1 ABSTRACT:

The analysis of handwritten documents for identifying the writer has great bearing on the criminal justice system. Lack of dynamic information like order of strokes,speed of writing, acceleration etc. makes it difficult task and this is what we are trying solve here. The project aims at extracting distinguishable handwritten features from collected handwritten documents and thereby using the information to distinguish the true and distorted handwritten behaviour of the same person

2 CONTENTS

3 CHAPTER : INTRODUCTION:

3.1 INTRODUCTION:

The computerized system makes it possible to compare handwriting under different conditions; therefore we will compare the handwriting of the same individuals when asked to write truthful and deceptive sentences. Our research hypothesis is that differences will be found between writing of truthful sentences and writing of distorted sentences in pressure, temporal (stroke duration on digitizer and in air) and spatial measures (strokes path length, height and width) obtained by the computerized system. Based on the finding of the studies above we can predict that in deceptive writing, the mean and standard deviations of handwriting measures of each participant will be varied. Thus while writing deceptive sentences, higher pressure will be implemented, longer duration time per stroke (on paper and in air) will be required, and letter strokes will be larger in comparison to truthful writing.

3.2 BACKGROUND OF THE STUDY:

The electronic framework makes it conceivable to look at under changed conditions; thusly we will think about the penmanship of similar people when requested to compose honest and tricky sentences. Our exploration theory is that distinctions will be found between composing of honest sentences and composing of mutilated sentences in weight, fleeting (stroke span on digitizer and in air) and spatial measures (strokes way length, tallness and width) acquired by the automated framework. In view of the finding of the investigations above we can anticipate that in beguiling written work, the mean and standard deviations of penmanship measures of every member will be shifted. Subsequently while composition tricky sentences, higher weight will be actualized, longer length time per stroke (on paper and in air) will be required, and letter strokes will be bigger in contrast with honest written work.

3.3 SIGNIFICANCE OF THE STUDY:

Offline handwritten document processing to predict deception tool is currently being used in more than 50 countries in the fields of corrections, criminal investigations, intelligence/counter intelligence and civil matters. In the United States alone all federal law enforcement agencies either employ their own deception examiners or use the services of examiners employed in other agencies. Examiners and quality control programs exist in the FBI, US. Secret Service, US Army CID, US Marine Corps CID, Air Force OSI, Navy NCIS, US Customs, US Marshals, Defence Criminal Investigation Service, Internal Revenue Service, US Capitol Police, Food & Drug Administration, Department of Energy, Central Intelligence Agency, Police & County Sheriff's departments, sex therapists and numerous other investigative bodies.

3.4 DOCUMENT CONVENTIONS

This document follows Calibri Format. Bold-faced text has been used to emphasize section and sub-section headings. Highlighting is to point out words in the glossary and italicized text is used to label and recognize diagrams.

3.5 RESEARCH AIM:

The goal of this study is to compare and analyse the handwriting behaviour of true and false or distorted writing. Based on the cognitive load known to be experienced while communicating a deceptive message, we will hypothesized a difference (in temporal and spatial, pressure measures and peak velocities) between the handwriting of true vs. distorted messages which are taken as image format. We would try to evaluate brain-hand performance, as manifested through handwriting behaviour which is a valid measure for detecting the dis-automaticity that is indicative of detecting deception.

4 CHAPTER : FUNCTIONALITIES:

Product Functions

The main functionalities of uisng this computersied tool are given bellow,

- To find distinguishable differences between true and distorted handwriting behaviour
- Testing informants to determine the veracity of information provided
- Narrowing the focus of enquiry
- Gathering additional information and evidence
- Assisting to focus the investigation on particular suspects

Operating Environment

Minimum Hardware Requirements:

1. Ram: - 2GB or Higher

2. Processor: Core i3

Minimum Software Requirements:

1. Platform:-
    a. Windows 7
    b. Linux

2. Coding Language:- Python

3. IDE Tool:-

    a. Spyder
    b. QTDesigner
    c. TKinter

Assumptions

- There is a GUI for processing the captured or recorded images which have to work properly in the system.
- All the algorithms of the defined features should be worked in convenient manner.
- Users know the English language, as the instructions of the user interface will be provided in English.

External Interface Requirements

User Interfaces

- User Authentication.

- Features like, file based operations (open, save as , save etc.) , edit (zoom in, zoom out, crop) ,options (Strock thickening, strok thining etc.) of the documents.

Hardware Interfaces

- A high-performance processing unit. We cannot guarantee multimedia components will work on slower power based unit. Unfortunately, we cannot distribute hard copies (e.g., CD – ROM, DVD – ROM) of multimedia items.

- RAM - 2GB or Higher

- Processor- Core i3 or Higher

Software Interfaces

1. Platform:

    - Windows 7

    - Linux

2. IDE Tool:

    - Spyder

    - QTDesigner

    - TKinter

Performance Requirements

System-Operation:

- Uploading captured images operation

    - Preprocessing of captured images

- feature extraction

- classification

System-Operation features :

- Image Operation efficiency

    - Image Operation security

    - Image Operation portability

System-Operation application :

- Governmental Domains

    - Requestor API

Safety Requirement

    - Disruption Handling (Information re-verification)

    - Avoidance of Hacking

Software Availability:

- Software should be used for types of documents like image.

- Software should be capable of providing the speed of allocation of documents to the user on demand

- Software must comprise ease of accessibility from the principle domain as well as the Third-Party cases.

Correctness:

The result obtained by software about documents and operations associated with documents should be correct and highly accurate.

The recognition systems that read cursive handwriting cannot guarantee the same levels of accuracy that systems deliver on machine print. However, they can meet the most demanding accuracy requirements by implementing highly sophisticated algorithms designed for that specific purpose.

Maintainability:

Data is maintain in JPEG, PNG file format, by which data can be easily maintained.

Usability:

Software can be used by all the user to achieve quantified objectives with effectiveness, efficiency, and satisfaction in a quantified context of use. These places the assurance by link handling and the ease allowable functionality.

Reliable:

The System is totally reliable by its allocated features and the result thus yielded are very well accomplished reliably.

There are multiple factors that influence the read rate and reliability in each particular case. This may include the quality of images, type of fields, type of documents, availability of the context, and many others. Advanced handwriting recognition systems exploit different powerful mechanisms that allow maximizing accuracy in each particular case. This includes voting mechanism, cross-validation of data, using context information.

Portability:

This is probably the most important feature of the system that ensures it on its functionality. The documents are well managed by the application that allows the user to carry the documents portably. Ease access from anywhere signifies the portability of the software.

5 CHAPTER : INSTALLATION:

Let's actualize fundamental segments in a well-ordered way keeping in mind the end goal to make a content arrangement structure in python. To begin with, import all the required libraries.

We need requisite libraries to run this code

- Pandas
- Scikit-learn
- Numpy
- QtDesigner

Pandas

pandas is an open source, BSD-licensed library providing high-performance, easy-to-use data structures and data analysis tools for the Python programming language.

pandas is a NumFOCUS sponsored project. This will help ensure the success of development of pandas as a world-class open-source project and makes it possible to donate to the project.

Way to Install

The best way to get pandas is via conda

- conda install pandas

Packages are available for all supported python versions on Windows, Linux, and MacOS.

What problem does pandas solve?

Python has long been great for data munging and preparation, but less so for data analysis and modelling. pandas helps fill this gap, enabling us to carry out our entire data analysis workflow in Python without having to switch to a more domain specific language like R.

Combined with the excellent IPython toolkit and other libraries, the environment for doing data analysis in Python excels in performance, productivity, and the ability to collaborate.

pandas does not implement significant modelling functionality outside of linear and panel regression; for this, look to statsmodels and scikit-learn. More work is still needed to make Python a first class statistical modeling environment, but we are well on our way toward that goal.

Scikit-learn

Scikit-learn provides a range of supervised and unsupervised learning algorithms via a consistent interface in Python.

It is licensed under a permissive simplified BSD license and is distributed under many Linux distributions, encouraging academic and commercial use.

The library is built upon the SciPy (Scientific Python) that must be installed before we can use scikit-learn. This stack that includes:

- NumPy:
  - Base n-dimensional array package
- SciPy:
  - Fundamental library for scientific computing

- Matplotlib:
    - Comprehensive 2D/3D plotting
- IPython:
    - Enhanced interactive console
- Sympy:
    - Symbolic mathematics
- Pandas:
    - Data structures and analysis

Extensions or modules for SciPy care conventionally named SciKits. As such, the module provides learning algorithms and is named scikit-learn.

The vision for the library is a level of robustness and support required for use in production systems. This means a deep focus on concerns such as easy of use, code quality, collaboration, documentation and performance.

Scikit-learn requires:

Python (>= 2.7 or >= 3.3),

NumPy (>= 1.8.2),

SciPy (>= 0.13.3).

The easiest way to install scikit-learn is using pip

- pip install -U scikit-learn

or

- conda: conda install scikit-learn

We can also install these using conda or pip. When using pip, please ensure that binary wheels are used, and NumPy and SciPy are not recompiled from source, which can happen when using particular configurations of operating system and hardware (such as Linux on a Raspberry Pi). Building numpy and scipy from source can be complex (especially on Windows) and requires careful configuration to ensure that they link against an optimized implementation of linear algebra routines. Instead, use a third-party distribution as described below.

We must install scikit-learn and its dependencies with pip, we can install it as scikit-learn[alldeps]. The most common use case for this is in a requirements.txt file used as part of an automated build process for a PaaS application or a Docker image. This option is not intended for manual installation from the command line.

Components of scikit-learn: Scikit-learn comes loaded with a lot of features. Here are a few of them to help we understand the spread:

•Supervised learning algorithms: Think of any supervised learning algorithm we might have heard about and there is a very high chance that it is part of scikit-learn. Starting from Generalized linear models (e.g. Linear Regression), Support Vector Machines (SVM), Decision Trees to Bayesian methods – all of them are part of scikit-learn toolbox. The spread of algorithms is one of the big reasons for high usage of scikit-learn.

Cross-validation:

There are various methods to check the accuracy of supervised models on unseen data.

Feature extraction:

Useful for extracting features from images and text (e.g. Bag of words)

The library is focused on modelling data. It is not focused on loading, manipulating and summarizing data. For these features, refer to NumPy and Pandas.

Some popular groups of models provided by scikit-learn include:

- Cross Validation: for estimating the performance of supervised models on unseen data.
- Datasets: for test datasets and for generating datasets with specific properties for investigating model behaviour.
- Feature extraction: for defining attributes in image and text data.
- Supervised Models: a vast array not limited to generalized linear models, discriminate analysis, naive bayes, lazy methods, neural networks, support vector machines and decision trees.

NumPy

NumPy is a general-purpose array-processing package. It provides a high-performance multidimensional array object, and tools for working with these arrays.

Arbitrary data-types can be defined using Numpy which allows NumPy to seamlessly and speedily integrate with a wide variety of databases.

Installation:

- Mac and Linux users can install NumPy via pip command:
- pip install numpy
- Windows does not have any package manager analogous to that in linux or mac.

Please download the pre-built windows installer for NumPy according to wer system configuration and Python version). Then install the packages manually.

OpenCv:

OpenCV (Open Source Computer Vision Library) is released under a BSD license and hence it's free for both academic and commercial use. It has C++, Python and Java interfaces and supports Windows, Linux, Mac OS, iOS and Android. OpenCV was designed for computational efficiency and with a strong focus on real-time applications. Written in optimized C/C++, the library can take advantage of multi-core processing. Enabled with OpenCL, it can take advantage of the hardware acceleration of the underlying heterogeneous compute platform.

This method serves best when using just for programming and developing OpenCV applications.

Install package python-opencv with following command in terminal (as root user).

- $ sudo apt-get install python-opencv

Open Python IDLE (or IPython) and type following codes in Python terminal.

- import cv2 as cv
- print(cv.__version__)

If the results are printed out without any errors, congratulations !!! You have installed OpenCV-Python successfully.

It is quite easy. But there is a problem with this. Apt repositories may not contain the latest version of OpenCV always. For example, at the time of writing this tutorial, apt repository contains 2.4.8 while latest OpenCV version is 3.x. With respect to Python API, latest version will always contain much better support and latest bug fixes.

QtDesigner:

Qt Designer is the Qt tool for designing and building graphical user interfaces (GUIs) with Qt Widgets. We can compose and customize we windows or dialogs in a what-we-see-is-what-we-get (WYSIWYG) manner, and test them using different styles and resolutions.

Widgets and forms created with Qt Designer integrate seamlessly with programmed code, using Qt's signals and slots mechanism, so that we can easily assign behavior to graphical elements. All properties set in Qt Designer can be changed dynamically within the code. Furthermore, features like widget promotion and custom plugins allow we to use we own components with Qt Designer.

This given process should work,

- sudo apt-get install python-qt4 qt4-designer

You can find it if you search it with apt-cache search qt | grep designer:

- libqt4-designer - Qt 4 designer module
- libqt4-designer-dbg - Qt 4 designer library debugging symbols
- qt4-designer - graphical designer for Qt 4 applications
- kdesignerplugin - Integration of KF5 widgets in Qt Designer/Creator
- kdesignerplugin-data - Integration of KF5 widgets in Qt Designer/Creator
- kgendesignerplugin - Integration of KF5 widgets in Qt Designer/Creator
- libopenrpt-dev - graphical SQL report writer, designer and rendering engine (development)
- libopenrpt1v5 - graphical SQL report writer, designer and rendering library
- libqscintilla2-designer - Qt4 Designer plugin for QScintilla 2
- libqscintilla2-designer-dbg - Qt4 Designer plugin for QScintilla 2 (debug)
- libqt5designer5 - Qt 5 designer module
- libqt5designercomponents5 - Qt 5 Designer components module
- libqt5scintilla2-designer - Qt5 Designer plugin for QScintilla 2
- libqt5scintilla2-designer-dbg - Qt5 Designer plugin for QScintilla 2 (debug)
- libqxt-designer0 - LibQxt extensions to Qt Designer
- openrpt - graphical SQL report writer, designer and rendering engine

GUI:

We created out GUI with the aid of QtDesigner.

First of all, the '.ui' file is created and it needs to convert into '.py'.

Conversion Process:

pyuic5 input.ui -o output.py

Then, we convert the resource file, '.qrc' into '.py'.

Conversion Process:

pyrcc5 -o resource_r.py resource.qrc

Code:

```
from PyQt5 import QtCore, QtGui, QtWidgets
##
from PyQt5.QtCore import *
from PyQt5.QtGui import *
from PyQt5.QtWidgets import QApplication, QWidget
from PyQt5.QtWidgets import *

import cv2
import numpy as np
import matplotlib.pyplot as plt

from tkinter import *
import tkinter as tk
from tkinter import filedialog

from PIL import Image
from PIL import ImageTk
from PIL import ImageQt

import numpy as np
from skimage import io, color, img_as_ubyte
from skimage.feature import greycomatrix, greycoprops
from sklearn.metrics.cluster import entropy
from skimage.feature import greycomatrix, greycoprops

from scipy.ndimage import imread
import sys

####NEW ADDD###
from OtherWindow import Ui_OtherWindow
##


class Ui_MainWindow(object):
    def __init__(self):
        self.MainWindow = QtWidgets.QMainWindow()

    def openWindow(self):
        self.window = QtWidgets.QMainWindow()
        self.ui = Ui_OtherWindow()
        self.ui.setupUi(self.window)
```

```
        self.window.show()

######FILE OPEN LEFT#######
    def fOpenCheck(self):
        print("Success File Open")
        root = tk.Tk()
        root.withdraw()
        file_path = filedialog.askopenfilename()
        print(file_path)

        #img_zi = Image.open(file_path)
        #area_zi = (10000,1000,20000,3000)
        #zoom_img_zi = img_zi.crop(area_zi)
        #img_zi.show()
        #tkimage = ImageTk.PhotoImage(img_zi)
        #tk.Label(root, image=tkimage).pack()
        #return img_zi
        #framed = QGraphicsView()
        #file_path = imread(file_path)
        #height, width, channels = file_path.shape
        #bytesPerLine = channels * width
        #qImg = QImage(file_path.data, width, height, bytesPerLine, QImage.Format_RGB888)
        #pixmap01 = QPixmap.fromImage(qImg)
        #pixmap_image = QPixmap(pixmap01)
        img = Image.open(file_path)
        w, h = img.size
        self.imgQ = ImageQt.ImageQt(img)  # we need to hold reference to imgQ, or it will crash
        pixMap = QPixmap.fromImage(self.imgQ)


        scene = QGraphicsScene()
        scene.addPixmap(pixMap)
        #scene.addPixmap(QPixmap(file_path))
        #scene.addPixmap(pixmap_image)
        self.frame.setScene(scene)
        self.frame.fitInView(QRectF(0, 0, w, h), Qt.KeepAspectRatio)
        self.frame.show()
######
######FILE OPEN Right#######
    def fOpenCheckR(self):
        print("Success File Open")
        root = tk.Tk()
        root.withdraw()
        file_path2 = filedialog.askopenfilename()
        print(file_path2)

        #img_zi = Image.open(file_path)
        #area_zi = (10000,1000,20000,3000)
        #zoom_img_zi = img_zi.crop(area_zi)
        #img_zi.show()
        #tkimage = ImageTk.PhotoImage(img_zi)
        #tk.Label(root, image=tkimage).pack()
        #return img_zi
        #framed = QGraphicsView()
        #file_path = imread(file_path)
        #height, width, channels = file_path.shape
        #bytesPerLine = channels * width
        #qImg = QImage(file_path.data, width, height, bytesPerLine, QImage.Format_RGB888)
        #pixmap01 = QPixmap.fromImage(qImg)
        #pixmap_image = QPixmap(pixmap01)
        img = Image.open(file_path2)
        w, h = img.size
```

```
        self.imgQ = ImageQt.ImageQt(img)  # we need to hold reference to imgQ, or it will crash
        pixMap = QPixmap.fromImage(self.imgQ)


        scene = QGraphicsScene()
        scene.addPixmap(pixMap)
        #scene.addPixmap(QPixmap(file_path))
        #scene.addPixmap(pixmap_image)
        self.frame_4.setScene(scene)
        self.frame_4.fitInView(QRectF(0, 0, w, h), Qt.KeepAspectRatio)
        self.frame_4.show()
#####

        #####Zoom In#####
    #def fzoomInCheck(self):

        #####Algo PracTice GSCM#####
    def GSCM(self):

        print("Success File Open")
        root = tk.Tk()
        root.withdraw()
        file_path1 = filedialog.askopenfilename()
        print(file_path1)

        print("Success File Open")
        root = tk.Tk()
        root.withdraw()
        file_path2 = filedialog.askopenfilename()
        print(file_path2)

        import numpy as np
        import cv2
        import matplotlib.pyplot as plt

        img1 = cv2.imread(file_path1,0)
        img2 = cv2.imread(file_path2,0)

        orb = cv2.ORB_create()

        kp1, des1 = orb.detectAndCompute(img1,None)
        kp2, des2 = orb.detectAndCompute(img2,None)

        bf = cv2.BFMatcher(cv2.NORM_HAMMING, crossCheck=True)

        matches = bf.match(des1,des2)
        matches = sorted(matches, key = lambda x:x.distance)

        img3 = cv2.drawMatches(img1,kp1,img2,kp2,matches[:10],None, flags=2)

        cv2.imwrite("/home/rahul/Desktop/tmp.jpg",img3)

        self.openWindow()

      ####
    def setupUi(self):
        MainWindow.setObjectName("MainWindow")
        MainWindow.resize(799, 600)
        MainWindow.setMaximumSize(QtCore.QSize(800, 600))
        self.centralwidget = QtWidgets.QWidget(MainWindow)
        self.centralwidget.setObjectName("centralwidget")
        #self.frame = QtWidgets.QFrame(self.centralwidget)
        self.frame = QtWidgets.QGraphicsView(self.centralwidget)
```

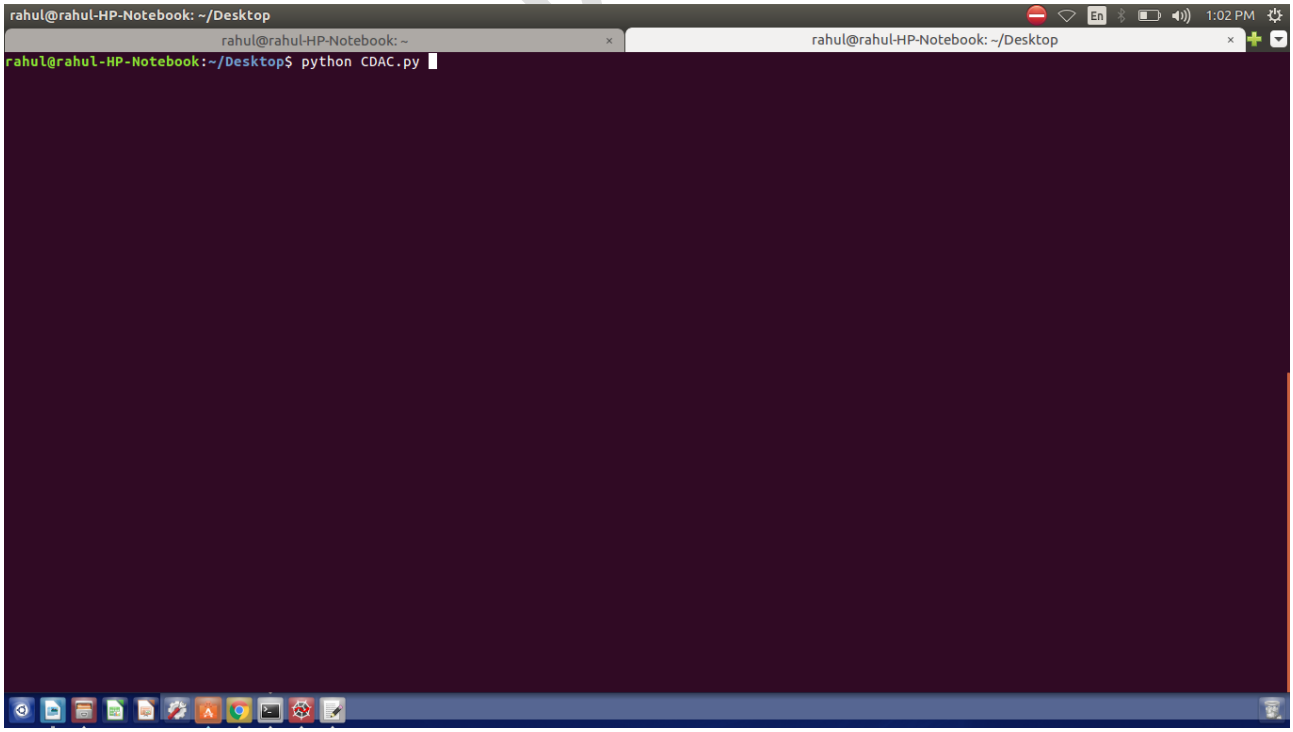
```
self.frame.setGeometry(QtCore.QRect(220, 60, 271, 451))
self.frame.setFrameShape(QtWidgets.QFrame.StyledPanel)
self.frame.setFrameShadow(QtWidgets.QFrame.Raised)
self.frame.setObjectName("frame")
self.pushButton = QtWidgets.QPushButton(self.centralwidget)
self.pushButton.setGeometry(QtCore.QRect(320, 520, 99, 27))
self.pushButton.setObjectName("pushButton")
##OPen Left Connection###
file_path = self.pushButton.clicked.connect(self.fOpenCheck)
###
self.pushButton_2 = QtWidgets.QPushButton(self.centralwidget)
self.pushButton_2.setGeometry(QtCore.QRect(610, 520, 99, 27))
self.pushButton_2.setObjectName("pushButton_2")
####OPen Right Connection###
file_path2 = self.pushButton_2.clicked.connect(self.fOpenCheckR)
###
self.pushButton_3 = QtWidgets.QPushButton(self.centralwidget)
self.pushButton_3.setGeometry(QtCore.QRect(20, 90, 99, 27))
self.pushButton_3.setObjectName("pushButton_3")
###ZOOM IN###
#self.pushButton_3.clicked.connect(self.fzoomInCheck)

self.pushButton_4 = QtWidgets.QPushButton(self.centralwidget)
self.pushButton_4.setGeometry(QtCore.QRect(20, 140, 99, 27))
self.pushButton_4.setObjectName("pushButton_4")
self.pushButton_5 = QtWidgets.QPushButton(self.centralwidget)
self.pushButton_5.setGeometry(QtCore.QRect(20, 190, 99, 27))
self.pushButton_5.setObjectName("pushButton_5")
self.pushButton_6 = QtWidgets.QPushButton(self.centralwidget)
self.pushButton_6.setGeometry(QtCore.QRect(60, 410, 99, 27))
self.pushButton_6.setObjectName("pushButton_6")
#self.frame_3 = QtWidgets.QFrame(self.centralwidget)
self.frame_3 = QtWidgets.QGraphicsView(self.centralwidget)
self.frame_3.setGeometry(QtCore.QRect(30, 460, 171, 80))
self.frame_3.setFrameShape(QtWidgets.QFrame.StyledPanel)
self.frame_3.setFrameShadow(QtWidgets.QFrame.Raised)
self.frame_3.setObjectName("frame_3")
#self.frame_4 = QtWidgets.QFrame(self.centralwidget)
self.frame_4 = QtWidgets.QGraphicsView(self.centralwidget)
self.frame_4.setGeometry(QtCore.QRect(520, 60, 271, 451))
self.frame_4.setMaximumSize(QtCore.QSize(800, 600))
self.frame_4.setFrameShape(QtWidgets.QFrame.StyledPanel)
self.frame_4.setFrameShadow(QtWidgets.QFrame.Raised)
self.frame_4.setObjectName("frame_4")
self.pushButton_7 = QtWidgets.QPushButton(self.centralwidget)
self.pushButton_7.setGeometry(QtCore.QRect(150, 20, 111, 27))
self.pushButton_7.setObjectName("pushButton_7")
self.pushButton_8 = QtWidgets.QPushButton(self.centralwidget)
self.pushButton_8.setGeometry(QtCore.QRect(280, 20, 111, 27))
self.pushButton_8.setObjectName("pushButton_8")
####GSCM COnnection###
self.pushButton_8.clicked.connect(self.GSCM)

self.pushButton_9 = QtWidgets.QPushButton(self.centralwidget)
self.pushButton_9.setGeometry(QtCore.QRect(410, 20, 111, 27))
self.pushButton_9.setObjectName("pushButton_9")
self.pushButton_10 = QtWidgets.QPushButton(self.centralwidget)
self.pushButton_10.setGeometry(QtCore.QRect(530, 20, 111, 27))
self.pushButton_10.setObjectName("pushButton_10")
self.pushButton_11 = QtWidgets.QPushButton(self.centralwidget)
self.pushButton_11.setGeometry(QtCore.QRect(660, 20, 111, 27))
self.pushButton_11.setObjectName("pushButton_11")
```

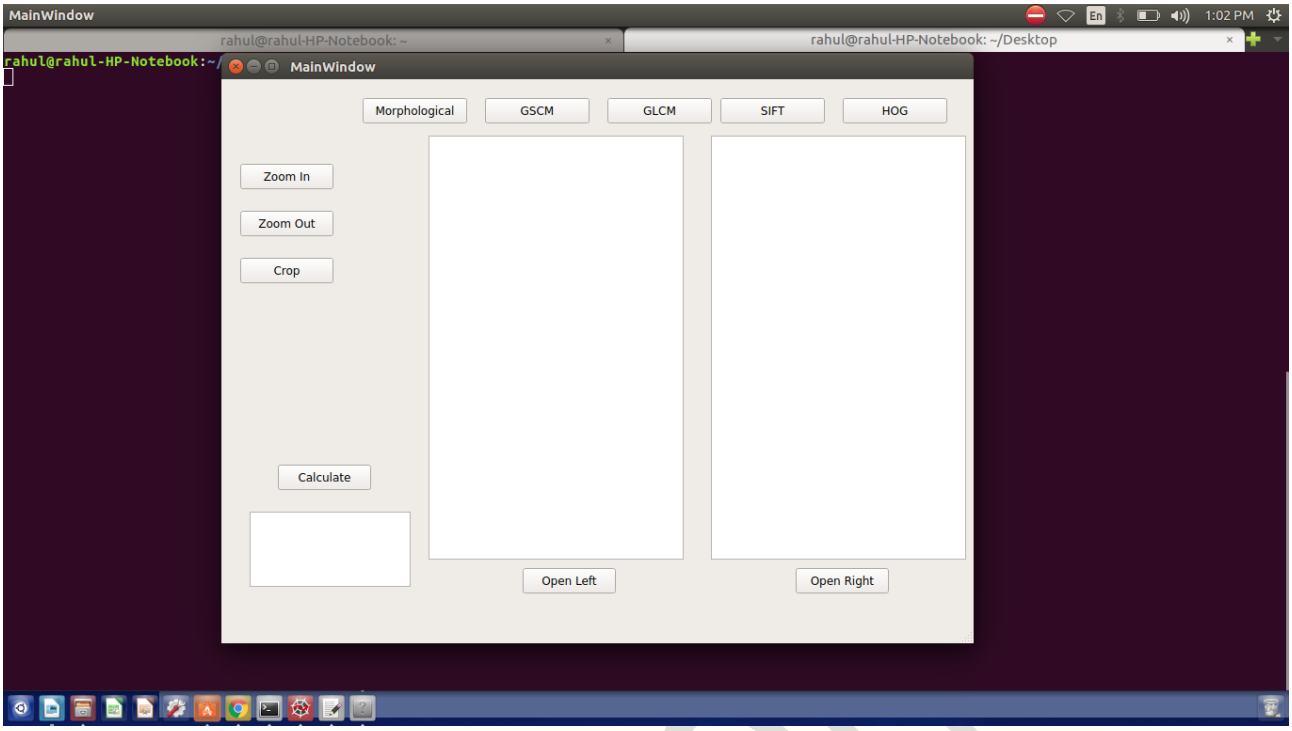
```
        MainWindow.setCentralWidget(self.centralwidget)
        self.menubar = QtWidgets.QMenuBar(MainWindow)
        self.menubar.setGeometry(QtCore.QRect(0, 0, 799, 25))
        self.menubar.setObjectName("menubar")
        MainWindow.setMenuBar(self.menubar)
        self.statusbar = QtWidgets.QStatusBar(MainWindow)
        self.statusbar.setObjectName("statusbar")
        MainWindow.setStatusBar(self.statusbar)

        self.retranslateUi()#(MainWindow)
        QtCore.QMetaObject.connectSlotsByName(MainWindow)

    def retranslateUi(self):
        _translate = QtCore.QCoreApplication.translate
        MainWindow.setWindowTitle(_translate("MainWindow", "MainWindow"))
        self.pushButton.setText(_translate("MainWindow", "Open Left"))
        self.pushButton_2.setText(_translate("MainWindow", "Open Right"))
        self.pushButton_3.setText(_translate("MainWindow", "Zoom In"))
        self.pushButton_4.setText(_translate("MainWindow", "Zoom Out"))
        self.pushButton_5.setText(_translate("MainWindow", "Crop"))
        self.pushButton_6.setText(_translate("MainWindow", "Calculate"))
        self.pushButton_7.setText(_translate("MainWindow", "Morphological"))
        self.pushButton_8.setText(_translate("MainWindow", "GSCM"))
        self.pushButton_9.setText(_translate("MainWindow", "GLCM"))
        self.pushButton_10.setText(_translate("MainWindow", "SIFT"))
        self.pushButton_11.setText(_translate("MainWindow", "HOG"))

    def show(self):
        MainWindow.show()

if __name__ == "__main__":
    import sys
    app = QtWidgets.QApplication(sys.argv)
    MainWindow = QtWidgets.QMainWindow()
    ui = Ui_MainWindow()
    ui.setupUi()
    ui.show()

    sys.exit(app.exec_())
```

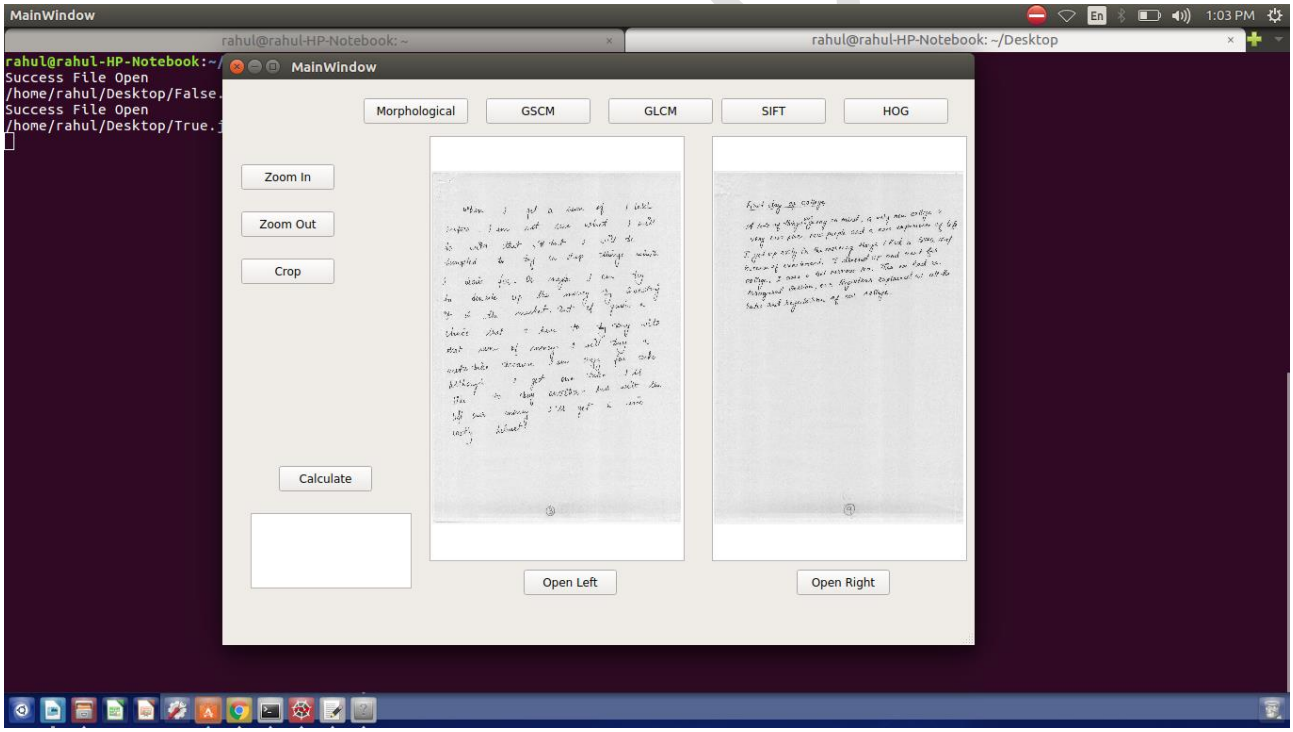
```
from PyQt5 import QtCore, QtGui, QtWidgets
from scipy.ndimage import imread

class Ui_OtherWindow(object):
    def setupUi(self, OtherWindow):
        OtherWindow.setObjectName("OtherWindow")
        OtherWindow.resize(568, 568)
        self.centralwidget = QtWidgets.QWidget(OtherWindow)
        self.centralwidget.setObjectName("centralwidget")
        self.label = QtWidgets.QLabel(self.centralwidget)
        self.label.setGeometry(QtCore.QRect(0, 0, 568, 568))
        font = QtGui.QFont()
        font.setPointSize(22)
        self.label.setFont(font)
        self.label.setObjectName("label")
        OtherWindow.setCentralWidget(self.centralwidget)
        self.statusbar = QtWidgets.QStatusBar(OtherWindow)
        self.statusbar.setObjectName("statusbar")
        OtherWindow.setStatusBar(self.statusbar)

        self.retranslateUi(OtherWindow)
```

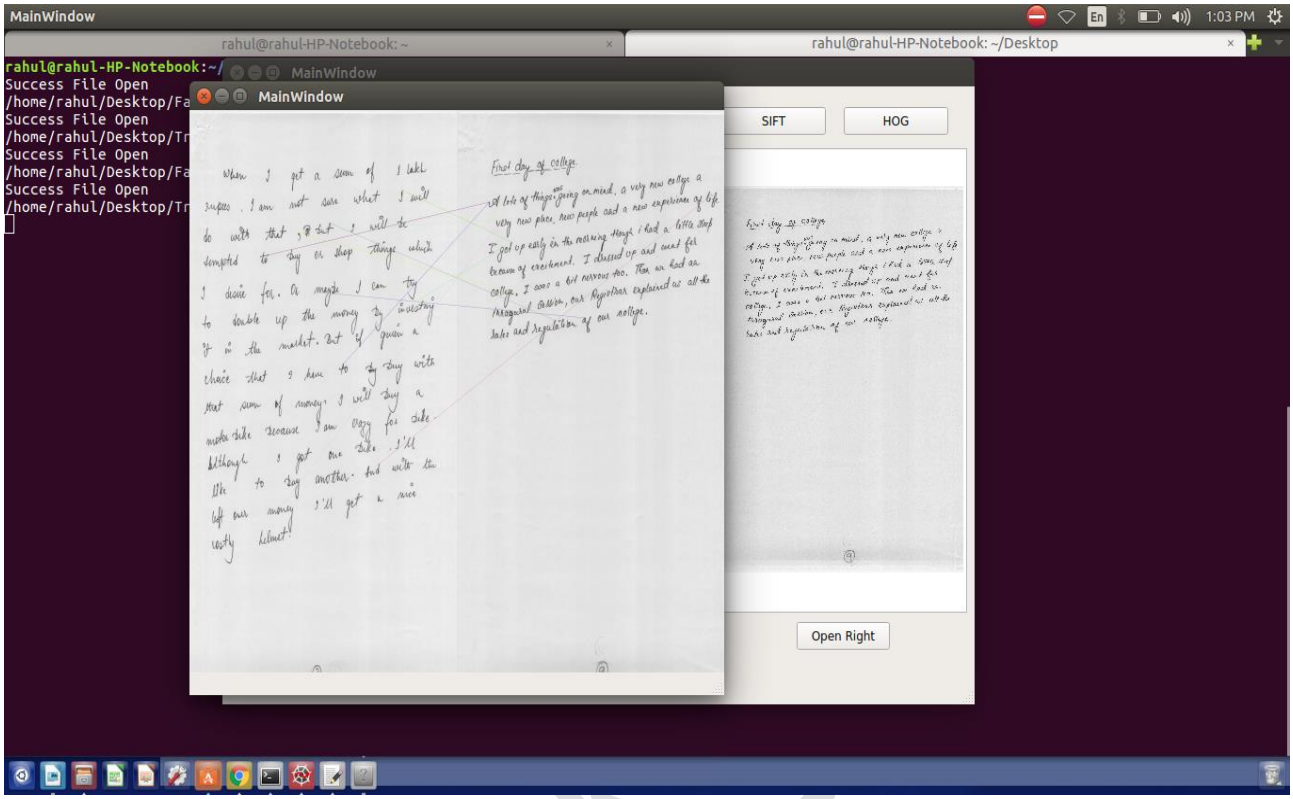
```
        QtCore.QMetaObject.connectSlotsByName(OtherWindow)

    def retranslateUi(self, OtherWindow):
        _translate = QtCore.QCoreApplication.translate
        OtherWindow.setWindowTitle(_translate("OtherWindow", "MainWindow"))
        input_image = imread("/home/rahul/Desktop/tmp.jpg")
        height, width, channels = input_image.shape
        bytesPerLine = channels * width
        qImg = QtGui.QImage(input_image.data, width, height, bytesPerLine, QtGui.QImage.Format_RGB888)
        pixmap01 = QtGui.QPixmap.fromImage(qImg)
        pixmap_image = QtGui.QPixmap(pixmap01)
        self.label.setText(_translate("OtherWindow", "Welcome To This Window"))
        self.label.setPixmap(pixmap_image)
        self.label.setAlignment(QtCore.Qt.AlignCenter)
        self.label.setScaledContents(True)
        self.label.setMinimumSize(568,568)
        self.label.show()

if __name__ == "__main__":
    import sys
    app = QtWidgets.QApplication(sys.argv)
    OtherWindow = QtWidgets.QMainWindow()
    ui = Ui_OtherWindow()
    ui.setupUi(OtherWindow)
    OtherWindow.show()
    sys.exit(app.exec_())
```

Output:




MainWindow
1:02 PM
rahul@rahul-HP-Notebook: ~
rahul@rahul-HP-Notebook: ~/Desktop
rahul@rahul-HP-Notebook:~/
MainWindow
Morphological
GSCM
GLCM
SIFT
HOG
Zoom In
Zoom Out
Crop
Calculate
Open Left
Open Right


MainWindow
1:03 PM
rahul@rahul-HP-Notebook: ~
rahul@rahul-HP-Notebook: ~/Desktop
rahul@rahul-HP-Notebook:~/
Success File Open
/home/rahul/Desktop/False.
Success File Open
/home/rahul/Desktop/True.j
MainWindow
Morphological
GSCM
GLCM
SIFT
HOG
Zoom In
Zoom Out
Crop
Calculate
Open Left
Open Right


MainWindow
1:03 PM
rahul@rahul-HP-Notebook: ~
rahul@rahul-HP-Notebook: ~/Desktop
rahul@rahul-HP-Notebook:~/
Success File Open
/home/rahul/Desktop/Fa
Success File Open
/home/rahul/Desktop/Tr
Success File Open
/home/rahul/Desktop/Fa
Success File Open
/home/rahul/Desktop/Tr
MainWindow
MainWindow
SIFT
HOG
Open Right

6 CHAPTER : LITERATURE REVIEW

6.1 THEORY, DATA ANALYSIS AND FINDINGS

6.1.1 Mean square error (MSE) & Structural Similarity Measure (SSIM):

The Mean Squared Error (MSE) is a measure of how close a fitted line is to data points. For every data point, we take the distance vertically from the point to the corresponding y value on the curve fit (the error), and square the value. Then we add up all those values for all data points, and, in the case of a fit with two parameters such as a linear fit, divide by the number of points minus two.

The squaring is done so negative values do not cancel positive values. The smaller the Mean Squared Error, the closer the fit is to the data. The MSE has the units squared of whatever is plotted on the vertical axis.

MSE is very simple to implement — but while using it for similarity, we can run into problems. The main one being that large distances between pixel intensities do not necessarily mean the contents of the images are dramatically different.

It's important to note that a value of 0 for MSE indicates perfect similarity. A value greater than one implies less similarity and will continue to grow as the average difference between pixel intensities increases as well.

$$MSE = \frac{1}{m\,n}\sum_{i=0}^{m-1}\sum_{j=0}^{n-1}[I(i,j) - K(i,j)]^2$$

The structural similarity (SSIM) index is a method for predicting the perceived quality of digital television and cinematic pictures, as well as other kinds of digital images and videos. SSIM is used for measuring the similarity between two images. The SSIM index is a full reference metric; in other words, the measurement or prediction of image quality is based on an initial uncompressed or distortion-free image as reference.

In recent years the structural similarity index has become an accepted standard among image quality metrics. Made up of three components, this technique assesses the visual impact of changes in image luminance, contrast, and structure. Applications of the index include image enhancement, video quality monitoring, and image encoding. As its status continues to rise, however, so do questions about its performance, it is shown, both empirically and analytically, that the index is directly related to the conventional, and often unreliable, mean squared error. In the first evaluation, the two metrics are statistically compared with one another. Then, a pair of functions that algebraically connects the two is derived. These suggest a much closer relationship between the structural similarity index and mean squared error.

SSIM is a perception-based model that considers image degradation as perceived change in structural information, while also incorporating important perceptual phenomena, including both luminance masking and contrast masking terms. Structural information is the idea that the pixels have strong inter-dependencies especially when they are spatially close. These dependencies carry important information about the structure of the objects in the visual scene. Luminance masking is a

phenomenon whereby image distortions (in this context) tend to be less visible in bright regions, while contrast masking is a phenomenon whereby distortions become less visible where there is significant activity or "texture" in the image.

$$\mathrm{SSIM}(x,y) = \frac{(2\mu_x\mu_y + c_1)(2\sigma_{xy} + c_2)}{(\mu_x^2 + \mu_y^2 + c_1)(\sigma_x^2 + \sigma_y^2 + c_2)}$$

6.1.2 Moment invarient structural measurement(mism)

The proposed approach here is very well understood as the approximation level of Discrete Wavelet Decomposition of an image results in revealing the structure of the images. The approximation levels remove detail successively and leave the structure intact even at deeper decomposition levels. At each successive level, structure of an image is maintained while removing the texture and detail. Once the image is reduced to an acceptable level, edge detection can be used to further sharpen the structure of the image. If a metric is produced using this structural information, it will truly capture the structural information and will be a valid measure to evaluate the structural integrity thereby making comparing images more meaningful. Moment Invariants have been used extensively in identifying shapes or outlay of objects for many years .

MISM calculation is outlined. An image is normalized (divided by its own standard deviation) such that the two images being compared have unit standard deviation. An image reduced to an approximation level (usually larger than 16x16) and then edge detected using 'Canny' operator and first moment invariant (ϕ 1) is calculated for the entire approximation . Then the approximation level is divided into four quadrants and the first and second moments (ϕ i1, ϕ i2) are calculated for each quadrant.

Here ϕ ' indicates the moment invariants of the second image. Computing image difference to check difference between two image.

An image difference is that accurately predicts human judgments is the Holy Grail of perception-based image processing. In computing image difference takes two images and parameters that specify the viewing conditions (e.g., viewing distance, illuminate, and luminance level). It returns a prediction of the perceived difference between the images under the specified viewing conditions. An accurate Image difference could supersede tedious psychophysical experiments that are required to optimize imaging systems and image processing.

It is improbable that Image difference will perfectly predict human perception before the cortical visual processing is comprehensively understood. However, Image difference could provide a reasonable median prediction of human judgments for only a few selected distortions, e.g., glossy compression or gamut mapping.

Many Image difference measurement create image-difference maps showing perceived pixel deviations between two input images. For image-difference evaluation, these maps are transformed into a single characteristic value, such as the mean or the 95th percentile. However, psychophysical experiments show that the degree of difference visibility is not well correlated with perceived overall image difference . For example, global intensity changes are generally less objectionable than compression artifacts . It is therefore likely that the prediction performance of Image difference measurment that only operate on image-difference maps can be improved.

In summary, we obtain image-difference measures by:

Feature extraction:

Computing a large number of IDFs for a set of training images.

Sorting:

Selecting the most important IDFs considering redundancies and prediction performance of individual IDFs.

Blending:

Optimizing the parameters of the selected blending model on the training images.

Example formula:

The difference is defined by the average % difference between each of the channels (R,G,B,A?) at each pair of pixels Axy, Bxy at the same coordinates in each of the two images (why they need to be the same size), averaged over all pairs of pixels.

For example, compare two 1x1 images A and B (a trivial example, >1 pixels would have another step to find the average of all pixels):

A1,1 = RGB(255,0,0) (pure red)

B1,1 = RGB(100,0,0) (dark red)

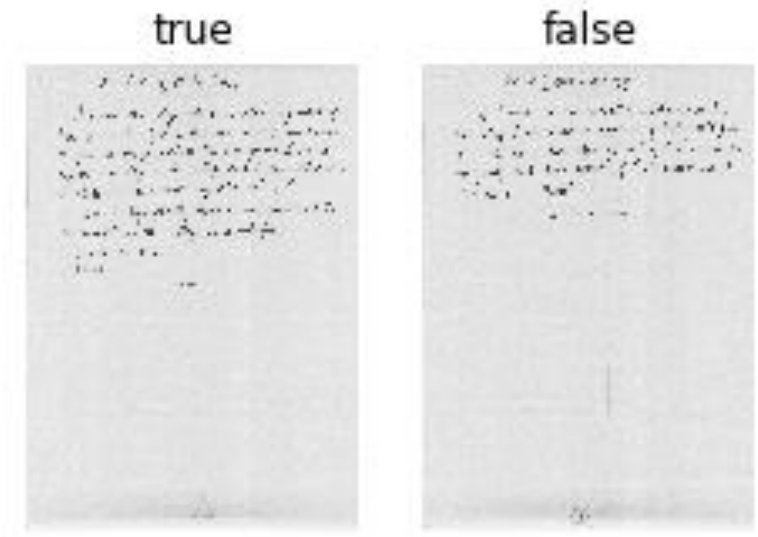
((255-100)/255 + (0/0)/255 + (0/0)/255))/3 = (155/255)/3 = 0.202614379

Code:

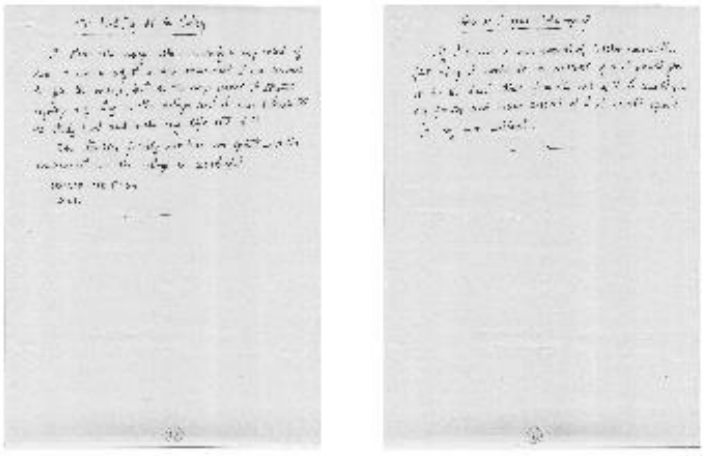
```
from skimage.measure import structural_similarity as ssim
import matplotlib.pyplot as plt
import numpy as np
import cv2
def mse(imageA, imageB):
    err = np.sum((imageA.astype("float") - imageB.astype("float")) ** 2)
    err /= float(imageA.shape[0] * imageA.shape[1])
    return err
def compare_images(imageA, imageB, title):
   m = mse(imageA, imageB)
   s = ssim(imageA, imageB)
  fig = plt.figure(title)
  plt.suptitle("MSE: %.2f, SSIM: %.2f" % (m, s))

  ax = fig.add_subplot(1, 2, 1)
 plt.imshow(imageA, cmap = plt.cm.gray)
 plt.axis("off")
 ax = fig.add_subplot(1, 2, 2)
 plt.imshow(imageB, cmap = plt.cm.gray)
 plt.axis("off")
 plt.show()
```

```
true = cv2.imread("//home/rahul/Documents/project/Scan_Guwahati_Handwriting_data25.8.14/WanjoplangWansai-13/True.jpg")
false =

cv2.imread("//home/rahul/Documents/project/Scan_Guwahati_Handwriting_data25.8.14/WanjoplangWansai-13/False.jpg")

true = cv2.cvtColor(true, cv2.COLOR_BGR2GRAY)

false = cv2.cvtColor(false, cv2.COLOR_BGR2GRAY)

fig = plt.figure("Images")
images = ("true", true), ("false", false)

for (i, (name, image)) in enumerate(images):
    ax = fig.add_subplot(1, 3, i + 1)
    ax.set_title(name)
    plt.imshow(image, cmap = plt.cm.gray)
    plt.axis("off")

plt.show()

compare_images(true, true, "true vs. true")
compare_images(true, false, "true vs. false")
```

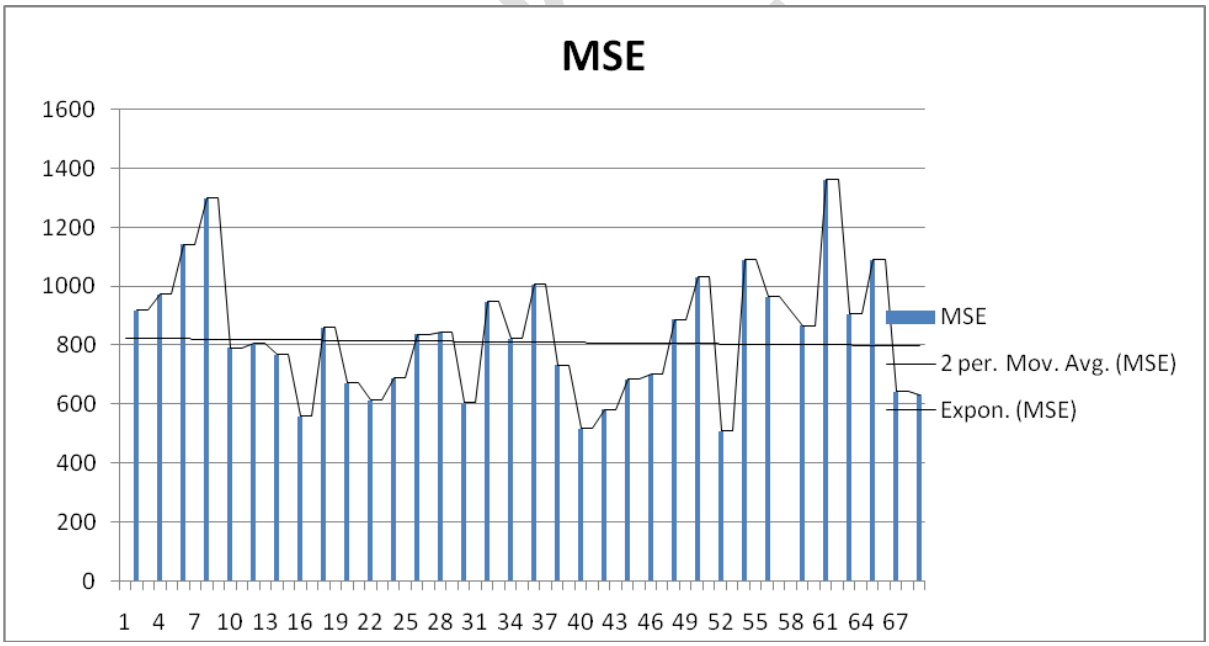
Output:



MSE: 633.95, SSIM: 0.72

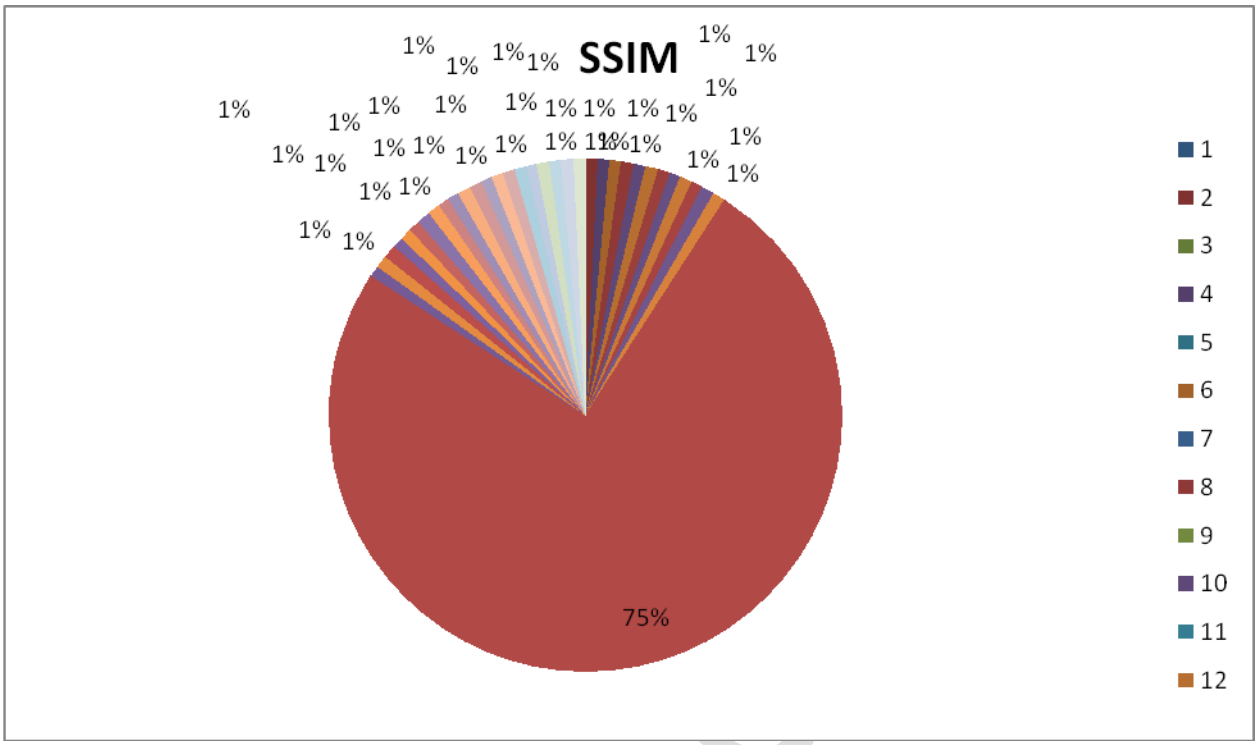


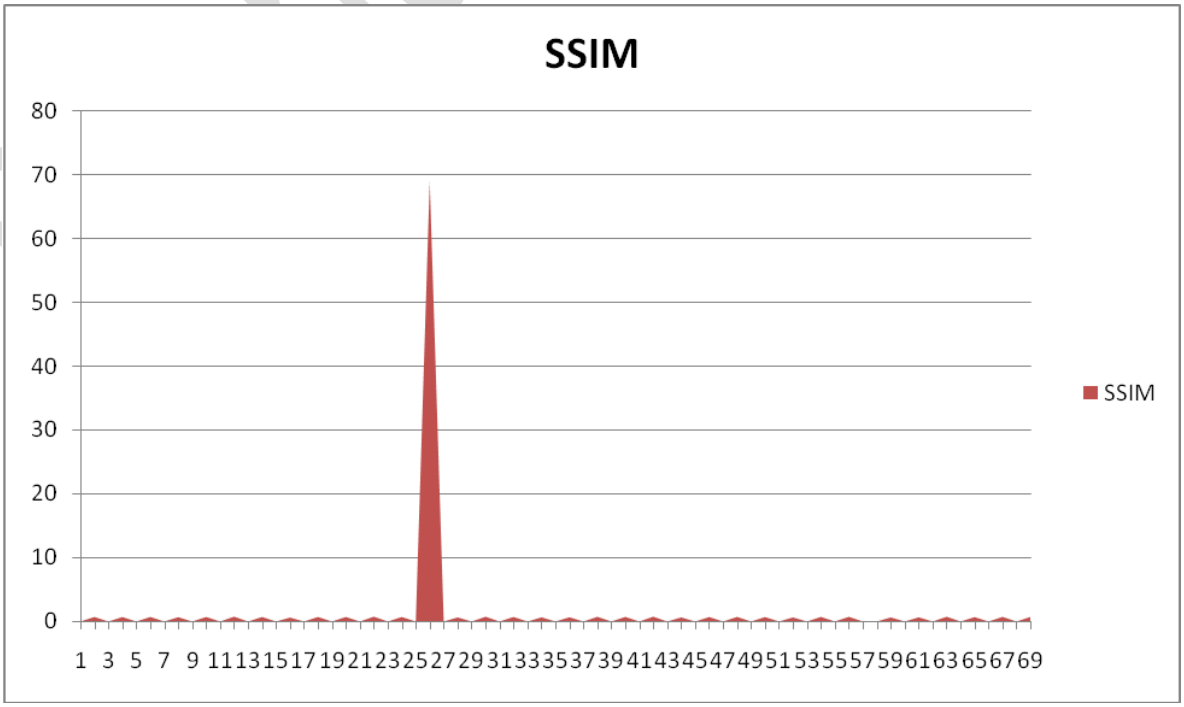
Graph:



SSIM:

Graph:


SSIM
1%
75%
1
2
3
4
5
6
7
8
9
10
11
12


SSIM
80
70
60
50
40
30
20
10
0
1 3 5 7 9 11 13 15 17 19 21 23 25 27 29 31 33 35 37 39 41 43 45 47 49 51 53 55 57 59 61 63 65 67 69
SSIM

6.1.3 Geometric Transformations of Images:

Perspective Techniques provides this flexible-viewpoint photography through a functional merging of photogrammetry, computer graphics, 3-D modeling and image processing technologies. Photography is first collected and digitized and then calibrated using photogrammetric techniques. This characteristics has the unique characteristic that it allows the construction of three-dimensional objects directly on the source photographs. No blue-prints or other drawings are required. The shape, orientation, position, and size of the modeled object is then stored in the data base along with the visual components of each of its surfaces.

For perspective transformation, we need a 3x3 transformation matrix. Straight lines will remain straight even after the transformation. To find this transformation matrix, we need 4 points on the input image and corresponding points on the output image. Among these 4 points, 3 of them should not be collinear. Then transformation matrix can be found by the function cv2.getPerspectiveTransform. Then apply cv2.warpPerspective with this 3x3 transformation matrix.

Calculates a perspective transform from four pairs of the corresponding points.

Python: cv2.getPerspectiveTransform(src, dst) → retval

Python: cv.GetPerspectiveTransform(src, dst, mapMatrix) → None

Parameters:
- src – Coordinates of quadrangle vertices in the source image.
- dst – Coordinates of the corresponding quadrangle vertices in the destination image.

The function calculates the 3×3 matrix of a perspective transform so that:

$$\begin{bmatrix} t_i x'_i \\ t_i y'_i \\ t_i \end{bmatrix} = \texttt{map\_matrix} \cdot \begin{bmatrix} x_i \\ y_i \\ 1 \end{bmatrix}$$

where

$$\texttt{dst}(i) = (x'_i, y'_i), \texttt{src}(i) = (x_i, y_i), i = 0, 1, 2, 3$$

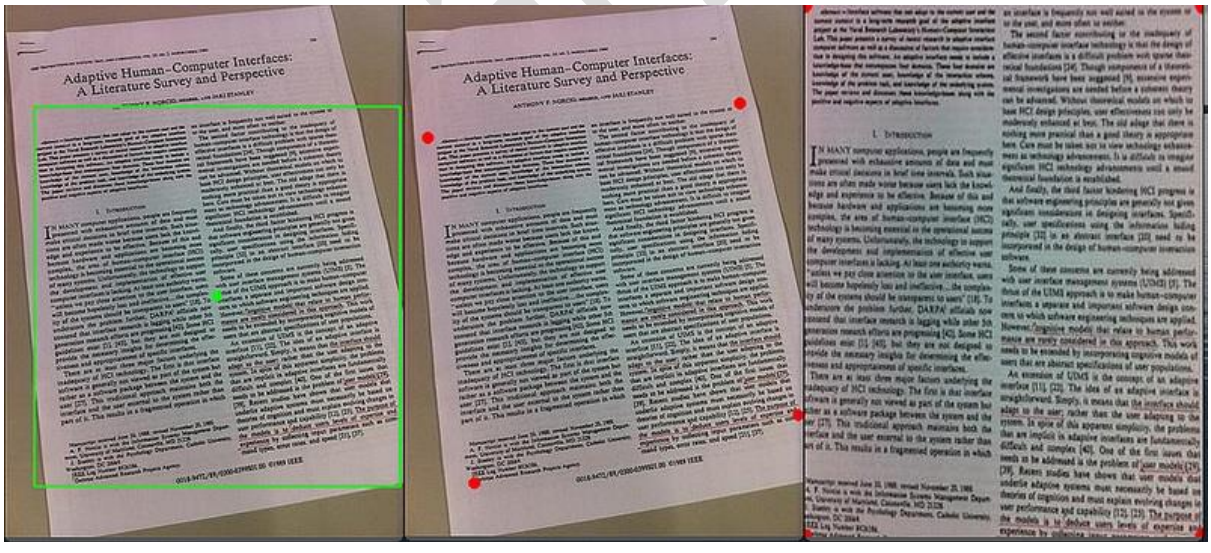
Code and analysis:

```
import cv2
import numpy as np
c = cv2.imread('/home/rahul/Desktop/False.jpg')
height, width = c.shape[:2]
cv2.namedWindow('Frame', cv2.WINDOW_NORMAL)
```

```
cv2.resizeWindow('Frame', 500, 500)
cv2.circle(c, (150,110), 15, (0,0,255), -10)
cv2.circle(c, (2400,110), 15, (0,0,255), -10)
cv2.circle(c, (150,1800), 15, (0,0,255), -10)
cv2.circle(c, (2400,1800), 15, (0,0,255), -10)

pts1 = np.float32([[150,110],[2400,110],[150,1800],[2400,1800]])

pts2 = np.float32([[0,0],[2300,0],[0,1900],[2300,1900]])

matrix = cv2.getPerspectiveTransform(pts1,pts2)


results = cv2.warpPerspective(c,matrix,(2300,1900))

cv2.imshow('New',results)

cv2.imshow('Frame', c)
r = cv2.waitKey(0)
cv2.destroyAllWindows()
```

Analysis:

Suppose, some of the selected images have not perpendicular views and it is one of the greatest problems are faced by us. The samples are given below,



Thus, we have to get the perpendicular view of those images to understand the images in more convenient approach. To scale the images and converts them into a fixed perpendicular structure, this perspective transformation technique is applied.
At the beginning, we identify the exact required area of the images and specify the boundary of that area with “red dots”. After applying the dots at boundary, we cropped out the exact required area from the images and it has the “Bird-View”.

6.1.4 Calculating Entropy of an Image:

The entropy of a system as defined by Shannon, gives a measure of uncertainty about its actual structure. Shannon's function is based on the concept that the information gain from an event is inversely related to its probability of occurrence. Many have used Shannon's concept for image processing and pattern recognition problems. Here we used Shannon's concept to define the entropy of an image assuming that an image is entirely represented by its gray level histogram only. As a result segmentation algorithms using Shannon's function resulted in an unappealing result, viz, same entropy and threshold values for different images with identical histogram. The present work first of all, introduces a new definition of classical entropy along with its justification. Based on the new concept, three definitions (e.g., global, local and conditional) of entropy of an image are then introduced and applied to formulate four algorithms for image segmentation.

One of the algorithms assumes a Poisson distribution to describe the gray level variation within the object and background.

The entropy or average information of an image can be determined approximately from the histogram of the image. The histogram shows the different grey level probabilities in the image. The entropy is useful, for example, for automatic image focusing: as the state of focusing of an image varies, so does its entropy.

6.1.5 Shannon's entropy and its applications

Shannon defined the entropy of an n-state system as,

$$H = -\sum_{i=1}^{n} ptlog\ (pi)$$

where p, is the probability of occurrence of the event i and

$$\sum_{i=1}^{n} pi = 1 \qquad\qquad 0 \leq pi \leq 1$$

Intuitively we feel that the gain in information from an event is inversely related to its probability of occurrence.

In the case of an image, the states correspond to the gray levels which the individual pixels can adopt. For example, in an 8-bit pixel there are 256 such states. If all such states are equally occupied, as they are in the case of an image which has been perfectly histogram equalized, the spread of states is a maximum, as is the entropy of the image. On the other hand, if the image has been thresholded, so that only two states are occupied, the entropy is low. If all of the pixels have the same value, the entropy of the image is zero.

The entropy of the image is decreased, so is its information content. We moved from a full gray scale image, with high entropy, to a thresholded binary image, with low entropy, to a single-valued image, with zero entropy. If the pixels of an image were inspected, and found to be the same. This information could have been communicated in a very short message. The information content is said to be low simply because it can be communicated in a short message. If the pixels are changing in unexpected ways, however, longer messages are required to communicate this fact and the information is said to increase. This assumes, of course, that all changes in the image are meaningful. Changes due to noise are still considered to be information in that they describe the image as it actually is, rather than as it should be. Entropy sets a lower bound on the average number of bits per pixel required to encode an image without distortion. This is only true, however, for uncorrelated images. Consider the images below:

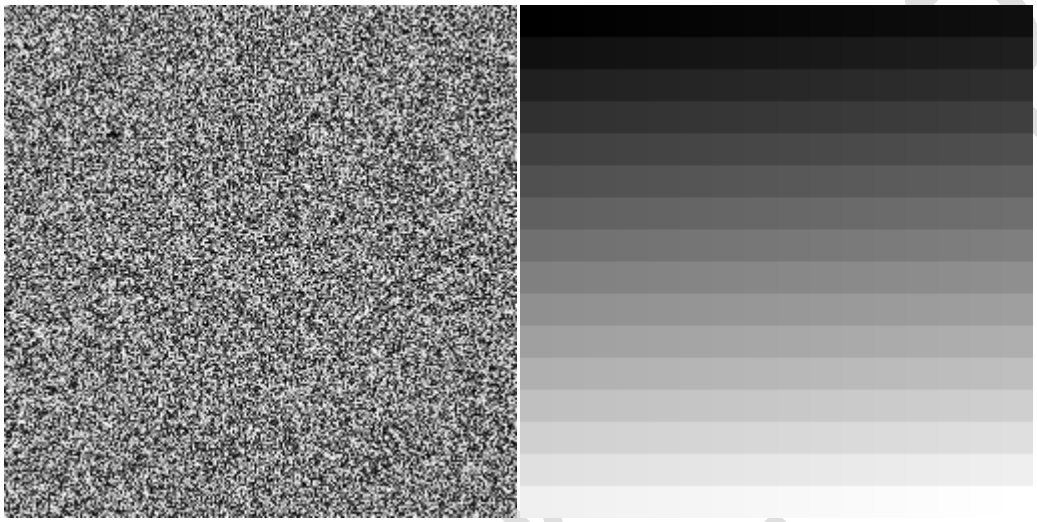


Fig: The image on the left (uniform random noise) has an entropy of 8 bits and is uncompressible (look at the GIF file size). The image on the right has the same distribution of gray levels but is highly spatially correlated.

The entropy or average information of an image can be determined approximately from the histogram of the image. The histogram shows the different grey level probabilities in the image. The entropy is useful, for example, for automatic image focusing: as the state of focusing of an image varies, so does its entropy. We present a method for measuring the entropy quite quickly and with reasonable accuracy. Our method is fast for two reasons: we have derived empirically a simple approximation formula for the entropy of images.

There are a wide variety of features to describe the texture of an image, for example local binary patterns, Gabor filters, wavelets, Laws' masks and many others. GLCM is one of the most popular texture descriptors. One possible approach to describe the texture of an image through GLCM features consists in computing the GLCM for different offsets (each offset is defined through a distance and an angle), and extracting different properties from each GLCM.

Let us consider for example three distances (1, 2 and 3 pixels), four angles (0, 45, 90 and 135 degrees) and two properties (energy and homogeneity). This results in $3 \times 4 = 12$ offsets (and hence 12 GLCM's) and a feature vector of dimension $12 \times 2 = 24$. Here's the code:

Code:

```
import numpy as np
from skimage import io, color, img_as_ubyte
from skimage.feature import greycomatrix, greycoprops
```

```
from sklearn.metrics.cluster import entropy

rgbImg = io.imread('/home/rahul/Documents/project/Scan_Guwahati_Handwriting_data25.8.14/WanjoplangWansai-13/True.jpg')
grayImg = img_as_ubyte(color.rgb2gray(rgbImg))

distances = [1, 2, 3]
angles = [0, np.pi/4, np.pi/2, 3*np.pi/4]
properties = ['energy', 'homogeneity']

glcm = greycomatrix(grayImg,
distances=distances,
angles=angles,
symmetric=True,
normed=True)

feats = np.hstack([greycoprops(glcm, prop).ravel() for prop in properties])
print(entropy(grayImg))
```

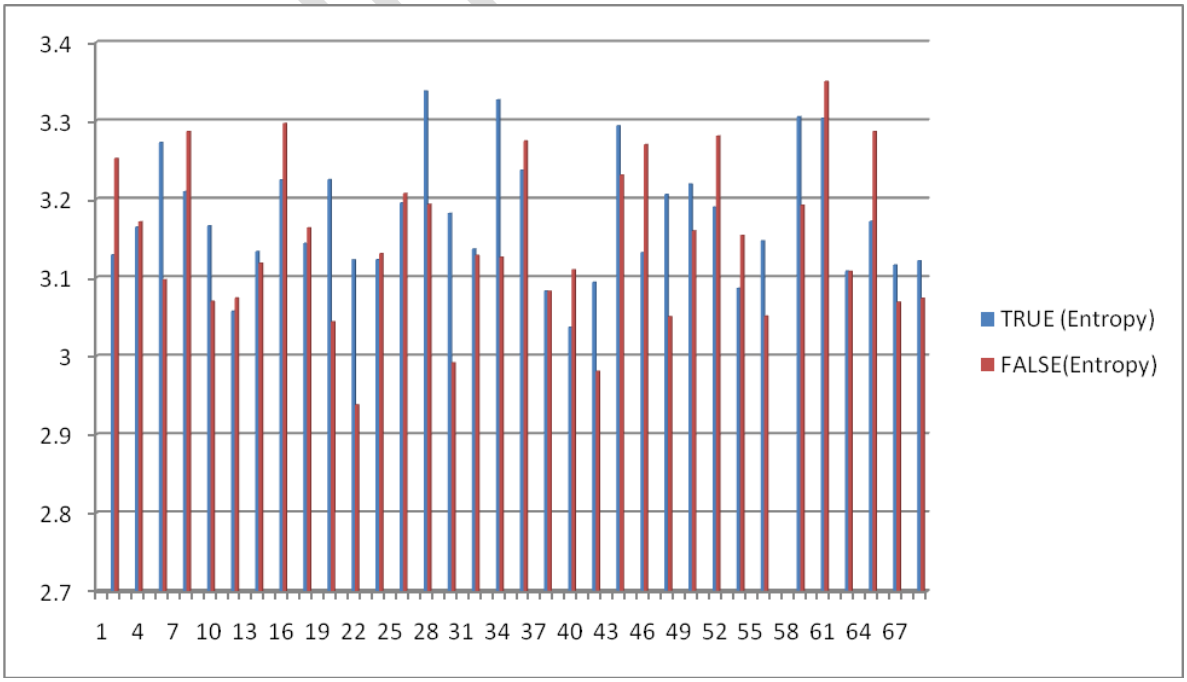
Output:

File Name	Entropy (True)	Entropy (False)
AbhilashSingh-9	3.128921577	3.251965245
AnmolKerketta	3.164346646	3.171049168
ArindamRoy-6	3.272300325	3.097250366
BanashreePaul-32	3.209583044	3.286320705
BethshebdaJRaththap-31	3.165854233	3.069598358
D.JeromeSudeep-16	3.057095906	3.073882929
DarvilHamarbabiangC.Sohliva-2	3.133129024	3.118369714
DebashisBurman-12	3.22466718	3.296430427
DipayanDas-8	3.143757672	3.163364063

DiskshantPatodia-23	3.224897101	3.04357762
FrancisNcarzary-30	3.122414967	2.937669724
GauravChaubey-29	3.122996872	3.130459748
HemantaAlley-10	3.195220922	3.2074157
HimanshuChoudhury-21	3.337850212	3.193397379
IawalinNongbri-28	3.181840178	2.991061448
IraniMahanta-27	3.136298184	3.128442261
KynsaiStephanKhdnsam-18	3.326429129	3.125895854
NamrataDas-25	3.236926422	3.274372862
NitunjitBrahma-1	3.082872248	3.0826087
ParthaPratimSinha-17	3.036653527	3.109983217
PriyankBaruah-24	3.093932749	2.980470556
PriyanshuThakuria-22	3.293438259	3.230896451
PujaMedhi-34	3.131561602	3.269522575
Pyndapbiang John Lyngdoh-20	3.205931677	3.050261065
RinkyGurumayum-33	3.219301655	3.159286363

SamujjalSarmah-5	3.189804393	3.280415609
SharmilaGayari-26	3.0863742	3.153629379
ShijoGeorge-11	3.146886955	3.050971962
SiddarthaSauravGogoi	3.305031384	3.1923481
SurajKumarSarmah-14	3.303237169	3.350044498
TapasDas-15	3.108156768	3.10776856
ThinggujamJoseph-4	3.171669631	3.286219056
WangmaoS.Konyak-19	3.115864005	3.068508715
WanjoplangWansai-13	3.12154267	3.073474188

Graph:



Analysis:

From the above graph, it is noticed that, most of the cases, entropy of the true images are much more higher than false images. And, some of the cases, the entropy of the True and False images are almost equal. And, there are some types of samples are acted in some different ways where, the value of false images are higher than true images. So, from the graph, we can predict that the higher entropy of the samples are included in the true part and others are included into false samples.

6.1.6 Pen Pressure measurement:

WRITER verification is a method used to specify an authentic writer from handwriting. The need for automated writer verification methods has increased in various applications (e.g., credit cards, checks, and passports). Writer verification plays an important role in biometrics and forensics. In biometrics, writer verification identifies a writer from behavioural characters automated writer verification assists forensic document examiner (FDE) investigations . Writer verification methods can be categorized into two types, online and offline methods. In online methods, dynamic information (e.g., pen location, pen inclination angles, and pen pressure) is used for analysis. In offline methods, only static information from an image of written characters is used.

Offline methods are very challenging because there is less information available than online methods. Todate,many methods have been proposed . However, there is room for improvement in the performance of offline writer verification methods compared with the performance of human beings, especially FDEs.

One approach is the use of pen pressure information. In forensic science, FDEs visually, microscopically, and instrumentally examine dynamic pen movements from static handwritten documents . In online writer verification, which is more accurate than offline verification, pen pressure, which can improve verification accuracy, is one of the important features among the available dynamic information . The shape of characters can be easily imitated by forgers; however, pen pressure is difficult to forge, and discrepancies in pen pressure are used to detect forged handwriting .

From a visible image, the pen pressure information based on ink intensity is extracted as local directional pattern (LDP) features, whereas, from the IR image, the pen pressure information based on indentations on a paper is extracted as the first- and second-order texture statistics . We have shown that this combined use of the pen pressure information can improve the accuracy of writer verification.

To improve the performance of automated writer verification methods, we have proposed methods to analyse pen pressure from static handwriting . The studies

showed the effectiveness of analyzing the pen pressure information through writing indentations. We alsoshowed the effectiveness of analyzing the pen pressure information through ink intensity from visible images, in addition to analyzing writing indentations .

To represent the pen pressure information through ink intensity from the visible images, we use the LDP as the basic representation. The LDP is an 8-bit binary code assigned to eachpixeloftheinputgray-levelimage andisdescribedas an effective pseudodynamic feature for writer

verification . It is calculated by comparing the relative edge response values of the pixel in different directions.

The LDP is calculated as follows:

LDP = ∑_(p=0)^7sp(|mp|−|mk|)•2p

To extract the penpressure information from writing indentations on paper from the IR images we use the first-and second order texture statistics

. 1) First-OrderStatistics: Thefirst-orderstatisticsdependon the histogram of each pixel value. In our study, the statistics are calculated as follows:

Mean μ = ∑_(i=0)^(L-1) 〖ip(i)〗

Variance σ = ∑_(i=0)^(L-1) 〖(i-u)^2p(i)〗

The second-order statistics are calculated based on the gray-level cooccurren cematrix (GLCM) . The concept of the GLCM is the analysis of the joint probability distribution of the gray-level i and j within a spatial relation of the image. Here, the spatial relation is defined with the distance d and angle θ.

The GLCM is represented by a G×G matrix, where G is the number of rows and columns equal to the maximum gray-level in the image. If the image has an 8-bit gray-level, the GLCM is represented by a 256×256 matrix with many zeroes in the matrix. To reduce the calculation cost due to the sparse GLCM matrix, the image is quantized by the maximum gray-level from 256 to 8.

6.1.7 Converting Image to Gray Scale:

Mean value	$\overline{x} = \frac{1}{N}\sum_{i=1}^{N} x_i$
Standard deviation	$\sigma = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(x_i - \overline{x})^2}$
Kurtosis	$\mathrm{K} = \frac{1}{N}\sum_{i=1}^{N} \frac{(x_i-\overline{x})^4}{\sigma^4}$
Skewness	$\mathrm{Sk} = \frac{1}{N}\sum_{i=1}^{N} \frac{(x_i-\overline{x})^3}{\sigma^3}$

The selected JPEG images are comprised of three dimensional arrays of the shape M x N x 3 with the final dimension comprised of RGB values. By playing with the RGB values we can change the colors of the image. They also have a few lines of code which showed how playing with the numbers increased or decreased the contrast.

This was a light bulb moment. I decided why not use my newfound skills to make the convert an image to grayscale. Googling the query landed me on the following page which told me that in there software they use three algorithms to convert the image to grayscale.

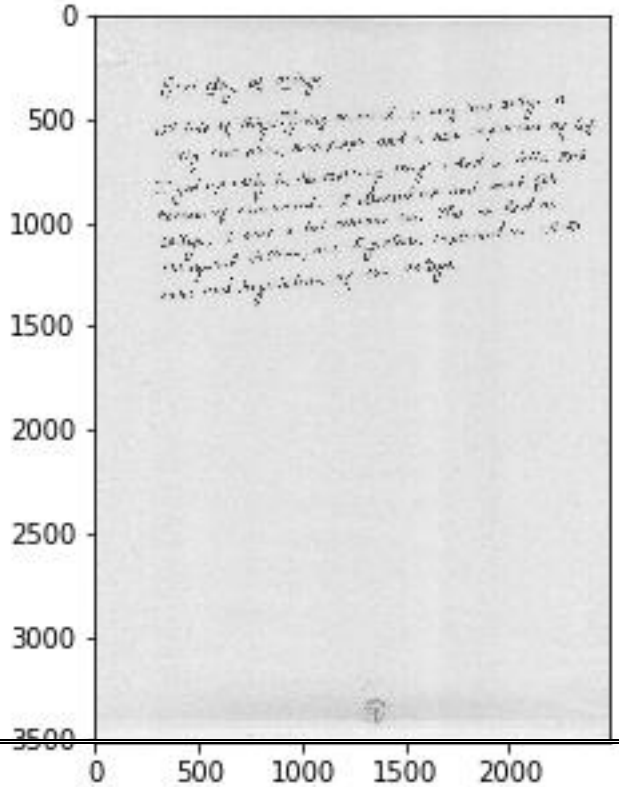
The lightness method averages the most prominent and least prominent colors: (max(R, G, B) + min(R, G, B)) / 2.The average method simply averages the values: (R + G + B) / 3.

The luminosity method is a more sophisticated version of the average method. It also averages the values, but it forms a weighted average to account for human perception. We're more sensitive to green than other colors, so green is weighted most heavily. The formula for luminosity is 0.21 R + 0.72 G + 0.07 B.

Code and analysis:

```
import numpy as np
from scipy import misc
import matplotlib.pyplot as plt
from skimage import data
photo_data = misc.imread("/home/rahul/Desktop/True.jpg")
x,y,z=photo_data.shape ## where z is the RGB dimension
### Method block begin
photo_data[:] = photo_data.mean(axis=-1,keepdims=1)
### Method Block ends
plt.figure(figsize=(5,5))

plt.imshow(photo_data)

img=photo_data
print(img)
```

Output:



Analysis:

This is the output of true images and it is converted into gray scale image.
In grayscale images, the watershed algorithm is fairly easy to conceptualize because we can think of the two spatial dimensions and one brightness dimension. For learning image processing, it's better to understand grayscale processing first and understand how it applies to multichannel processing rather than starting with full color imaging and missing all the important insights that can (and should) be learned from single channel processing.

6.1.8 Thresholding:

Thresholding is a process of converting a grayscale input image to a bi-level image by using an optimal threshold.

The purpose of thresholding is to extract those pixels from some image which represent an object (either text or other line image data such as graphs, maps). Though the information is binary the pixels represent a range of intensities. Thus the objective of binarization is to mark pixels that belong to true foreground regions with a single intensity and background regions with different intensities.

Image thresholding is a simple, yet effective, way of partitioning an image into a foreground and background. This image analysis technique is a type of image segmentation that isolates objects by converting grayscale images into binary images. Image thresholding is most effective in images with high levels of contrast. Thresholding is a non-linear operation that converts a gray-scale image into a binary image where the two levels are assigned to pixels that are below or above the specified threshold value.

One of the most used techniques for the analysis of the images is that of the thresholding, ie the application of a threshold along a particular scale of values, to filter in some way an image.

One of these techniques is for example the one that converts any image in grayscale (or color) in a totally black and white image. Often this is very useful for recognizing the regular shapes, contours

within an image, or even to delimit and divide zones inside, to then be used in a different way in the subsequent processing.

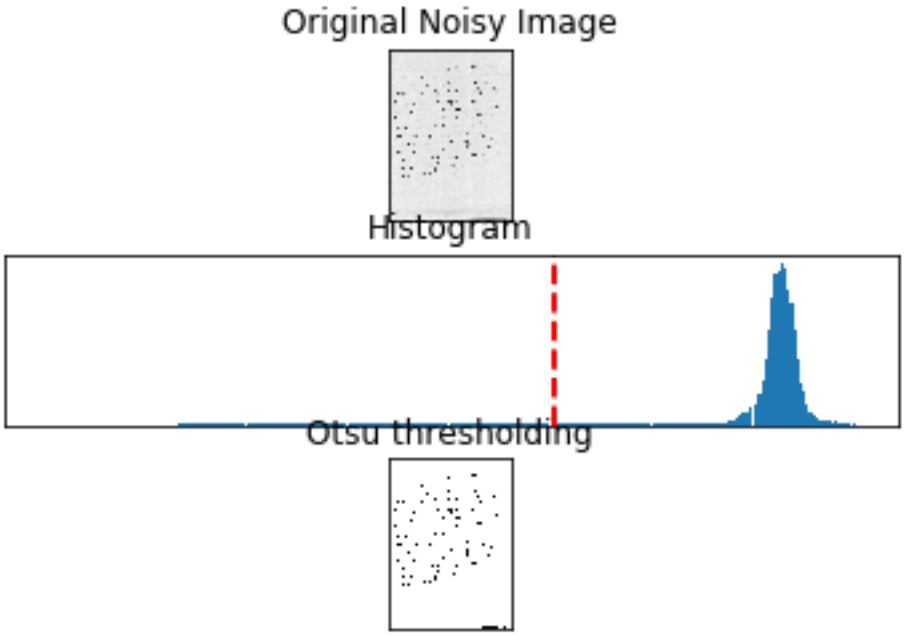
- Histogram shape-based methods, where, for example, the peaks, valleys and curvatures of the smoothed histogram are analyzed
- Clustering-based methods, where the gray-level samples are clustered in two parts as background and foreground (object), or alternately are modeled as a mixture of two Gaussians
- Entropy-based methods result in algorithms that use the entropy of the foreground and background regions, the cross-entropy between the original and binarized image, etc.[1]
- Object Attribute-based methods search a measure of similarity between the gray-level and the binarized images, such as fuzzy shape similarity, edge coincidence, etc.
- Spatial methods [that] use higher-order probability distribution and/or correlation between pixels
- Local methods adapt the threshold value on each pixel to the local image characteristics. In these methods, a different T is selected for each pixel in the image.

So applied to a histogram, we will choose a value in which all the underlying values will be converted to 0 (white) and all those overlying to 255 (black), by converting an image to grayscale into black and white.
In OpenCV to perform the thresholding we can use the cv2.threshold() function
Take the case of the image of the previous leaf. Make the case that we need to recognize the shape of the leaf, but we can not use a histogram. As a first approach we'll try to apply a threshold (a threshold) at random, and then after several attempts able to find an optimal value.

Code and analysis:

```
import cv2
import numpy as np
from matplotlib import pyplot as plt

img = cv2.imread('/home/rahul/Desktop/False.jpg',0)

ret, imgf = cv2.threshold(img, 0, 255, cv2.THRESH_BINARY+cv2.THRESH_OTSU)

plt.subplot(3,1,1), plt.imshow(img,cmap = 'gray')
plt.title('Original Noisy Image'), plt.xticks([]), plt.yticks([])
plt.subplot(3,1,2), plt.hist(img.ravel(), 256)
plt.axvline(x=ret, color='r', linestyle='dashed', linewidth=2)
plt.title('Histogram'), plt.xticks([]), plt.yticks([])
plt.subplot(3,1,3), plt.imshow(imgf,cmap = 'gray')
plt.title('Otsu thresholding'), plt.xticks([]), plt.yticks([])
plt.show()
```

Output:



Analysis:

This image has single threshold (T1). In the simplest implementation, the output is a binary image representing the segmentation. Black pixels correspond to background and white pixels correspond to foreground (or *vice versa*). In simple implementations, the segmentation is determined by a single parameter known as the *intensity threshold*. In a single pass, each pixel in the image is compared with this threshold. If the pixel's intensity is higher than the threshold, the pixel is set to, say, white in the output. If it is less than the threshold, it is set to black. Image can be correctly segmented this way can be determined by looking at an intensity histogram of the image.

Calculation of Thresolding value:

Code and analysis:

```
import cv2
import numpy as np

img = cv2.imread('/home/rahul/Documents/project/Scan_Guwahati_Handwriting_data25.8.14/WanjoplangWansai-13/True.jpg',0)
#blur = cv2.GaussianBlur(img,(5,5),0)

# find normalized_histogram, and its cumulative distribution functio
hist = cv2.calcHist([img],[0],None,[256],[0,256])
hist_norm = hist.ravel()/hist.max()
Q = hist_norm.cumsum()
bins = np.arange(256)
fn_min = np.inf
```

```
thresh = -1
xrange=range
for i in xrange(1,256):
 p1,p2 = np.hsplit(hist_norm,[i]) # probabilities
 q1,q2 = Q[i],Q[255]-Q[i] # cum sum of classes
 if q1 == 0:
   q1 = 0.00000001
 if q2 == 0:
   q2 = 0.00000001
 b1,b2 = np.hsplit(bins,[i]) # weights
 # finding means and variances
 m1,m2 = np.sum(p1*b1)/q1, np.sum(p2*b2)/q2
 v1,v2 = np.sum(((b1-m1)**2)*p1)/q1,np.sum(((b2-m2)**2)*p2)/q2
 # calculates the minimization function
 fn = v1*q1 + v2*q2
 if fn < fn_min:
   fn_min = fn
   thresh = i
# find otsu's threshold value with OpenCV function
otsu = cv2.threshold(img,0,255,cv2.THRESH_BINARY+cv2.THRESH_OTSU)
print (thresh)
```

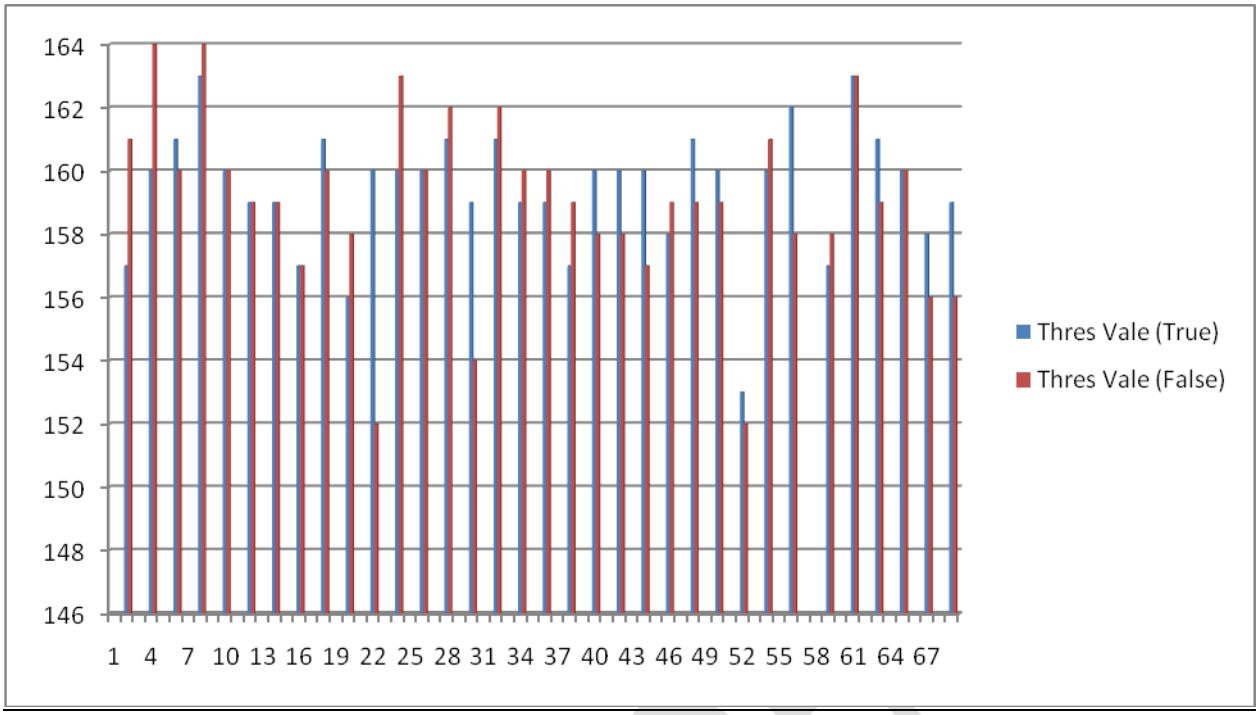
Output:

File Name	Thresold value (True)	Thresold value (False)
AbhilashSingh-9	157	161
AnmolKerketta	160	164
ArindamRoy-6	161	160
BanashreePaul-32	163	164
BethshebdaJRaththap-31	160	160
D.JeromeSudeep-16	159	159
DarvilHamarbabiangC.Sohliva-2	159	159

DebashisBurman-12	157	157
DipayanDas-8	161	160
DiskshantPatodia-23	156	158
FrancisNcarzary-30	160	152
GauravChaubey-29	160	163
HemantaAlley-10	160	160
HimanshuChoudhury-21	161	162
IawalinNongbri-28	159	154
IraniMahanta-27	161	162
KynsaiStephanKhdnsam-18	159	160
NamrataDas-25	159	160
NitunjitBrahma-1	157	159
ParthaPratimSinha-17	160	158
PriyankBaruah-24	160	158
PriyanshuThakuria-22	160	157

PujaMedhi-34	158	159
Pyndapbiang John Lyngdoh-20	161	159
RinkyGurumayum-33	160	159
SamujjalSarmah-5	153	152
SharmilaGayari-26	160	161
ShijoGeorge-11	162	158
SiddarthaSauravGogoi	157	158
SurajKumarSarmah-14	163	163
TapasDas-15	161	159
ThinggujamJoseph-4	160	160
WangmaoS.Konyak-19	158	156
WanjoplangWansai-13	159	156

Graph:



Analysis:

From the above graph, it is noticed that, most of the cases, thresolding value of the false images are much more higher than True images. And, some of the cases, the thresolding vcalue of the True and False images are almost equal. And, there are some types of samples are acted in some different ways where, the value of True images are higher than False images. So, from the graph, we can predict that the higher thresolding value of the samples are included in the flase part and others are included into true samples.

6.1.9 Histogram matching:

An image histogram is a type of histogram that acts as a graphical representation of the tonal distribution in a digital image.[1] It plots the number of pixels for each tonal value. By looking at the histogram for a specific image a viewer will be able to judge the entire tonal distribution at a glance.

Image histograms are present on many modern digital cameras. Photographers can use them as an aid to show the distribution of tones captured, and whether image detail has been lost to blown-out highlights or blacked-out shadows.[2] This is less useful when using a raw image format, as the dynamic range of the displayed image may only be an approximation to that in the raw file.

6.1.10 Histogram Calculation in OpenCV:

We use cv2.calcHist() function to find the histogram. Let's familiarize with the function and its parameters :

cv2.calcHist(images, channels, mask, histSize, ranges[, hist[, accumulate]])

1. images : it is the source image of type uint8 or float32. it should be given in square brackets, ie, “[img]”.

2. channels : it is also given in square brackets. It the index of channel for which we calculate histogram. For example, if input is grayscale image, its value is [0]. For color image, you can pass [0],[1] or [2] to calculate histogram of blue,green or red channel respectively.

3. mask : mask image. To find histogram of full image, it is given as “None”. But if you want to find histogram of particular region of image, you have to create a mask image for that and give it as mask. (I will show an example later.)

4. histSize : this represents our BIN count. Need to be given in square brackets. For full scale, we pass [256].

5. ranges : this is our RANGE. Normally, it is [0,256].

In image processing, histogram matching or histogram specification is the transformation of an image so that its histogram matches a specified histogram.The well-known histogram equalization method is a special case in which the specified histogram is uniformly distributed.

It is possible to use histogram matching to balance detector responses as a relative detector calibration technique. It can be used to normalize two images, when the images were acquired at the same local illumination (such as shadows) over the same location, but by different sensors, atmospheric conditions or global illumination

In typical real-world applications, with 8-bit pixel values (discrete values in range [0, 255]), histogram matching can only approximate the specified histogram. All pixels of a particular value in the original image must be transformed to just one value in the output image.

Exact histogram matching is the problem of finding a transformation for a discrete image so that its histogram exactly matches the specified histogram.[4] Several techniques have been proposed for this. One simplistic approach converts the discrete-valued image into a continuous-valued image and adds small random values to each pixel so their values can be ranked without ties. However, this introduces noise to the output image.

Because of this there may be holes or open spots in the output matched histogram.

To compare two histograms (and), first we have to choose a metric ( ) to express how well both histograms match.

OpenCV implements the function compareHist to perform a comparison. It also offers 4 different metrics to compute the matching:

Correlation (CV_COMP_CORREL)

$$d(H_1, H_2) = \frac{\sum_I (H_1(I) - \bar{H_1})(H_2(I) - \bar{H_2})}{\sqrt{\sum_I (H_1(I) - \bar{H_1})^2 \sum_I (H_2(I) - \bar{H_2})^2}}$$

Where

$$\bar{H_k} = \frac{1}{N} \sum_J H_k(J)$$

and is the total number of histogram bins.

Chi-Square (CV_COMP_CHISQR)

$$d(H_1, H_2) = \sum_I \frac{(H_1(I) - H_2(I))^2}{H_1(I)}$$

Intersection (method=CV_COMP_INTERSECT)

$$d(H_1, H_2) = \sum_I \min(H_1(I), H_2(I))$$

It is just another way of understanding the image. By looking at the histogram of an image, we get intuition about contrast, brightness, intensity distribution etc of that image.We can see the image and its histogram. (Remember, this histogram is drawn for grayscale image, not color image). Left region of histogram shows the amount of darker pixels in image and right region shows the amount of brighter pixels. From the histogram, we can see dark region is more than brighter region, and amount of midtones (pixel values in mid-range, say around 127) are very less.

Now we have an idea on what is histogram, we can look into how to find this. Both OpenCV and Numpy come with in-built function for this. Before using those functions, we need to understand some terminologies related with histograms.

BINS :The above histogram shows the number of pixels for every pixel value, ie from 0 to 255. ie we need 256 values to show the above histogram. But consider, what if we need not find the number of pixels for all pixel values separately, but number of pixels in a interval of pixel values? say for example, we need to find the number of pixels lying between 0 to 15, then 16 to 31, ..., 240 to 255. We will need only 16 values to represent the histogram. So what we do is simply split the whole histogram to 16 sub-parts and value of each sub-part is the sum of all pixel count in it. This each sub-part is called "BIN". In first case, number of bins where 256 (one for each pixel) while in second case, it is only 16. BINS is represented by the term histSize in OpenCV docs.

DIMS : It is the number of parameters for which we collect the data. In this case, we collect data regarding only one thing, intensity value. So here it is 1.

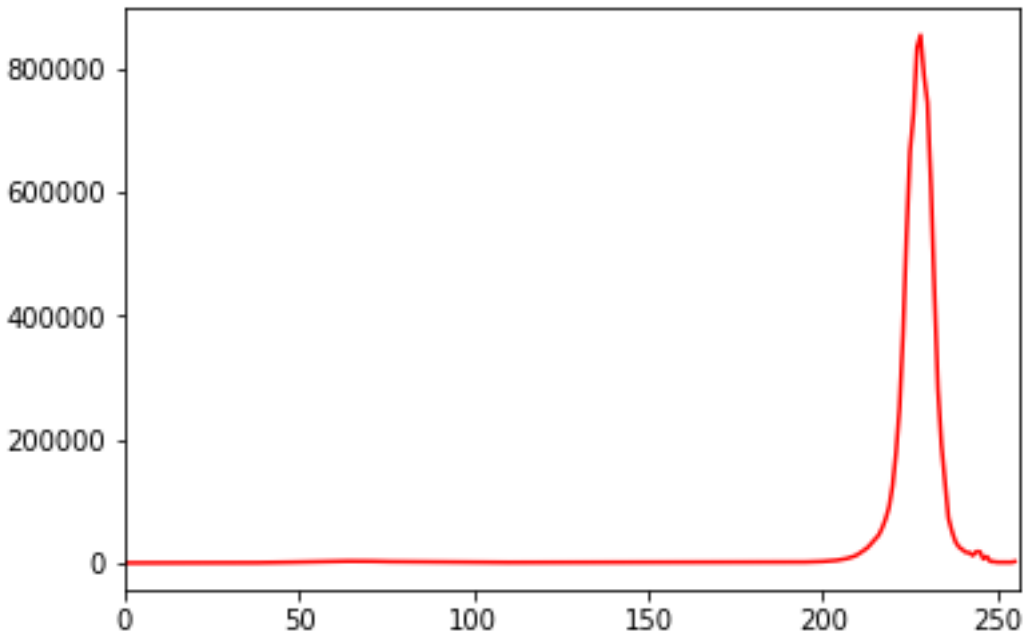
RANGE : It is the range of intensity values we want to measure. Normally, it is [0,256], ie all intensity values.

Code and analysis:

```
import cv2
import numpy as np
from matplotlib import pyplot as plt

img = cv2.imread('/home/rahul/Desktop/True.jpg')
```

```
color = ('b','g','r')
for i,col in enumerate(color):
histr = cv2.calcHist([img],[i],None,[256],[0,256])
plt.plot(histr,color = col)
plt.xlim([0,256])
plt.show()
```

Output:



Analysis:

The total number of pixels constituting the image can be obtained by adding up the number of pixels corresponding to each gray level. Pixel counts that are restricted to a smaller range indicate low contrast.

6.1.11 Stroke matching:

A local image feature is a tiny patch in the image that's invariant to image scaling, rotation and change in illumination. It's like the tip of a tower, or the corner of a window in the image above. Unlike a random point on the background in the image above and other characteristic can be precise detected in most images of the same scene. It is geometricly (translation, rotation, ...) and photometricly (brightness, exposure, ...) invariant.

A good local feature is like the piece you start with when solving a jigsaw puzzle, except on a much smaller scale. It's the eye of the cat or the corner of the table, not a piece on a blank wall.

The extracted local features must be:

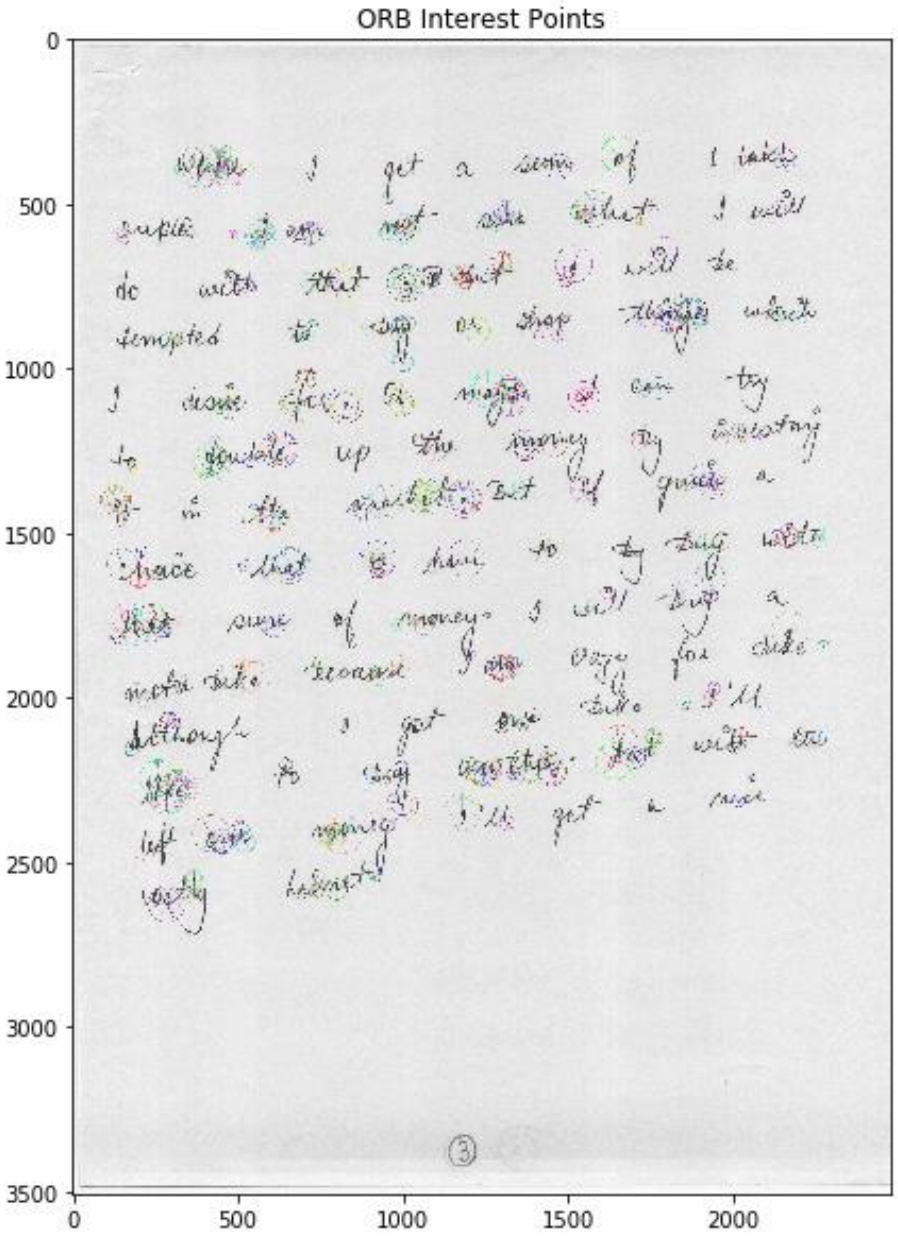
- *Repeatable* and *precise* so they can be extracted from different images showing the same object.
- *Distinctive* to the image, so images with different structure will not have them.

There could be hundreds or thousands of such features in an image. An image matcher algorithm could still work if some of the features are blocked by an object or badly deformed due to change in brightness or exposure. Many local feature algorithms are highly efficient and can be used in real-time applications.

Code:

```
import numpy as np

import pandas as pd

import cv2

import matplotlib.pyplot as plt

import os


dataset_path = '/home/rahul/Documents/project/Scan_Guwahati_Handwriting_data25.8.14/AnmolKerketta'

img_building = cv2.imread(os.path.join(dataset_path, 'False.jpg'))

img_building = cv2.cvtColor(img_building, cv2.COLOR_BGR2RGB)  # Convert from cv's BRG default color order to RGB


orb = cv2.ORB_create()  # OpenCV 3 backward incompatibility: Do not create a detector with `cv2.ORB()`.

key_points, description = orb.detectAndCompute(img_building, None)

img_building_keypoints = cv2.drawKeypoints(img_building,
                                           key_points,
                                           img_building,
                                           flags=cv2.DRAW_MATCHES_FLAGS_DRAW_RICH_KEYPOINTS)
# Draw circles.

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

plt.title('ORB Interest Points')

plt.imshow(img_building_keypoints)

plt.show()
```

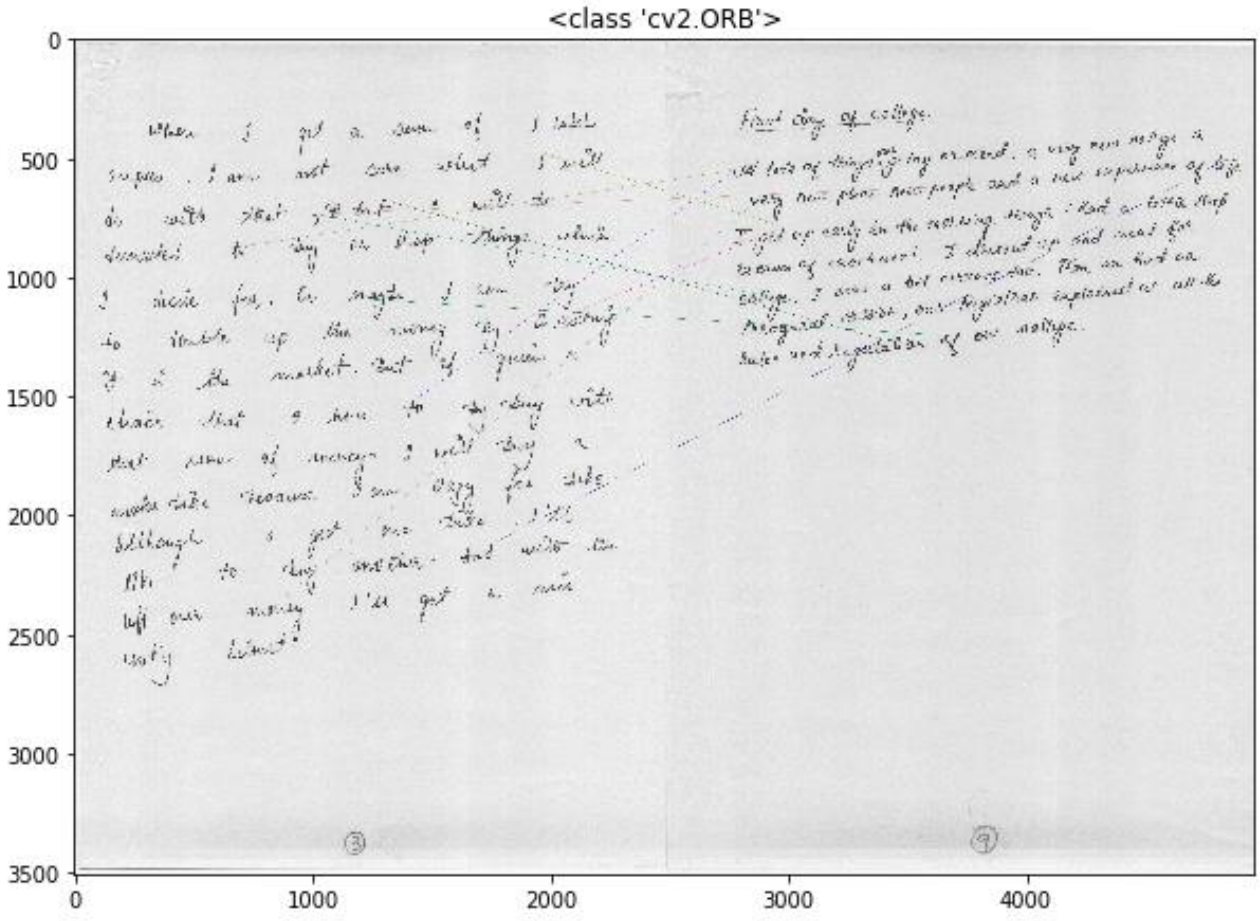
Output:


ORB Interest Points
0
500
1000
1500
2000
2500
3000
3500
0
500
1000
1500
2000

Code:

```
import numpy as np
import pandas as pd
import cv2
import matplotlib.pyplot as plt
import os


def image_detect_and_compute(detector, img_name):
    """Detect and compute interest points and their descriptors."""
    img = cv2.imread(os.path.join(dataset_path, img_name))
    img = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
    kp, des = detector.detectAndCompute(img, None)
    return img, kp, des
def draw_image_matches(detector, img1_name, img2_name, nmatches=10):
    """Draw ORB feature matches of the given two images."""
    img1, kp1, des1 = image_detect_and_compute(detector, img1_name)
    img2, kp2, des2 = image_detect_and_compute(detector, img2_name)


    bf = cv2.BFMatcher(cv2.NORM_HAMMING, crossCheck=True)
    matches = bf.match(des1, des2)
    matches = sorted(matches, key = lambda x: x.distance) # Sort matches by distance.  Best come first.
    img_matches = cv2.drawMatches(img1, kp1, img2, kp2, matches[:nmatches], img2, flags=2) # Show top 10 matches
    plt.figure(figsize=(10, 10))
    plt.title(type(detector))
    plt.imshow(img_matches); plt.show()
orb = cv2.ORB_create()
draw_image_matches(orb, '/home/rahul/Desktop/False.jpg', '/home/rahul/Desktop/True.jpg')
```

Output:



Analysis:

From this feature mtaching process, we used Brute Firce mtaching algorithm. By, this process, we can identify the same strokes or features of characters which are exactly equivalent with each others. So, from this output, we can concluded that, both are written from same writers.

6.1.12 Contours:

Contours can be explained simply as a curve joining all the continuous points (along the boundary), having same color or intensity. The contours are a useful tool for shape analysis and object detection and recognition.

- For better accuracy, use binary images. So before finding contours, apply threshold or canny edge detection.
- findContours function modifies the source image. So if you want source image even after finding contours, already store it to some other variables.

- In OpenCV, finding contours is like finding white object from black background. So remember, object to be found should be white and background should be black.

A sequence of data is obtained by tracing the exterior contour of the signature which allows the application of string-matching algorithms. The exterior and interior contours of the signature are first determined by ignoring small gaps between signature components. The contours are combined into a single sequence so as to define a pseudo-writing path. To match two signatures a non-linear normalization method, viz., dynamic time warping, is applied to segment them into curves. Shape descriptors based on Zernike moments are extracted as features from each segment. A harmonic distance is used for measuring signature similarity. Performance is significantly better than that of a word-shape based signature verification method.

To draw the contours, cv2.drawContours function is used. It can also be used to draw any shape provided you have its boundary points. Its first argument is source image, second argument is the contours which should be passed as a Python list, third argument is index of contours (useful when drawing individual contour. To draw all contours, pass -1) and remaining arguments are color, thickness etc.

6.1.13 Contour Approximation Method:

This is the third argument in cv2.findContours function. What does it denote actually?

Above, we told that contours are the boundaries of a shape with same intensity. It stores the (x,y) coordinates of the boundary of a shape. But does it store all the coordinates ? That is specified by this contour approximation method.

If you pass cv2.CHAIN_APPROX_NONE, all the boundary points are stored. But actually do we need all the points? For eg, you found the contour of a straight line. Do you need all the points on the line to represent that line? No, we need just two end points of that line. This is what cv2.CHAIN_APPROX_SIMPLE does. It removes all redundant points and compresses the contour, thereby saving memory.

6.1.14 Similarity measure using Hausdorff distance:

Image similarity measuring process is generally composed of an information mining in each image which results in an image signature and then a signature comparison to make the decision about the image similarity. In the scope of binary images, we propose to replace the information mining by a new straight image comparison which does not require a priori knowledge. The second stage is then replaced by a decision process based on the image comparison.

Classical Hausdorff measure need to express image as a feature matrix firstly, then calculate feature values or feature vectors, so it is time-consuming. Otherwise, it is difficult for part pattern matching when shadow and noise existed.

Set A = {a1,….,ap} and B = {b1,….,bq}

Hausdorff distance:

$$H(A,B) = \max(h(A,B), h(B,A))$$

Directed Hausdorff distance:

h(A,B) ranks each point of A based on its nearest point of B and uses the most mismatched point.

$$\vec{h}(A,B) = \max_{a \in A} \min_{b \in B} \| a - b \|$$

6.1.15 Morphological Dilation:

Morphology is a broad set of image processing operations that process images based on shapes. Morphological operations apply a structuring element to an input image, creating an output image of the same size. In a morphological operation, the value of each pixel in the output image is based on a comparison of the corresponding pixel in the input image with its neighbors. By choosing the size and shape of the neighborhood, you can construct a morphological operation that is sensitive to specific shapes in the input image.

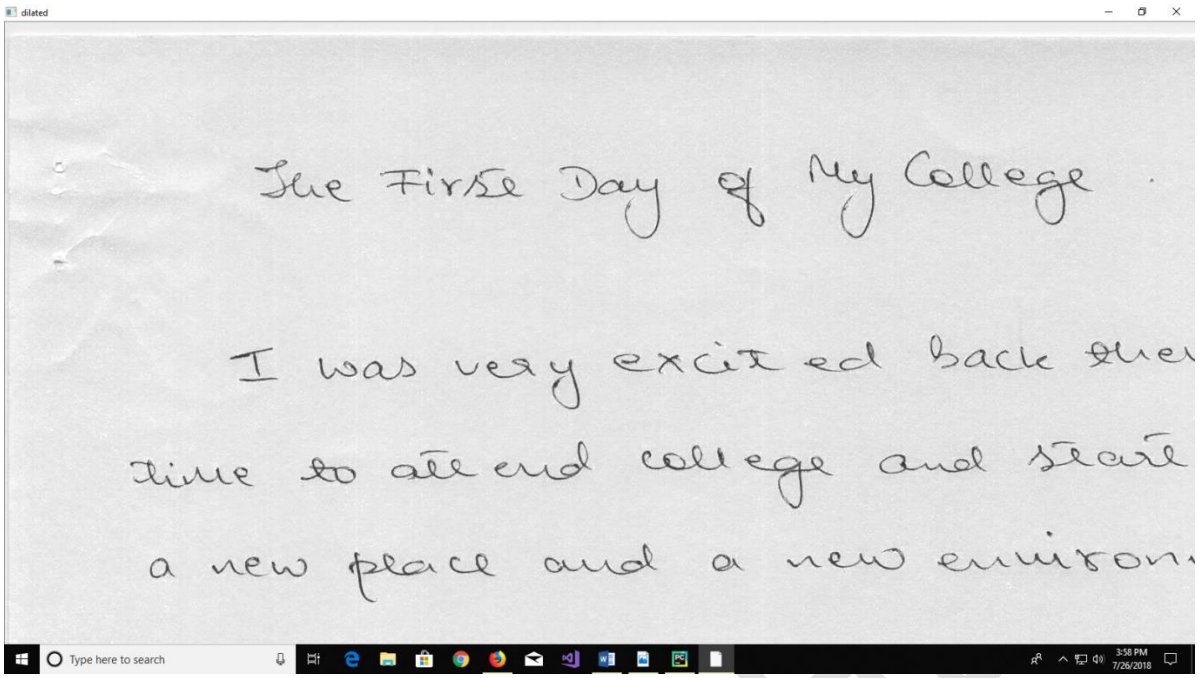
The most basic morphological operations are dilation and erosion. Dilation adds pixels to the boundaries of objects in an image, while erosion removes pixels on object boundaries. The number of pixels added or removed from the objects in an image depends on the size and shape of the structuring element used to process the image. In the morphological dilation and erosion operations, the state of any given pixel in the output image is determined by applying a rule to the corresponding pixel and its neighbors in the input image. The rule used to process the pixels defines the operation as a dilation or an erosion.

Stroke thickening:

The value of the output pixel is the maximum value of all the pixels in the input pixel's neighborhood. In a binary image, if any of the pixels is set to the value 1, the output pixel is set to 1.

Pixels beyond the image border are assigned the minimum value afforded by the data type.

For binary images, these pixels are assumed to be set to 0. For grayscale images, the minimum value for uint 8 images is 0.



Stroke thinning:

The value of the output pixel is the minimum value of all the pixels in the input pixel's neighborhood. In a binary image, if any of the pixels is set to 0, the output pixel is set to 0.

Pixels beyond the image border are assigned the maximum value afforded by the data type.

For binary images, these pixels are assumed to be set to 1. For grayscale images, the maximum value for uint8 images is 255.

Morphological transformations are some simple operations based on the image shape. It is normally performed on binary images. It needs two inputs, one is our original image, second one is called structuring element or kernel which decides the nature of operation. Two basic morphological operators are Erosion and Dilation.

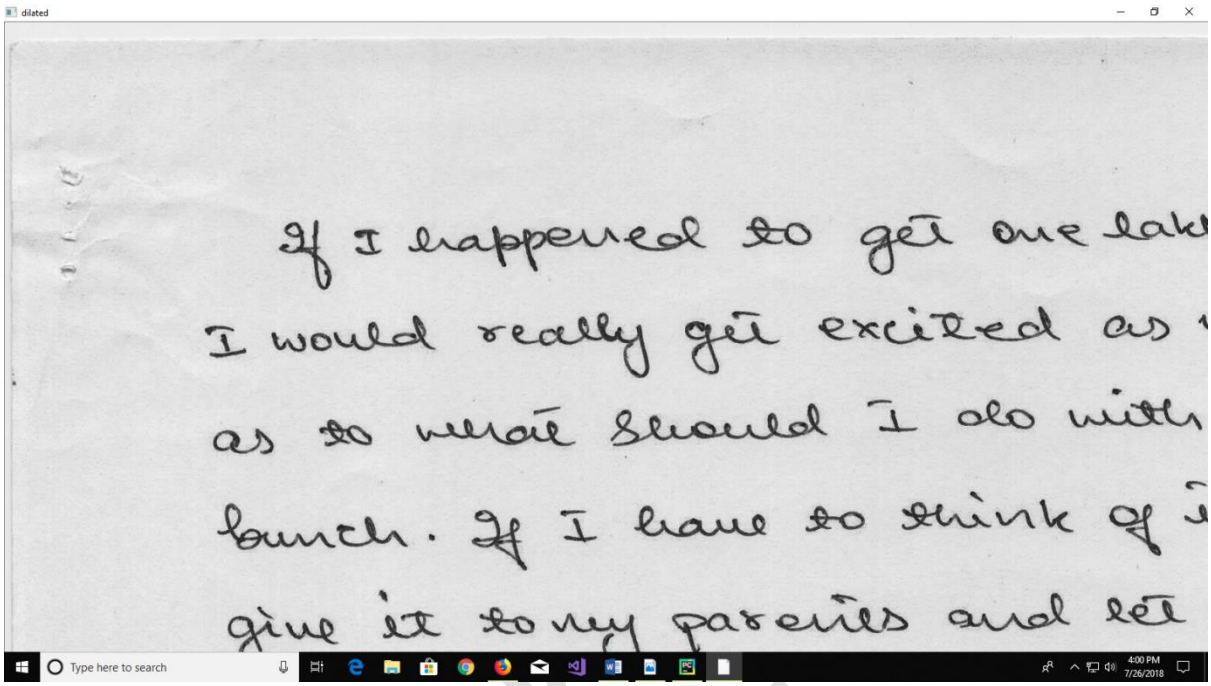
Operations	Rule
Dilation	The value of the output pixel is the maximum value of all the pixels in the input pixel's neighborhood. In a binary image, if any of the pixels is set to the value1, the output pixel is set to 1.
Erosion	The value of the output pixel is the minimum value of all the pixels in the input pixel's neighborhood. In a binary image, if any of the pixels is set to 0, the output pixel is set to 0.

Erosion

The basic idea of erosion is just like soil erosion only, it erodes away the boundaries of foreground object (Always try to keep foreground in white). The kernel slides through the image (as in 2D convolution). A pixel in the original image (either 1 or 0) will be considered 1 only if all the pixels under the kernel is 1, otherwise it is eroded (made to zero).

So what happends is that, all the pixels near boundary will be discarded depending upon the size of kernel. So the thickness or size of the foreground object decreases or simply white region decreases in the image. It is useful for removing small white noises (as we have seen in colorspace chapter), detach two connected objects etc.

Here, as an example, I would use a 5x5 kernel with full of ones. Let's see it how it works:



Code and analysis:

```
import cv2

import numpy as np

c = cv2.imread('/home/rahul/Desktop/True.jpg')

kernel = np.ones((5,5),np.uint8)

erosion = cv2.erode(c,kernel,iterations = 1)

cv2.imshow('erosion', erosion)

r = cv2.waitKey(0)

cv2.destroyAllWindows()
```

Analysis:

Basically, by this process, we used to do the thinning of the boundaries of characters. Erosion generally decreases the sizes of objects and removes small anomalies by subtracting objects with a radius smaller than the structuring element of the images. With grayscale images, erosion reduces

the brightness (and therefore the size) of bright objects on a dark background by taking the neighbourhood minimum when passing the structuring element over the image. With binary images, erosion completely removes objects smaller than the structuring element and removes perimeter pixels from larger image objects.

Dilation

It is just opposite of erosion. Here, a pixel element is '1' if atleast one pixel under the kernel is '1'. So it increases the white region in the image or size of foreground object increases. Normally, in cases like noise removal, erosion is followed by dilation. Because, erosion removes white noises, but it also shrinks our object. So we dilate it. Since noise is gone, they won't come back, but our object area increases. It is also useful in joining broken parts of an object.

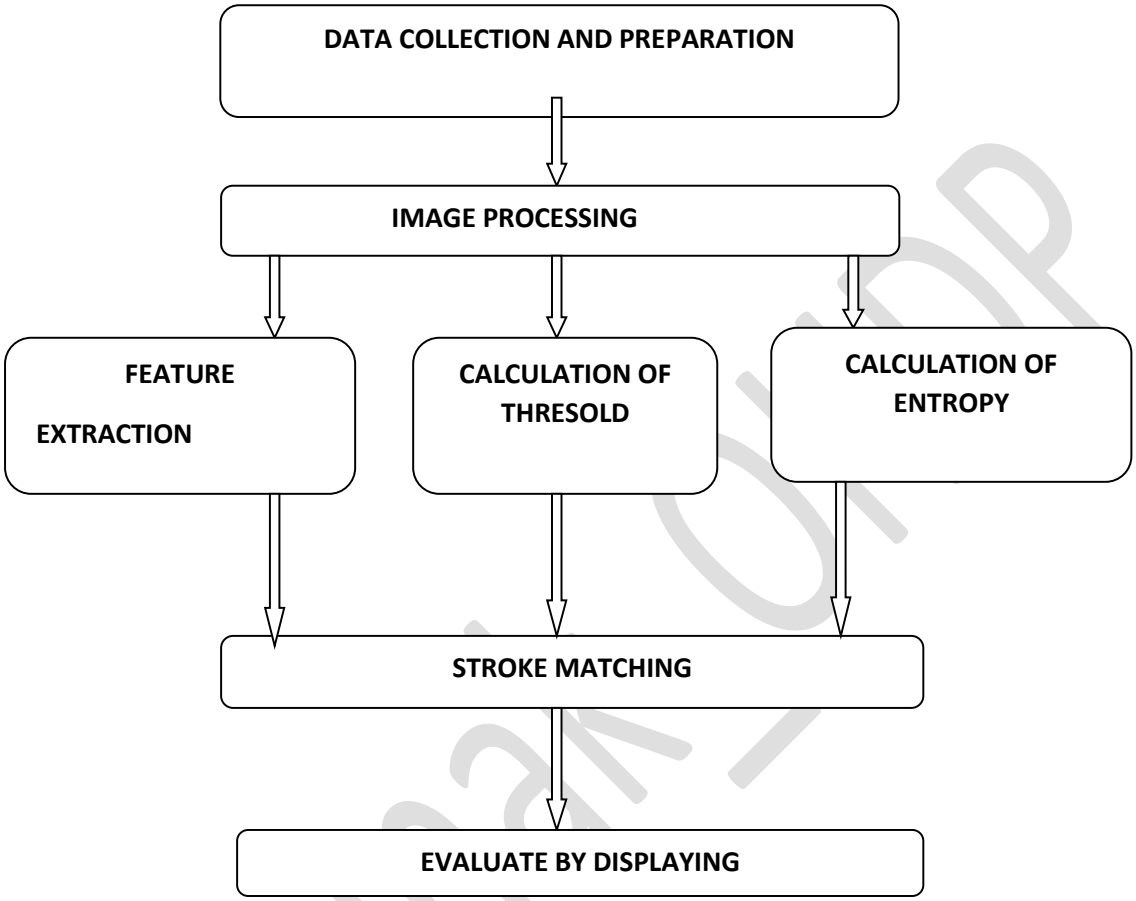
Code and analysis:

```
import cv2

import numpy as np

c = cv2.imread('/home/rahul/Desktop/True.jpg')

kernel = np.ones((5,5),np.uint8)

dialation = cv2.dilate(c,kernel,iterations = 1)

cv2.imshow('dialation',dialation)

r = cv2.waitKey(0)

cv2.destroyAllWindows()
```

Analysis:

Dilation generally increases the sizes of objects, filling in holes and broken areas, and connecting areas that are separated by spaces smaller than the size of the structuring element. With grayscale images, dilation increases the brightness of objects by taking the neighbourhood maximum when passing the structuring element over the image. With binary images, dilation connects areas that are separated by spaces smaller than the structuring element and adds pixels to the perimeter of each image object.

7 CHAPTER: DATA FLOW



8 CHAPTER : CONCLUSION AND RECOMMENDATIONS

8.1 5.1. CONCLUSION

This paper presented a framework for writer identification from offline handwritten documents. The objective of this study was not to present a novel set of features but to propose an effective combination of existing features which results in improved identification rates. Traditionally, multiple features are combined into a single large feature vector or the distances of multiple features are combined using a weighted average. The proposed framework, instead of considering the features individually and applying a single distance function, combines these features using multiple distance functions. We have presented an effective method for writer identification in handwritten documents. The technique is based on the extraction of forms that a specific writer would use frequently as he draws the characters. The achieved identification rates are very promising and validate the argument of the existence of redundant patterns in a handwritten text. The identification can be further improved by using a more adaptive division of text following the natural direction of hand. Varying the window size n, this method can also be applied to languages like English etc. The system can be made more robust making it capable of automatically adjusting the window size depending upon the writing details. Moreover, the system can be extended beyond identification to perform the verification of the author as well. In this paper, a study on the position accuracy of recorded offline handwriting data is presented. It is motivated by the use ofhandwriting characteristics in the forensic investigation of questioned signatures. An experiment is detailed that focuses on cross-coupling effects between pen orientation and pen-position records. In order to employ recorded writing movements in the analysis / measuring of residual ink traces on paper, a method for assigning / superimposing digitized offline ink traces and captured online pen trajectories is provided.

8.2 5.3. RECOMMENDATIONS:

In case of Lie detection, suspected person may be identified by asking him to write a statement about a particular incident in which his involvement is suspected. A typical pattern will emerge through the analysis of perceptual behavioural pattern of the suspect which will reflect the stressed condition of the person and it may be used as an aid to detect whether a person is lying or not i.e. the described incident is true or false.

9 REFERENCES

- https://stackoverflow.com/
- https://pdfs.semanticscholar.org/0e0b/236c6a88a1183e3c12f47601330f8d4d3bb6.pdf
- http://people.idsia.ch/~juergen/nips2009.pdf
- https://www.matec-conferences.org/articles/matecconf/pdf/2016/39/matecconf_cscc2016_05004.pdf
- https://opencv.org/
- https://anaconda.org/anaconda/python
- http://doc.qt.io/qt-5/qtdesigner-manual.html
- https://papers.nips.cc/paper/3449-offline-handwriting-recognition-with-multidimensional-recurrent-neural-networks