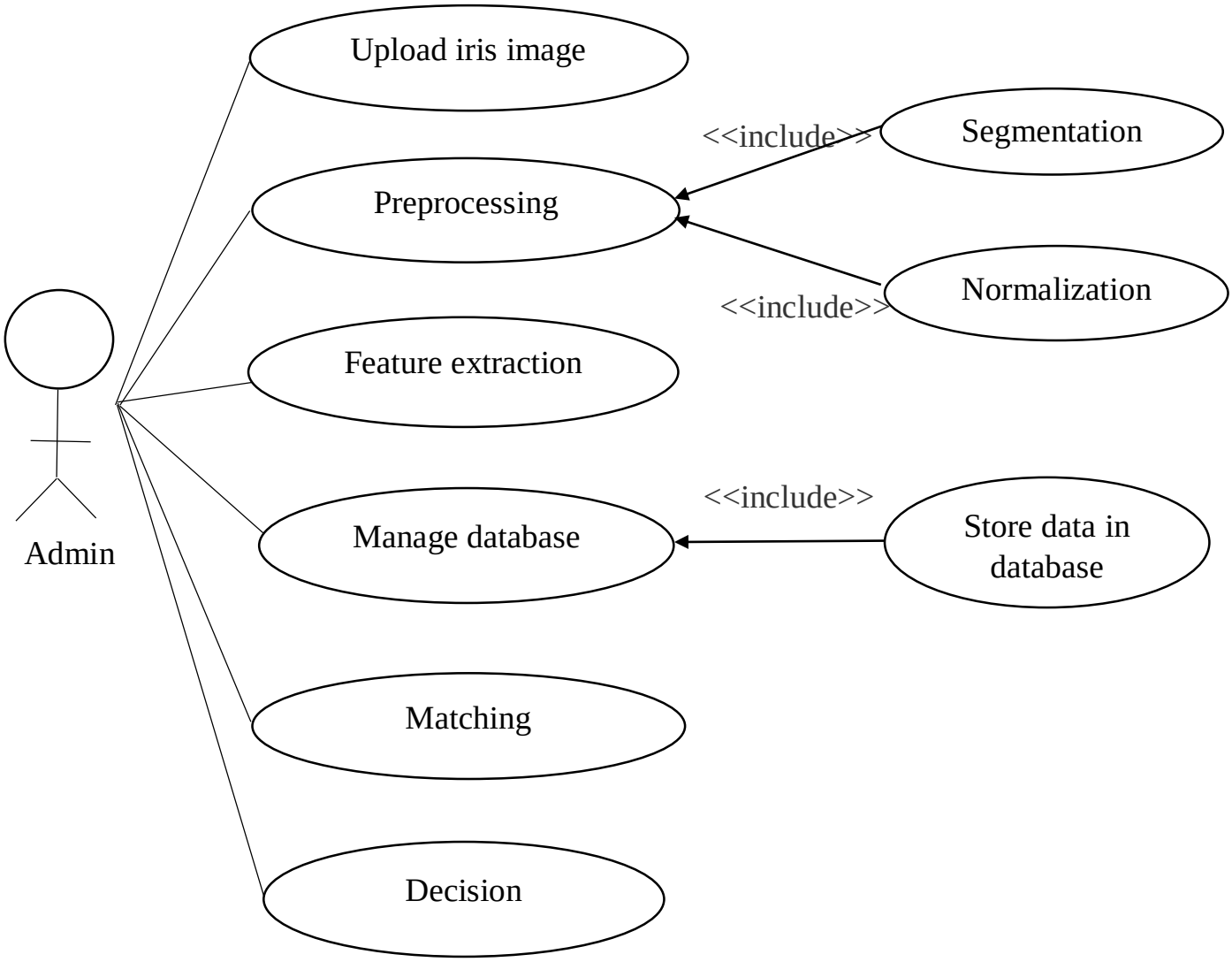
LEVEL 4:



LEVEL 5:



4.3 USECASE DIAGRAM



4.4MODULE DESCRIPTION

4.4.1 PREPROCESSING

Smooth: Spatial smoothing for images.

Background Subtraction(Flatfield): Rolling-ball background subtraction for images.

Close (Dilate+Erode): Perform dilation followed by erosion on a binary image.

Dilate: Perform dilation on a binary image.

Erode: Perform erosion on a binary image.

Max: Max value over neighboring pixels.

Median: Median value over neighboring pixels.

Mean: Mean value over neighboring pixels.

Min: Min value over neighboring pixels.

Open (Erode + Dilate): Perform erosion followed by dilation on a binary image.

Trimmed Median: Trimmed mean value over neighboring pixels.

METHOD DESCRIPTION

Dilate

Dilation of a binary (0/1, 'off'/'on') image expands contiguous regions of 'on' pixels in an image. It examines the neighboring pixels within a 'Window' around each pixel. If more than the specified fraction ('Threshold') of neighboring pixels are 'on' then dilation turns the pixel 'on'. Pixels which are 'on' already are never altered.

Parameters:

Window:an odd positive integer indicating the size of square window to use. Default = 5. Larger values increase dilation

Threshold:fraction of an 'off' pixel's neighbors (within window) which must be 'on' to turn that pixel 'on'. Default = 0.1. Increased values inhibit dilation.

Erode

Erosion of a binary (0/1, 'off'/'on') image contracts contiguous regions of 'on' pixels in an image. It examines the neighboring pixels within a specified 'Window'. If more than the specified fraction ('Threshold') of neighboring pixels are 'off' then erosion turns the pixel 'off'. Pixels which are 'off' already are never altered.

Parameters:

Window:an odd positive integer indicating the size of square window to use. Default = 5. Larger values increase erosion.

Threshold:fraction of an 'on' pixel's neighbors (within window) which must be 'off' to turn that pixel 'off'. Default = 0.1. Increased values inhibit erosion.

Close (Dilate+Erode)

Perform dilation followed by erosion. This has the effect of smoothing shapes (clusters of 'on' pixels) in the binary image and filling small holes within shapes.

Parameters:

Window: an odd positive integer indicating the size of square window to use. Default = 5.

Threshold:fraction of pixel's neighbors (within window) used in the Dilate and Erode operations. Default = 0.1.

Open (Erode+Dilate)

Perform erosion followed by dilation. This has the effect of smoothing shapes (clusters of 'on' pixels) in the binary image and removing small shapes or isolated pixels.

Parameters:

Window:an odd positive integer indicating the size of square window to use. Default = 5.

Threshold:fraction of pixel's neighbors (within window) used in the Dilate and Erode operations. Default = 0.1

4.4.2 SEGMENTATION

Segmentation partitions an image into distinct regions containing each pixels with similar attributes. To be meaningful and useful for image analysis and interpretation, the regions should strongly relate to depicted objects or features of interest. Meaningful segmentation is the first step from low-level image processing transforming a greyscale or color image into one or more other images to high-level image description in terms of features, objects, and scenes. The success of image analysis depends on reliability of segmentation, but an accurate partitioning of an image is generally a very challenging problem.

Segmentation techniques are either contextual or non-contextual. The latter take no account of spatial relationships between features in an image and group pixels together on the basis of some global attribute, e.g. grey level or colour. Contextual techniques additionally exploit these relationships, e.g. group together pixels with similar grey levels and close spatial locations.

NON CONTEXTUAL THRESHOLDING

SIMPLE THRESHOLDING

The most common image property to threshold is pixel grey level: $g(x,y) = 0$ if $f(x,y) < T$ and $g(x,y) = 1$ if $f(x,y) \geq T$, where T is the threshold. Using two

thresholds, $T_1 < T_1$, a range of grey levels related to region 1 can be defined: $g(x,y) = 0$ if $f(x,y) < T_1$ OR $f(x,y) > T_2$ and $g(x,y) = 1$ if $T_1 \leq f(x,y) \leq T_2$.

The main problems are whether it is possible and, if yes, how to choose an adequate threshold or a number of thresholds to separate one or more desired objects from their background. In many practical cases the simple thresholding is unable to segment objects of interest, as shown in the above images.

A general approach to thresholding is based on assumption that images are multimodel, that is, different objects of interest relate to distinct peaks (or modes) of the 1D signal histogram. The thresholds have to optimally separate these peaks in spite of typical overlaps between the signal ranges corresponding to individual peaks. A threshold in the valley between two overlapping peaks separates their main bodies but inevitably detects or rejects falsely some pixels with intermediate signals. The optimal threshold that minimises the expected numbers of false detections and rejections may not coincide with the lowest point in the valley between two overlapping peaks.

CONTEXTUAL SEGMENTATION

Pixel connectivity is defined in terms of pixel neighbourhoods. A normal rectangular sampling pattern producing a finite arithmetic lattice {(x,y): x = 0, 1, ..., X−1; y = 0, 1, ..., Y−1} supporting digital images allows us to define two types of neighbourhood surrounding a pixel. A 4-neighbourhood {(x−1,y), (x,y+1), (x+1,y), (x,y−1)} contains only the pixels above, below, to the left and to the right of the central pixel (x,y). An 8-neighbourhood adds to the 4-neighbourhood four diagonal neighbours: {(x−1,y−1),(x−1,y), (x−1,y+1), (x,y+1), (x+1,y+1), (x+1,y), (x+1,y−1), (x,y−1)}.

A 4-connected path from a pixel p_1 to another pixel p_n is defined as the sequence of pixels $\{p_1, p_2, ..., p_n\}$ such that p_{i+1} is a 4-neighbour of p_i for all $i = 1, ..., n-1$. The path is 8-connected if p_{i+1} is an 8-neighbour of p_i. A set of pixels is a 4-connected region if there exists at least one 4-connected path between any pair of pixels from that set. The 8-connected region has at least one 8-connected path between any pair of pixels from that set.One of the simplest and most common algorithms for labelling connected regions after greyscale or colour thresholding exploits the "grassfire" or "wave propagation" principle: after a "fire" or "wave" starts at one pixel, it propagates to any of the pixel's 4- or 8-neighbours detected by thresholding. Each already visited (i.e. "burnt away" or "wet") pixel cannot be visited again, and after the entire connected region is labelled, its pixels are assigned a region number, and the procedure continues to search for the next connected region. Magenta and yellow stars below indicate the fire, or wave front pand the burnt away pixels, respectively.

4.4.3 NORMALIZATION

The purpose of image normalization is often either (1) to make features of the image more familiar to human senses—perhaps more appealing; or (2) in Machine Vision applications to increase contrast for improved feature extraction or image segmentation. Linear Normalization will “stretch” the intensity range of an image and follows this formula:

$$I_N = (I - Min) * ((Max_N - Min_N) / (Max - Min)) + Min_N ,$$

where IN is the normalized pixel, I is the intensity of the original pixel, Min and Max are the minimum and maximum pixel intensity in the image to be “stretched” to the new intensity, and MinN and MaxN are the new minimum and maximum pixel range for the normalized image. In the Cognex Insight platform, this is similar to the

image filter called “stretch filter”. The lower and upper histogram intensities are stretched to be lower and higher in intensity to increase contrast. It is also possible to perform a non-linear normalization. In this case, the image Gamma, Brightness, and Contrast can all be adjusted producing a nonlinear transformation.

4.4.4 FEATURE EXTRACTION

Transformation of input data into a set of features. Features are distinctive properties of input patterns that help in differentiating between the Categories of input patterns. Quantification of image content by means of computer algorithms, aiming to capture tissue alterations, due to underlying biological processes reflected either as morphology or as texture variations, mimicking or complementing radiologist interpretation, used in subsequent pattern analysis. The process to represent raw image in a reduced form to facilitate decision making such as pattern detection, classification or recognition. Feature extraction transforms an input image into a set of features. In machine learning, feature extraction starts from an initial set of measured data and builds derived values (features) intended to be informative and non-redundant, facilitating the subsequent learning and generalization steps, and in some cases leading to better human interpretations. Feature extraction is a dimensionality reduction process, where an initial set of raw variables is reduced to more manageable groups (features) for processing, while still accurately and completely describing the original data set. Feature extraction is the process of transforming the input data into a set of features which can very well represent the input data. It is a special form of dimensionality reduction.

RESULT AND DISCUSSION

In this project, ANN is main method used to take Decision for Authentication process. By using ANN, this system compares the generated threshold value with the stored threshold value. If the threshold values match, the system welcomes the user. Else it displays error. This system concentrates on the accuracy of the image in authenticating the user.

CHAPTER 5

SYSTEM ENVIRONMENT

5.1 HARDWARE ENVIRONMENT

The hardware requirements may serve as the basics for a contract for the implementation of the system and should therefore be a complete and consistent specification of the whole system. They are used by software engineers as the starting point for the system design.

PROCESSOR	:Core i5
MONITOR	:15” COLOR
HARD DISK	:1TB
KEYBOARD	:STANDARD 102 KEYS
MOUSE	:3 BUTTONS
OUTPUT DEVICE	:LED MONITOR
RAM	:4GB

5.2 SOFTWARE ENVIRONMENT

The software requirements document is the specification of the system. It should include both a definition and a specification of requirements.

OPERATING SYSTEM	:Windows 10
FRONTEND	:Python

CHAPTER 6

SOFTWARE DESCRIPTION

6.1 PYTHON

Python is an object-oriented, high-level programming language with integrated dynamic semantics primarily for web and app development. It is extremely attractive in the field of Rapid Application Development because it offers dynamic typing and dynamic binding options. Python is relatively simple, so it's easy to learn since it requires a unique syntax that focuses on readability. Developers can read and translate Python code much easier than other languages. In turn, this reduces the cost of program maintenance and development because it allows teams to work collaboratively without significant language and experience barriers.

Additionally, Python supports the use of modules and packages, which means that programs can be designed in a modular style and code can be reused across a variety of projects. Once you've developed a module or package you need, it can be scaled for use in other projects, and it's easy to import or export these modules.

One of the most promising benefits of Python is that both the standard library and the interpreter are available free of charge, in both binary and source form. There is no exclusivity either, as Python and all the necessary tools are available on all major platforms. Therefore, it is an enticing option for developers who don't want to worry about paying high development costs.

1. It is easy to learn- the time needed to learn python shorter than for many other languages; this means that is possible to start the actual programming faster.
2. It is easy to teach- the teaching workload is smaller than that needed by other languages; this means that the teacher can put more emphasis on

general (language-independent) programming techniques, not wasting energy on exotic tricks, strange exceptions and incomprehensible rules;

OPENCV

OpenCV-Python is a library of Python bindings designed to solve computer vision problems.Python is a general purpose programming language started by Guido van Rossum that became very popular very quickly, mainly because of its simplicity and code readability. It enables the programmer to express ideas in fewer lines of code without reducing readability.

Compared to languages like C/C++, Python is slower. That said, Python can be easily extended with C/C++, which allows us to write computationally intensive code in C/C++ and create Python wrappers that can be used as Python modules. This gives us two advantages: first, the code is as fast as the original C/C++ code (since it is the actual C++ code working in background) and second, it easier to code in Python than C/C++. OpenCV-Python is a Python wrapper for the original OpenCV C++ implementation.

CHAPTER 7
CONCLUSION

7.1 CONCLUSION

The proposed methodology uses canny edge detection with hough transform to segment iris images for locating the iris and removes noise. Normalization method is used for unwrapping of iris to obtain polar coordinates of the image. Results of feature extraction is used for matching the features of the users who are logging into the system newly. Supervised learning Neural Network tool is used in order to increase the matching accuracy. Classification using Neural Network gave the best results in obtaining high accuracy rate. The final result of this system is decision making, where the user was verified as an authorized user or not through the comparison of the stored and calculated threshold values.

APPENDIX I

SOURCE CODE

Training:

```
import cv2
import numpy as np
import glob
import cv2
# import os
## getting the mask from the rgb images
def preprocessing(img):
    # resizing using aspect ratio intact and finding the circle
    # reduce size retain aspect ratio intact
    # invert BGR 2 RGB
    RGB = cv2.cvtColor(img,cv2.COLOR_BGR2RGB)
    Ig = RGB[:, :, 2]
    [w,h] = np.shape(Ig)
    r=1200.0/Ig.shape[1]
    dim=(1200,int(Ig.shape[0]*r))
```

```
rz = cv2.resize(Ig,dim,interpolation=cv2.INTER_AREA)
#  convert in to float and get log trasform for contrast streching
g = 0.2 * (np.log(1 + np.float32(rz)))
print (g[0][0])
# change into uint8
cvuint = cv2.convertScaleAbs(g)
# cvuint8.dtype
ret, th = cv2.threshold(cvuint, 0, 255, cv2.THRESH_OTSU)
ret1,th1 = cv2.threshold(Ig,0,255,cv2.THRESH_OTSU)
# closeing operation
# from skimage.morphology import disk
# from skimage.morphology import erosion, dilation, opening, closing, white_tophat
# selem = disk(30)
# cls = opening(th, selem)
# plot_comparison(orig_phantom, eroded, 'erosion')
# in case using opencv
kernel = cv2.getStructuringElement(cv2.MORPH_ELLIPSE, (35,35))
cls = cv2.morphologyEx(th, cv2.MORPH_OPEN, kernel)
Im = cls*rz # the mask with resize image
```

```
    # cv2.imwrite('mynew.jpg', mask)
    return (Im,th,th1,cls,g,RGB)
path_dir = glob.glob('*.jpg')
for k in path_dir:
    #print('running code....',k)
    img = cv2.imread(k)
    (Im,th,th1,cls,g,RGB) = preprocessing(img)
    from matplotlib import pyplot as plt
    # plt.imshow(cls)
    # plt.show()
    # cv2.imshow('asd',cls)
    # cv2.waitKey(0)
    # cv2.destroyAllWindows()
    # plot the data
    titles = ['Original Image', 'log_transform','mask using logT','mask without log_T ']
    images = [RGB,g,cls,th]
    #print(cls)
    for i in range(0,np.size(images)):
        print(i)
```

```
        plt.subplot(2, 3, i + 1)
        plt.imshow((images[i]),'gray')
        plt.title(titles[i])
        plt.xticks([]), plt.yticks([])
    plt.show()
```

Testing:

```
 import glob
import cv2
import numpy as np
from scipy.special import expit
import sys
# import os
## getting the mask from the rgb images
def preprocessing(img):
    # resizing using aspect ratio intact and finding the circle
    # invert BGR 2 RGB
    RGB = cv2.cvtColor(img,cv2.COLOR_BGR2RGB)
    Ig = RGB[:, :, 2]
    [w,h] = np.shape(Ig)
```

```
r=1200.0/Ig.shape[1]
dim=(1200,int(Ig.shape[0]*r))
rz = cv2.resize(Ig,dim,interpolation=cv2.INTER_AREA)
#  convert in to float and get log trasform for contrast streching
g = 0.2 * (np.log(1 + np.float32(rz)))
if str(g[0][0]) == '1.0150348':
  print ('welcom mr rajesh your access sucessfully')
if str(g[0][0]) == '0.91699356':
    print ('welcom mr b your access sucessfully')
if str(g[0][0]) == '1.003456':
    print ('welcom mr c your access sucessfully')
  # change into uint8
cvuint = cv2.convertScaleAbs(g)
# cvuint8.dtype
ret, th = cv2.threshold(cvuint, 0, 255, cv2.THRESH_OTSU)
ret1,th1 = cv2.threshold(Ig,0,255,cv2.THRESH_OTSU)
# closeing operation
# from skimage.morphology import disk
# from skimage.morphology import erosion, dilation, opening, closing, white_tophat
```

```
# selem = disk(30)

# cls = opening(th, selem)

# plot_comparison(orig_phantom, eroded, 'erosion')

# in case using opencv

kernel = cv2.getStructuringElement(cv2.MORPH_ELLIPSE, (35,35))

cls = cv2.morphologyEx(th, cv2.MORPH_OPEN, kernel)

Im = cls*rz # the mask with resize image

# cv2.imwrite('mynew.jpg', mask)

return (Im,th,th1,cls,g,RGB)

path_dir = glob.glob('*.jpg')

 for k in path_dir:

#print('running code....',k)

img = cv2.imread(k)

(Im,th,th1,cls,g,RGB) = preprocessing(img)

from matplotlib import pyplot as plt

# plt.imshow(cls)

# plt.show()

# cv2.imshow('asd',cls)

# cv2.waitKey(0)

# cv2.destroyAllWindows()
```

```
    # plot the data
    titles = ['Original Image', 'log_transform','mask using logT','mask without log_T ']
    images = [RGB,g,cls,th]
    #print(cls)
    for i in range(0,np.size(images)):
        #print(i)
        plt.subplot(2, 3, i + 1)
        plt.imshow((images[i]),'gray')
        plt.title(titles[i])
        plt.xticks([]), plt.yticks([])
    plt.show()
class NeuralNetMLP(object):
    """ Feedforward neural network / Multi-layer perceptron classifier.
    Parameters
    ------------
    n_output : int
        Number of output units, should be equal to the
        number of unique class labels.
    n_features : int
```

Number of features (dimensions) in the target dataset.

Should be equal to the number of columns in the X array.

n_hidden : int (default: 30)

Number of hidden units.

l1 : float (default: 0.0)

Lambda value for L1-regularization.

No regularization if l1=0.0 (default)

l2 : float (default: 0.0)

Lambda value for L2-regularization.

No regularization if l2=0.0 (default)

epochs : int (default: 500)

Number of passes over the training set.

eta : float (default: 0.001)

Learning rate.

alpha : float (default: 0.0)

Momentum constant. Factor multiplied with the

gradient of the previous epoch t-1 to improve

learning speed

w(t) := w(t) - (grad(t) + alpha*grad(t-1))

decrease_const : float (default: 0.0)

```
    Decrease constant. Shrinks the learning rate
    after each epoch via eta / (1 + epoch*decrease_const)
  shuffle : bool (default: True)
    Shuffles training data every epoch if True to prevent circles.
  minibatches : int (default: 1)
    Divides training data into k minibatches for efficiency.
    Normal gradient descent learning if k=1 (default).
  random_state : int (default: None)
    Set random state for shuffling and initializing the weights.
Attributes
  -----------
  cost_ : list
    Sum of squared errors after each epoch.
  """
  def __init__(self, n_output, n_features, n_hidden=30,
               l1=0.0, l2=0.0, epochs=500, eta=0.001,
               alpha=0.0, decrease_const=0.0, shuffle=True,
               minibatches=1, random_state=None):
    np.random.seed(random_state)
    self.n_output = n_output
```

```
    self.n_features = n_features
    self.n_hidden = n_hidden
    self.w1, self.w2 = self._initialize_weights()
    self.l1 = l1
    self.l2 = l2
    self.epochs = epochs
    self.eta = eta
    self.alpha = alpha
    self.decrease_const = decrease_const
    self.shuffle = shuffle
    self.minibatches = minibatches
def _encode_labels(self, y, k):
    """Encode labels into one-hot representation
    Parameters
    ------------
    y : array, shape = [n_samples]
       Target values.

    Returns
    -----------
```

```
    onehot : array, shape = (n_labels, n_samples)
    """
    onehot = np.zeros((k, y.shape[0]))
    for idx, val in enumerate(y):
        onehot[val, idx] = 1.0
    return onehot
def _initialize_weights(self):
    """Initialize weights with small random numbers."""
    w1 = np.random.uniform(-1.0, 1.0,
                           size=self.n_hidden*(self.n_features + 1))
    w1 = w1.reshape(self.n_hidden, self.n_features + 1)
    w2 = np.random.uniform(-1.0, 1.0,
                           size=self.n_output*(self.n_hidden + 1))
    w2 = w2.reshape(self.n_output, self.n_hidden + 1)
    return w1, w2
def _sigmoid(self, z):
    """Compute logistic function (sigmoid)
    Uses scipy.special.expit to avoid overflow
    error for very small input values z.
    """
```

```
    # return 1.0 / (1.0 + np.exp(-z))
    return expit(z)
def _sigmoid_gradient(self, z):
    """Compute gradient of the logistic function"""
    sg = self._sigmoid(z)
    return sg * (1.0 - sg)
def _add_bias_unit(self, X, how='column'):
    """Add bias unit (column or row of 1s) to array at index 0"""
    if how == 'column':
        X_new = np.ones((X.shape[0], X.shape[1] + 1))
        X_new[:, 1:] = X
    elif how == 'row':
        X_new = np.ones((X.shape[0] + 1, X.shape[1]))
        X_new[1:, :] = X
    else:
        raise AttributeError('`how` must be `column` or `row`')
    return X_new
def _feedforward(self, X, w1, w2):
    """Compute feedforward step
    Parameters
```

X : array, shape = [n_samples, n_features]

Input layer with original features.

w1 : array, shape = [n_hidden_units, n_features]

Weight matrix for input layer -> hidden layer.

w2 : array, shape = [n_output_units, n_hidden_units]

Weight matrix for hidden layer -> output layer.

Returns

----

a1 : array, shape = [n_samples, n_features+1]

Input values with bias unit.

z2 : array, shape = [n_hidden, n_samples]

Net input of hidden layer.

a2 : array, shape = [n_hidden+1, n_samples]

Activation of hidden layer.

z3 : array, shape = [n_output_units, n_samples]

Net input of output layer.

a3 : array, shape = [n_output_units, n_samples]

Activation of output layer.

"""

```
    a1 = self._add_bias_unit(X, how='column')
    z2 = w1.dot(a1.T)
    a2 = self._sigmoid(z2)
    a2 = self._add_bias_unit(a2, how='row')
    z3 = w2.dot(a2)
    a3 = self._sigmoid(z3)
    return a1, z2, a2, z3, a3
def _L2_reg(self, lambda_, w1, w2):
    """Compute L2-regularization cost"""
    return (lambda_/2.0) * (np.sum(w1[:, 1:] ** 2) +
                            np.sum(w2[:, 1:] ** 2))
def _L1_reg(self, lambda_, w1, w2):
    """Compute L1-regularization cost"""
    return (lambda_/2.0) * (np.abs(w1[:, 1:]).sum() +
                            np.abs(w2[:, 1:]).sum())
def _get_cost(self, y_enc, output, w1, w2):
    """Compute cost function.
    Parameters
    ----------
    y_enc : array, shape = (n_labels, n_samples)
```

```
        one-hot encoded class labels.
    output : array, shape = [n_output_units, n_samples]
        Activation of the output layer (feedforward)
    w1 : array, shape = [n_hidden_units, n_features]
        Weight matrix for input layer -> hidden layer.
    w2 : array, shape = [n_output_units, n_hidden_units]
        Weight matrix for hidden layer -> output layer.
    Returns
    ---------
    cost : float
        Regularized cost.
    """
    term1 = -y_enc * (np.log(output))
    term2 = (1.0 - y_enc) * np.log(1.0 - output)
    cost = np.sum(term1 - term2)
    L1_term = self._L1_reg(self.l1, w1, w2)
    L2_term = self._L2_reg(self.l2, w1, w2)
    cost = cost + L1_term + L2_term
    return cost
def _get_gradient(self, a1, a2, a3, z2, y_enc, w1, w2):
```

""" Compute gradient step using backpropagation.

Parameters

----

a1 : array, shape = [n_samples, n_features+1]

Input values with bias unit.

a2 : array, shape = [n_hidden+1, n_samples]

Activation of hidden layer.

a3 : array, shape = [n_output_units, n_samples]

Activation of output layer.

z2 : array, shape = [n_hidden, n_samples]

Net input of hidden layer.

y_enc : array, shape = (n_labels, n_samples)

one-hot encoded class labels.

w1 : array, shape = [n_hidden_units, n_features]

Weight matrix for input layer -> hidden layer.

w2 : array, shape = [n_output_units, n_hidden_units]

Weight matrix for hidden layer -> output layer.

Returns

----

grad1 : array, shape = [n_hidden_units, n_features]

```
        Gradient of the weight matrix w1.
    grad2 : array, shape = [n_output_units, n_hidden_units]
        Gradient of the weight matrix w2.
    """
    # backpropagation
    sigma3 = a3 - y_enc
    z2 = self._add_bias_unit(z2, how='row')
    sigma2 = w2.T.dot(sigma3) * self._sigmoid_gradient(z2)
    sigma2 = sigma2[1:, :]
    grad1 = sigma2.dot(a1)
    grad2 = sigma3.dot(a2.T)
    # regularize
    grad1[:, 1:] += self.l2 * w1[:, 1:]
    grad1[:, 1:] += self.l1 * np.sign(w1[:, 1:])
    grad2[:, 1:] += self.l2 * w2[:, 1:]
    grad2[:, 1:] += self.l1 * np.sign(w2[:, 1:])
    return grad1, grad2
def predict(self, X):
    """Predict class labels
    Parameters
```

```
    ----------
    X : array, shape = [n_samples, n_features]
      Input layer with original features.
    Returns:
    ----------
    y_pred : array, shape = [n_samples]
      Predicted class labels.
    """
    if len(X.shape) != 2:
      raise AttributeError('X must be a [n_samples, n_features] array.\n'
                           'Use X[:,None] for 1-feature classification,'
                           '\nor X[[i]] for 1-sample classification')

    a1, z2, a2, z3, a3 = self._feedforward(X, self.w1, self.w2)
    y_pred = np.argmax(z3, axis=0)
    return y_pred
  def fit(self, X, y, print_progress=False):
    """ Learn weights from training data.
    Parameters
    ----------
```

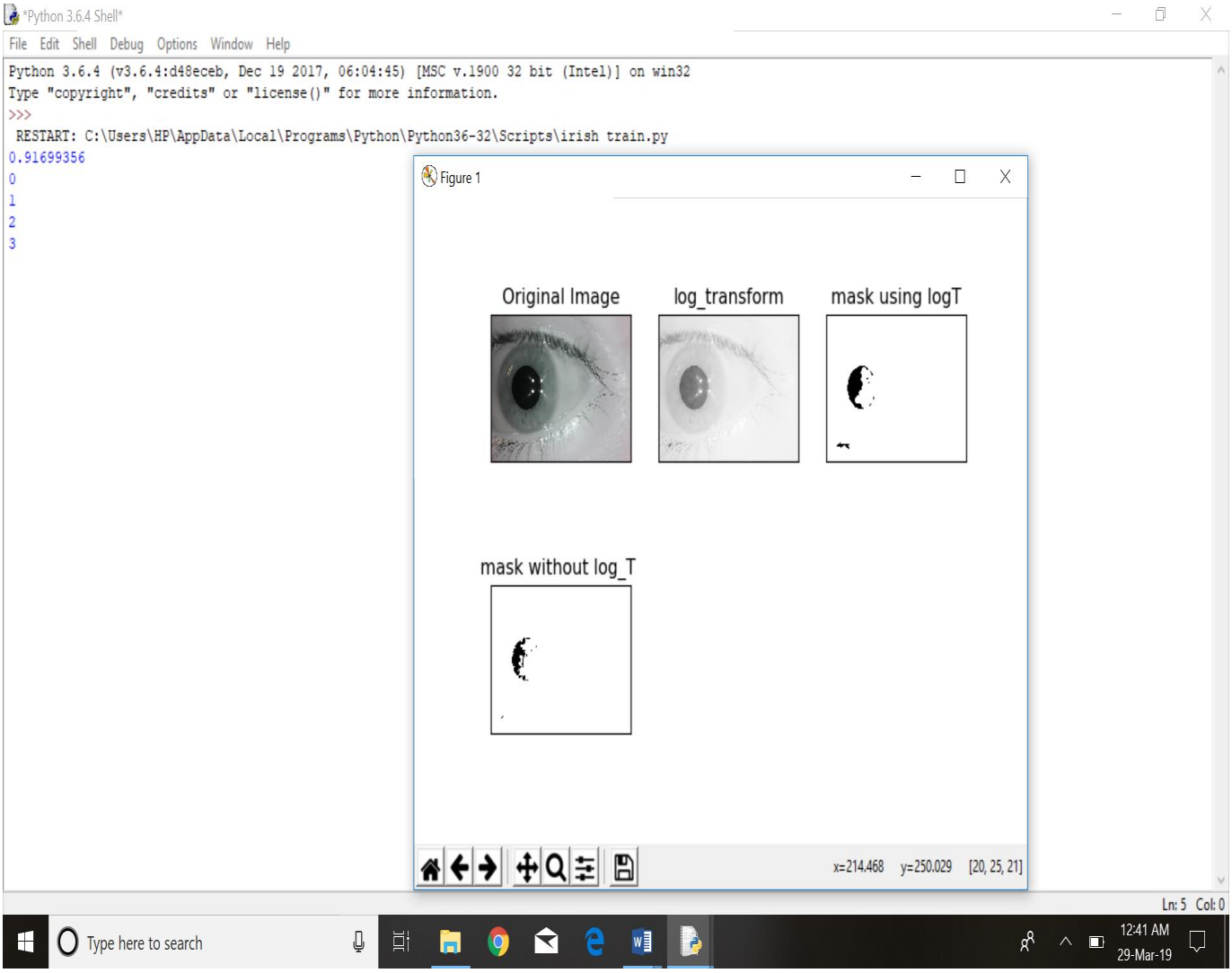
```
X : array, shape = [n_samples, n_features]
    Input layer with original features.
y : array, shape = [n_samples]
    Target class labels.
print_progress : bool (default: False)
    Prints progress as the number of epochs
    to stderr.
Returns:
----------
self
"""
self.cost_ = []
X_data, y_data = X.copy(), y.copy()
y_enc = self._encode_labels(y, self.n_output)
delta_w1_prev = np.zeros(self.w1.shape)
delta_w2_prev = np.zeros(self.w2.shape)
for i in range(self.epochs):
    # adaptive learning rate
    self.eta /= (1 + self.decrease_const*i)
    if print_progress:
```

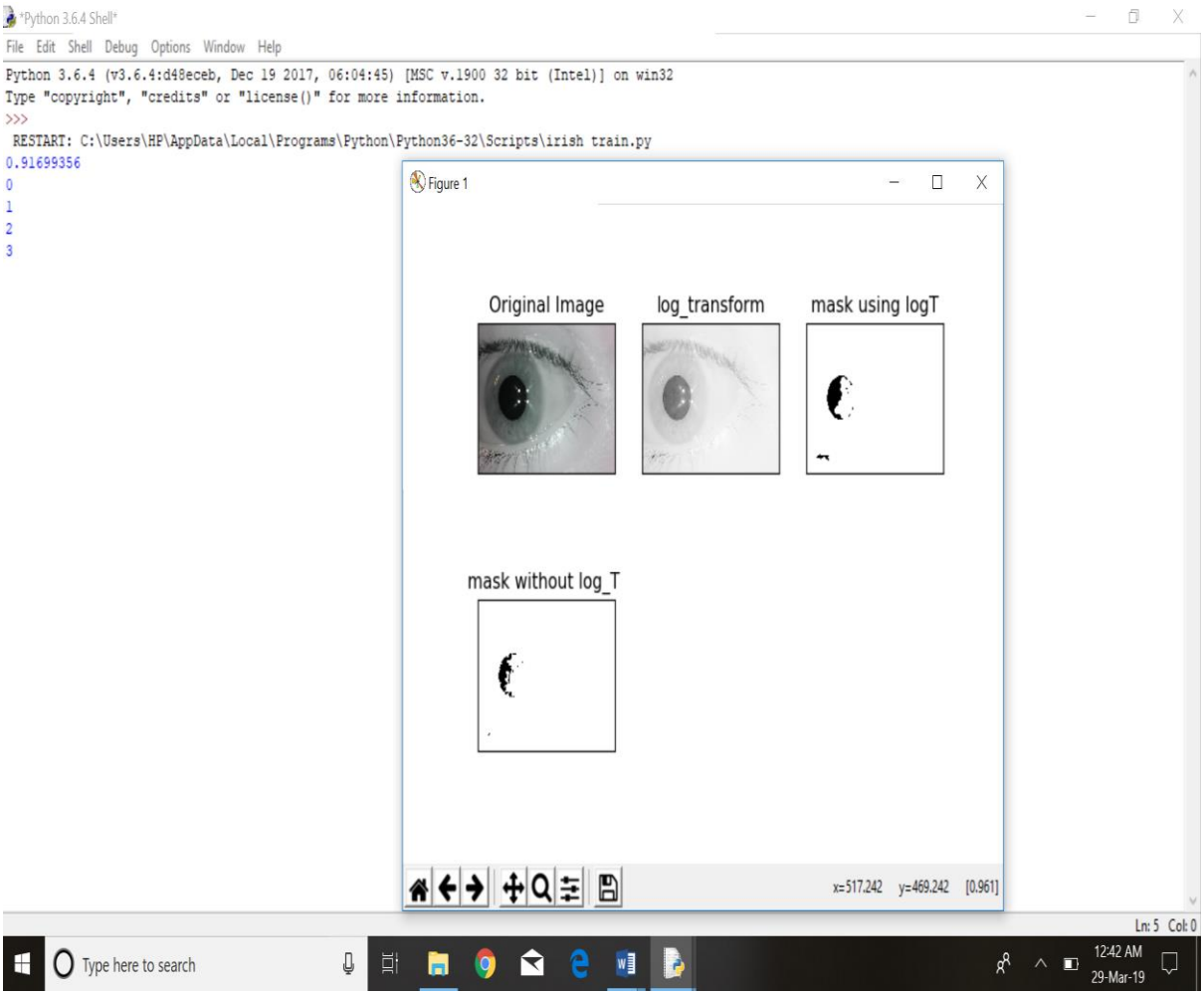
```
        sys.stderr.write('\rEpoch: %d/%d' % (i+1, self.epochs))
        sys.stderr.flush()
    if self.shuffle:
        idx = np.random.permutation(y_data.shape[0])
        X_data, y_enc = X_data[idx], y_enc[:, idx]
    mini = np.array_split(range(y_data.shape[0]), self.minibatches)
    for idx in mini:
        # feedforward
        a1, z2, a2, z3, a3 = self._feedforward(X_data[idx],
                                               self.w1,
                                               self.w2)
        cost = self._get_cost(y_enc=y_enc[:, idx],
                              output=a3,
                              w1=self.w1,
                              w2=self.w2)
        self.cost_.append(cost)
        # compute gradient via backpropagation
        grad1, grad2 = self._get_gradient(a1=a1, a2=a2,
                                          a3=a3, z2=z2,
                                          y_enc=y_enc[:, idx],
```

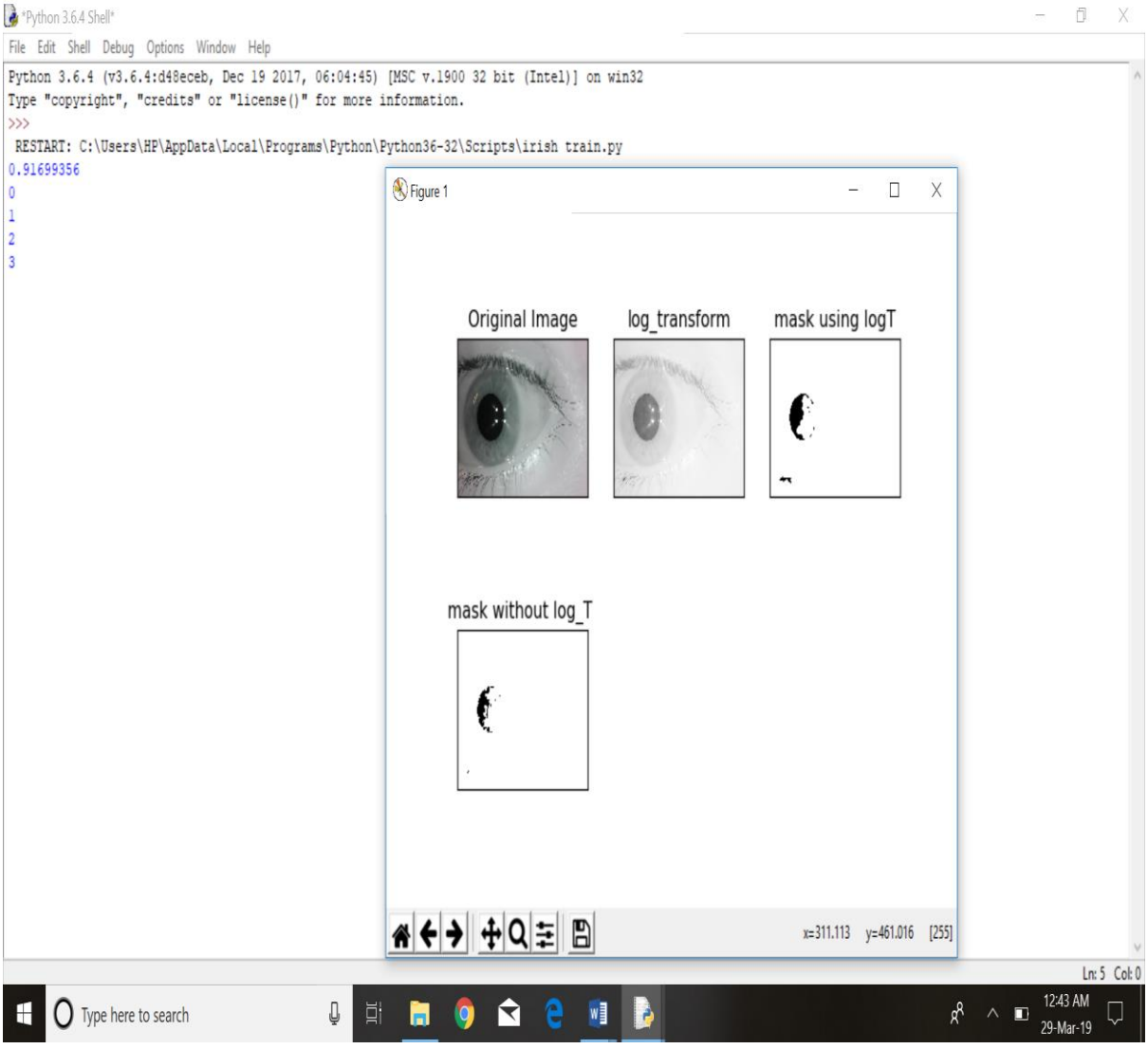
```
                    w1=self.w1,
                    w2=self.w2)
        delta_w1, delta_w2 = self.eta * grad1, self.eta * grad2
        self.w1 -= (delta_w1 + (self.alpha * delta_w1_prev))
        self.w2 -= (delta_w2 + (self.alpha * delta_w2_prev))
        delta_w1_prev, delta_w2_prev = delta_w1, delta_w2
return self
```

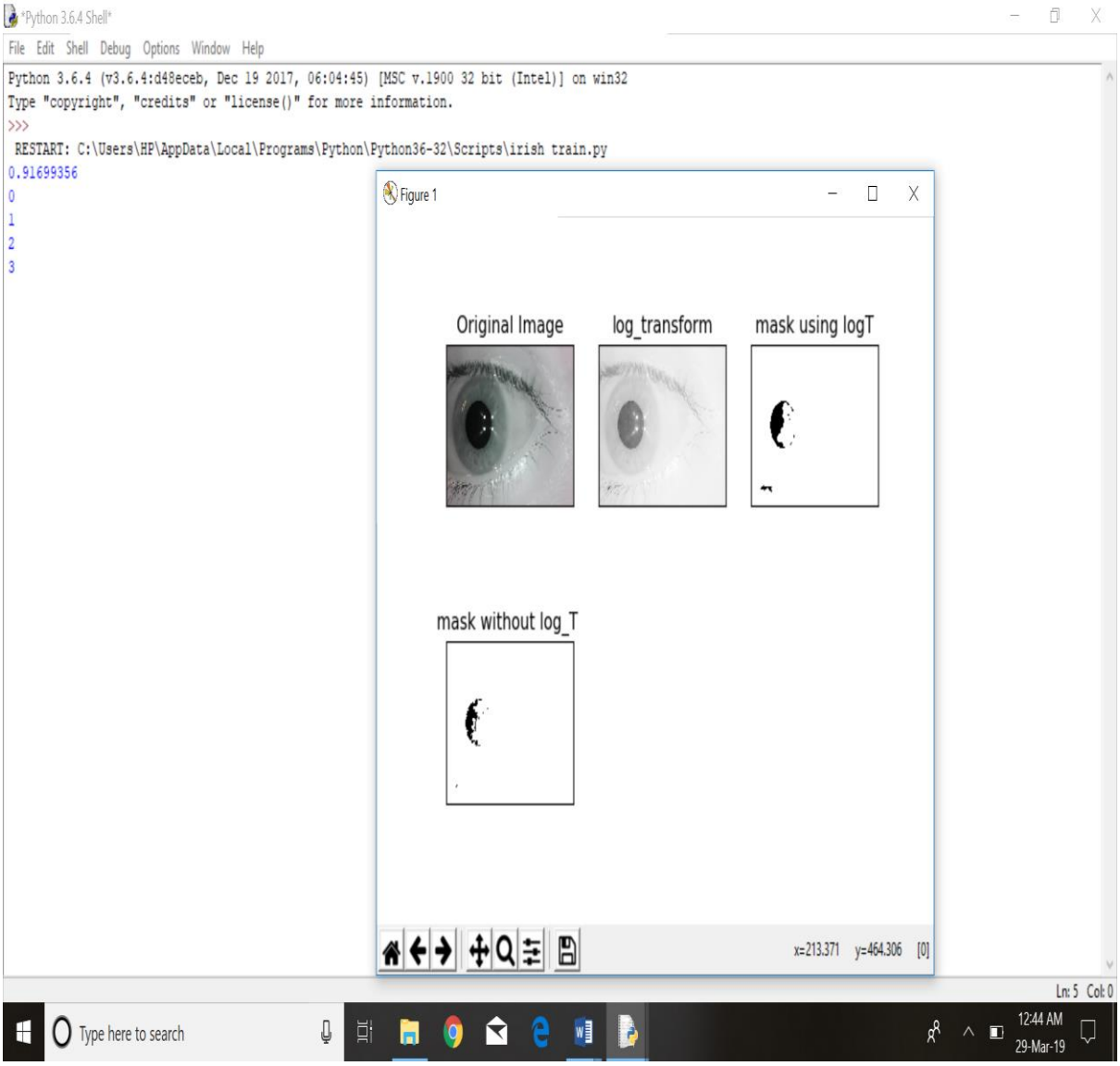
APPENDIX II

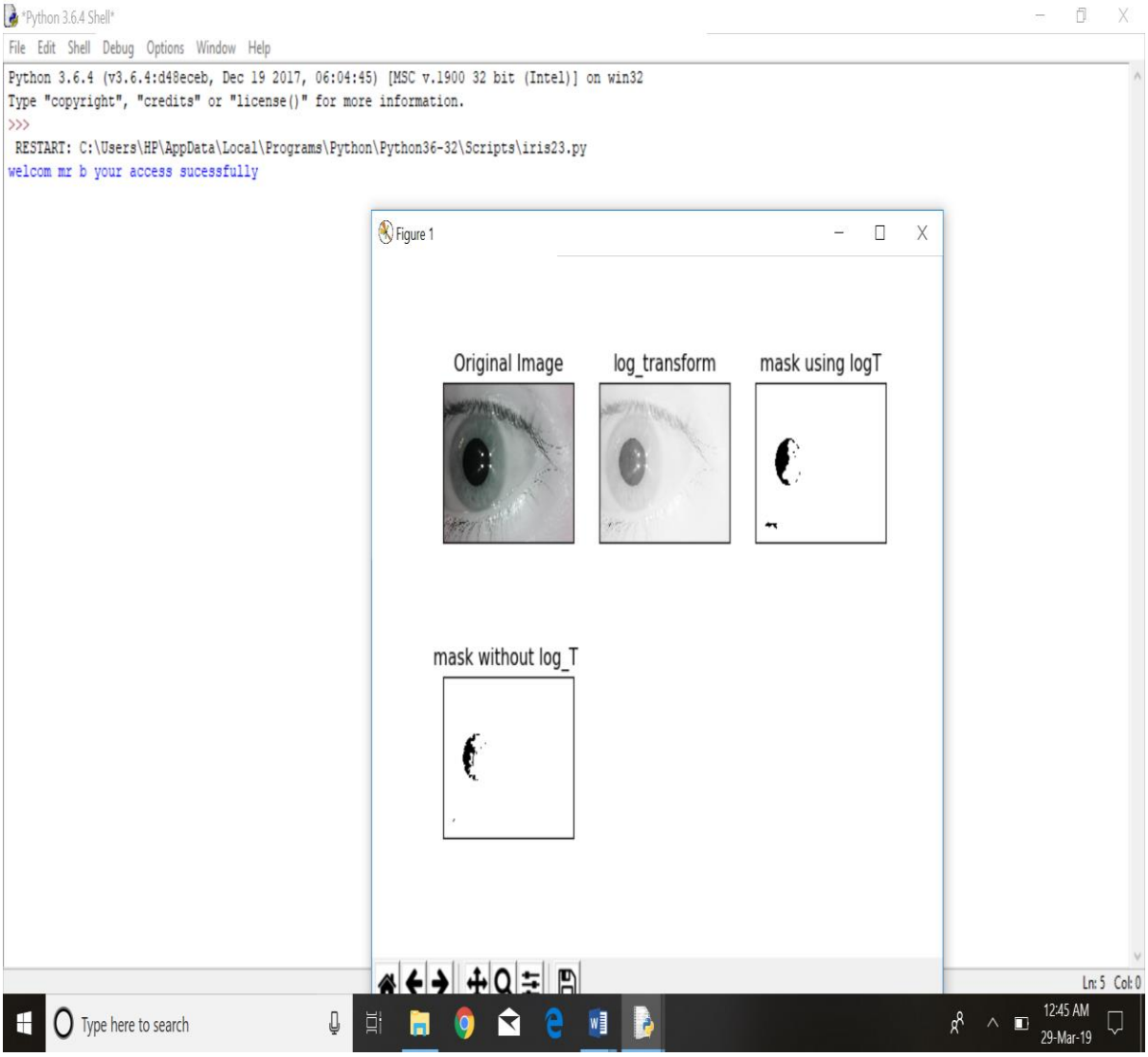
SCREENSHOTS




Python 3.6.4 Shell
File Edit Shell Debug Options Window Help
Python 3.6.4 (v3.6.4:d48eceb, Dec 19 2017, 06:04:45) [MSC v.1900 32 bit (Intel)] on win32
Type "copyright", "credits" or "license()" for more information.
>>>
RESTART: C:\Users\HP\AppData\Local\Programs\Python\Python36-32\Scripts\irish train.py
0.91699356
0
1
2
3
Figure 1
Original Image
log_transform
mask using logT
mask without log_T
x=517.242 y=469.242 [0.961]
Ln: 5 Col: 0
Type here to search
12:42 AM
29-Mar-19


Python 3.6.4 Shell
File Edit Shell Debug Options Window Help
Python 3.6.4 (v3.6.4:d48eceb, Dec 19 2017, 06:04:45) [MSC v.1900 32 bit (Intel)] on win32
Type "copyright", "credits" or "license()" for more information.
>>>
RESTART: C:\Users\HP\AppData\Local\Programs\Python\Python36-32\Scripts\irish train.py
0.91699356
0
1
2
3
Figure 1
Original Image
log_transform
mask using logT
mask without log_T
x=311.113 y=461.016 [255]
Ln: 5 Col: 0
Type here to search
12:43 AM
29-Mar-19


Python 3.6.4 Shell
File Edit Shell Debug Options Window Help
Python 3.6.4 (v3.6.4:d48eceb, Dec 19 2017, 06:04:45) [MSC v.1900 32 bit (Intel)] on win32
Type "copyright", "credits" or "license()" for more information.
>>>
RESTART: C:\Users\HP\AppData\Local\Programs\Python\Python36-32\Scripts\irish train.py
0.91699356
0
1
2
3
Figure 1
Original Image
log_transform
mask using logT
mask without log_T
x=213.371 y=464.306 [0]
Ln: 5 Col: 0
Type here to search
12:44 AM
29-Mar-19


Python 3.6.4 Shell
File Edit Shell Debug Options Window Help
Python 3.6.4 (v3.6.4:d48eceb, Dec 19 2017, 06:04:45) [MSC v.1900 32 bit (Intel)] on win32
Type "copyright", "credits" or "license()" for more information.
>>>
RESTART: C:\Users\HP\AppData\Local\Programs\Python\Python36-32\Scripts\iris23.py
welcom mr b your access sucessfully
Figure 1
Original Image
log_transform
mask using logT
mask without log_T
Ln: 5 Col: 0
Type here to search
12:45 AM
29-Mar-19

REFERENCES

1. Fatin Zuriah Binti Mohammad Zuhary..Iris Recognition Using Aritificial neural network, June 2014
2. C.Soma Sundar Reddy, K.Durga Srinivas..Recognition Of Iris For Person Identification.ISSN:2248-9622. Vol. 3. 2013
3. Sudha Gupta, Asst. Professor ,LMIETE, LMISTE, Viral Doshi, Abhinav Jain and Sreeram Iyer, K.J.S.C.E.. Iris Recognition System using Biometric Template Matching Technology, Volume 1 – No. 2
4. Nurdan Akhan, Yuksel Ozbay.. Iris Recognition Using Neural Network, August 2006.
5. Omaima Al-Allaf , Abdelfatah A Tamimi.. Artificial Neural Networks for Iris Recognition System: Comparisons between Different Models, Architectures and Algorithms, November 2012.
6. Idrus, S.Z.S., Cherrier, E., Rosenberger, C., Bours, P.. Soft biometrics for keystroke dynamics: Profiling individuals while typing passwords. Computers & Security 2014;45:147–155.
7. Hayes, J.. Identifying fatigue through keystroke dynamics.The UNSW Canberra at ADFA Journal of Undergraduate Engineering Research 2018;9(2).
8. Roy, S., Roy, U., Sinha, D.. Protection of kids from internet threats: A machine learning approach for classification of age-group based on typing pattern. In: Proceedings of the International MultiConference of Engineers and Computer Scientists; vol. 1. 2018, .
9. Epp, C., Lippold, M., Mandryk, R.L.. Identifying emotional states using keystroke dynamics. In: Proceedings of the SIGCHI Conference on Human Factors in Computing Systems. ACM; 2011, p. 715–724.

10. Dr.Sanjay R.Ganorkar, Mirza Shujaur Rahman.. Iris Recognition System and analysis Using Neural Netwok.ISSN:2278-0181.vol. 2. 2013
11. Giot, R., El-Abed, M., Hemery, B., Rosenberger, C.. Unconstrained keystroke dynamics authentication with shared secret. Computers &Security 2011;30(6-7):427–445.