Cancer Detection Through Handwriting

Alaa Tarek

Shorouk Alalem

Maryam El-Fdaly

Nehal Fooda

Abstract

Having a look at the medical field in the previous years and the drawbacks in it, Egypt's health level is declining year after year; due to the statistical study on the health level published by the British medical journal Lancet in 2016 Egypt ranked 124 out of 188 countries. Cancer is a major burden of disease worldwide. Each year, 10,000,000 of people are diagnosed with cancer around the world, and more than half of the patients eventually die because of it. In many countries, cancer ranks the second most common cause of death following cardiovascular diseases. With significant improvement in treatment and prevention of cardiovascular diseases, cancer has or will soon become the number one killer in many parts of the world. Nearly 90,000 people do not know they have got cancer until they arrive at Accident and Emergency wards, by that time only 36 percent will live longer than a year. So, we needed to find controlled ways to diagnose patients earlier. As no aspect of human life has escaped the impact of the information age, and perhaps in no area of life is information more critical than in health and medicine. However, computers have become available for all aspects of human endeavors. After so, we have designed a program that could detect if a person has cancer or not through your handwriting. We chose "efficiency, cost, and applicability" as the design requirements that have been tested. The program could be tested by scanning the text, searching for specific features that are related to cancer and displaying "1" or "0" according to your state. After testing the program many times, we finally reached a mean efficiency of 93.75%. So, this program saves lives, time and money.

Introduction

Each country has its own challenges, but the distinction is how they deal with them. Pollution, population growth, water crisis, and public health issues are some of Egypt's grand challenges. The death rate in Egypt due to health issues was 4.77 per 1000 people in 2014. That's why we chose to work on the public health issue. Cancer, the second globally leading cause of death, was responsible for 8.8 million deaths in 2015 according to "World Health Organization". Nearly 1 in 6 death cases is due to cancer with approximately 14 million new cases in 2012. In addition, Cancer Researchers at the UK refer to that almost half of people who get cancer are diagnosed late, which makes treatment less likely to succeed. As a result, it reduces their chances of survival as shown in (Chart (1)). The latest solution was early detection of cancer through handwriting by a graphologist (Kanfer Method). Although this solution reached an accuracy of 84%, it takes a lot of time, expensive, uncommon and the results differ from an expert to another. We developed this prior solution by constructing a program that detects cancer through handwriting instead of the graphologist. This program has met our design requirements (high efficiency, low cost, and availability). It was designed using "Deep Learning" which extracts specific features from handwriting samples like the slant of writing (baseline), variable pressure…etc. Consequently, the result is shown as "1" for a cancer patient or "0" for normal people. To calculate the baseline, we depended on "Oscillation Theory of handwriting" which states that handwriting arises from orthogonal oscillations horizontal and vertical. So, writing slant is somehow a function of various parameters resulted from the combination of vertical and horizontal oscillations. That's why we used Transverse wave equation "$y(x, \mathrm{t}) = ym \sin(kx + \mathrm{v}t)$" to find the vertical displacement. For the programing language, we chose to work on "Python" because it is free which can be used by a lot of people and does not require ones with sufficient funds to buy a

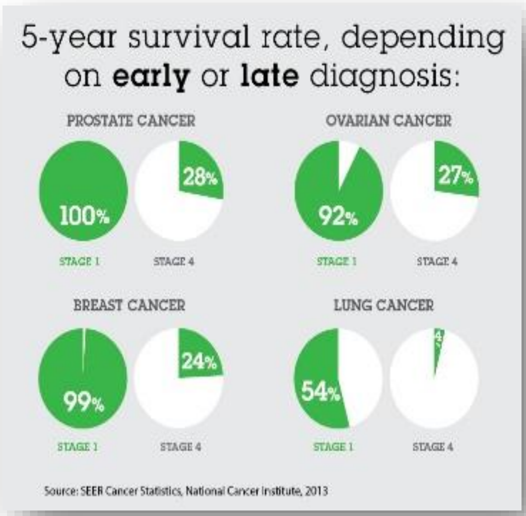


Chart 1

license. The efficiency was increased by using a large number of samples grouped into 4 equal sub-groups in training the program to minimize the percentage of errors as much as possible.

Materials

Hardware

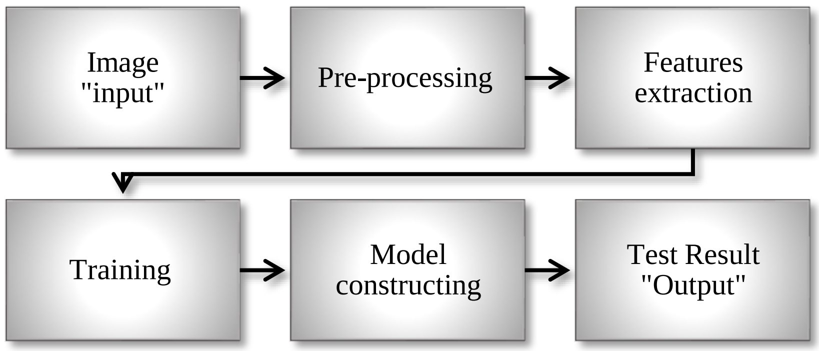
- White papers (A4 40 gm)
- PRIMA Bronzo Blue Pen (0.7 mm)
- Laptop
- Mobile camera

Software

- Visual Studio 2015
- Python 2.7.13
- cmake-3.10.1
- matplotlib-2.1.0
- numpy-1.10.0
- scipy-1.0.0
- scikit_learn-0.19.1

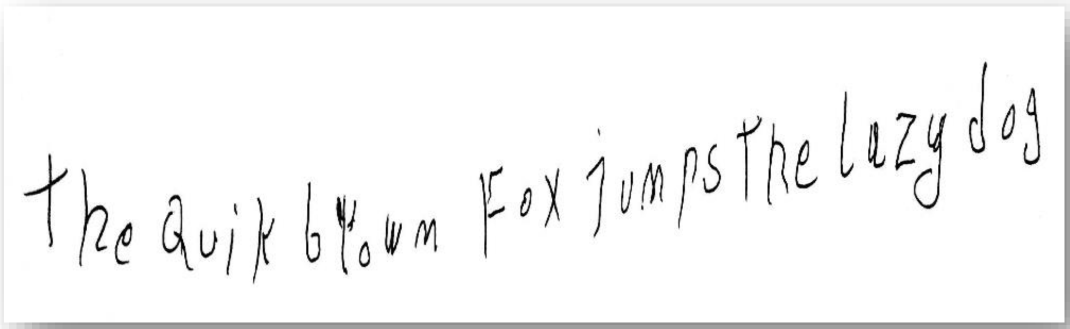
Methods

The whole program follows specific steps; the chart below illustrates them.

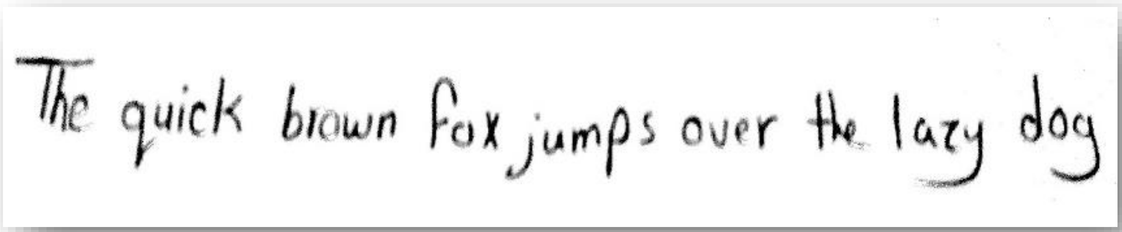


Constructing the program in detail:

1) We determined a standard sentence which contains all English alphabet "the quick brown fox jumps over the lazy dog" to facilitate comparison between samples.
2) We collected 200 samples from different cancer patients with various ages and gender (first group).



3) 200 samples were collected from normal people in different ages and gender (a second group which we call control group).


The quick brown fox jumps over the lazy dog

4) We extracted the features of cancer patients' handwriting from the samples.
5) We compared these features and determining the most frequently appear.
6) We built the program using open CV with python language.
7) The samples of each group were divided into two equal groups (training and testing).

   So, we have:

   - 100 patient samples for training
   - 100 patient samples for testing.
   - 100 normal samples for training.
   - 100 normal samples for testing.
8) We divided each testing group into four sub-groups (each group contains 25 samples).

Design 1

9) We trained the program on the two training groups using machine learning.
10) The efficiency wasn't high enough, we redesigned the program.

Detect variable pressure.

This code finds the area of the text so we can determine the text thickness, which in return represents the pressure.

1) Import open cv,numpy and pyplot.

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

2) Define our dataset path.

```
path2 = "C:\Python27\OCR\datasets\sick"
path ="C:\Python27\OCR\datasets\good"
image = path + "\g99"
image_string = image + ".jpg"
image_tangled = image + "_tangled.jpg"
image_round = image + "_round.jpg"
```

3) Load the image.

```
img1 = cv2.imread(image_string)
```

4) Define the resize dimension and arc length factor for poly contours functions.

```
resize_dim = (1240, 230)
circle_poly_factor = 0.001
tangled_poly_factor = 0.035
```

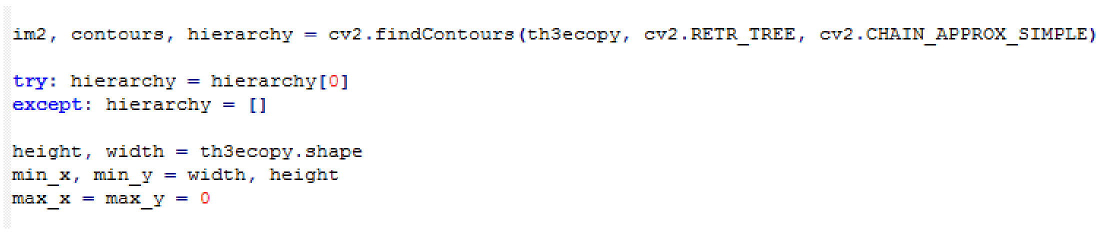
5) Convert the image into gray scale and blur it to reduce noise, then convert it to binary image.

```
img2gray = cv2.cvtColor(img1,cv2.COLOR_BGR2GRAY)

blur = cv2.GaussianBlur(img2gray,(5,5), 0)

thresh = cv2.adaptiveThreshold(blur, 255, cv2.ADAPTIVE_THRESH_GAUSSIAN_C, cv2.THRESH_BINARY,11,2)
```

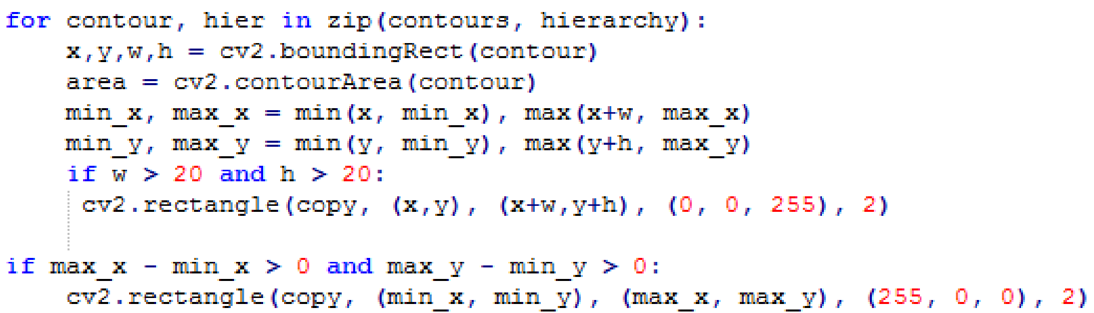
6) Dilate the thickness of the writing.

```
kernel = np.ones((5,5), np.uint8)
img_dilation = cv2.dilate(thresh, kernel, iterations=1)
cv2.imshow('dilated',img_dilation)
```

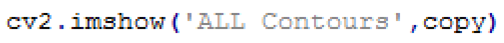
7) find the contours inside the image to classify text and charachters later


```
im2, contours, hierarchy = cv2.findContours(th3ecopy, cv2.RETR_TREE, cv2.CHAIN_APPROX_SIMPLE)

try: hierarchy = hierarchy[0]
except: hierarchy = []

height, width = th3ecopy.shape
min_x, min_y = width, height
max_x = max_y = 0
```

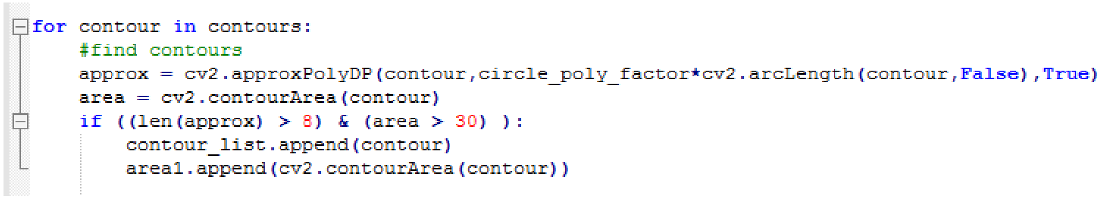
8) computes the bounding box for the contour, and draws it on the frame.


```
for contour, hier in zip(contours, hierarchy):
    x,y,w,h = cv2.boundingRect(contour)
    area = cv2.contourArea(contour)
    min_x, max_x = min(x, min_x), max(x+w, max_x)
    min_y, max_y = min(y, min_y), max(y+h, max_y)
    if w > 20 and h > 20:
     cv2.rectangle(copy, (x,y), (x+w,y+h), (0, 0, 255), 2)

if max_x - min_x > 0 and max_y - min_y > 0:
    cv2.rectangle(copy, (min_x, min_y), (max_x, max_y), (255, 0, 0), 2)
```

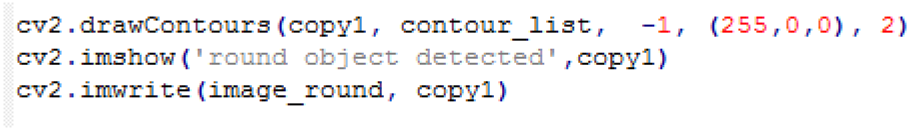
9) draw image with all rect contours.


```
cv2.imshow('ALL Contours',copy)
```

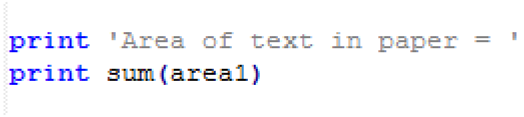
10) Find contours and store their areas in list.


```
for contour in contours:
    #find contours
    approx = cv2.approxPolyDP(contour,circle_poly_factor*cv2.arcLength(contour,False),True)
    area = cv2.contourArea(contour)
    if ((len(approx) > 8) & (area > 30) ):
        contour_list.append(contour)
        area1.append(cv2.contourArea(contour))
```

11) Draw the image with circular contours and save it.


```
cv2.drawContours(copy1, contour_list,  -1, (255,0,0), 2)
cv2.imshow('round object detected',copy1)
cv2.imwrite(image_round, copy1)
```

12) Print the summtion area of all contours "area of text".


```
print 'Area of text in paper = '
print sum(area1)
```

Slope of writing.

This code finds the area of the maximum contour and measure its slope, which in return measure the slope of the text

1) Import open cv and numpy.

```
from __future__ import division
import math
import cv2
import numpy as np
```

2) Define our dataset path.

```
path2 = "C:\Python27\OCR\datasets\sick"
path ="C:\Python27\OCR\datasets\good"
image = path2 + "\c22"
image_string = image + ".jpg"
```

3) Convert the image into gray scale, threshold and dilate the thickness of the writing.

```
gray = cv2.cvtColor(image,cv2.COLOR_BGR2GRAY)
cv2.imshow('gray',gray)
cv2.waitKey(0)

ret,thresh = cv2.threshold(gray,127,255,cv2.THRESH_BINARY_INV)
cv2.imshow('second',thresh)
cv2.waitKey(0)

kernel = np.ones((5,100), np.uint8)
img_dilation = cv2.dilate(thresh, kernel, iterations=1)
cv2.imshow('dilated',img_dilation)
cv2.waitKey(0)
```

4) Find the contours inside an the image, this defines areas of text.

```
im2,ctrs, hier = cv2.findContours(img_dilation.copy(), cv2.RETR_TREE, cv2.CHAIN_APPROX_SIMPLE)
```

5) Draw bounding rectangles around the found contours and get the area of every contour.

```
for i, ctr in enumerate(sorted_ctrs):
    rect = cv2.minAreaRect(ctr)

    box = cv2.boxPoints(rect)
    boxpointscopy.append(cv2.boxPoints(rect))

    box = np.int0(box)
    cv2.drawContours(image,[box],0,(0,0,255),2)

    area2.append(cv2.contourArea(ctr))

    cv2.imshow('segment no:'+str(i),image)
    cv2.waitKey(0)
```

6) Get the index of the biggest contour in the list of contour.

```
maxarea = area2.index(max(area2))
minarea = area2.index(min(area2))

print 'max area contour = '
print maxarea
```

7) Get the index of the biggest contour in the list of contour.

```
print boxpointscopy[maxarea][0]
print boxpointscopy[maxarea][1]
print boxpointscopy[maxarea][2]
print boxpointscopy[maxarea][3]
```

8) Loop over all contours to get the slope of every one.

```
for i, ctr in enumerate(sorted_ctrs):
    dy.append( boxpointscopy[i][1][1] - boxpointscopy[i][0][1])
    dx.append( boxpointscopy[i][1][0] - boxpointscopy[i][0][0])
    slope.append(dy[i] / dx[i])
    print 'incline = '
    print("%.2f" % slope[i])
```

9) Print the text slope.

```
print 'TEXT SLOPE IS = '
print slope[maxarea]
print "dy of max area contour is = "
print dy[maxarea]
print "dx of max area contour is = "
print dx[maxarea]
print "x angle = "
print math.degrees(math.atan2(dy[maxarea], dx[maxarea]))

text_slope = slope[maxarea]
```

Design 2

11) We trained the program using deep learning.

12) The efficiency was calculated.

Training code.

1) Import libraries.

```
import cv2
import imutils
import numpy as np
import os
from sklearn import svm
from sklearn.externals import joblib
from scipy.cluster.vq import *
from sklearn.preprocessing import StandardScaler
from sklearn.neural_network import MLPClassifier
from sklearn.grid_search import GridSearchCV
import time
```

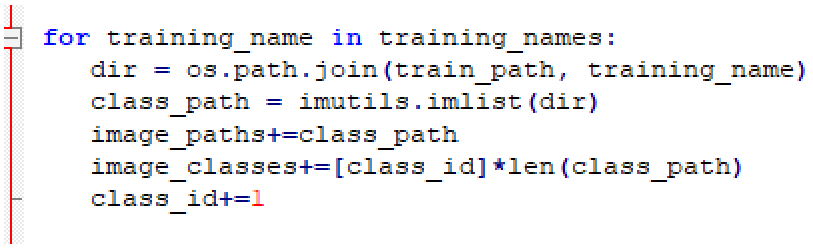
2) Get the training classes names and store them in a list.

```
train_path = "dataset/train/"
training_names = os.listdir(train_path)
```

3) Get all the path to the images and save them in a list.

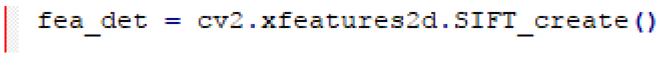
```
image_paths = []
image_classes = []
class_id = 0
```

4) Get the image one after one from the list.



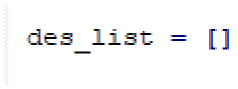
```
for training_name in training_names:
    dir = os.path.join(train_path, training_name)
    class_path = imutils.imlist(dir)
    image_paths+=class_path
    image_classes+=[class_id]*len(class_path)
    class_id+=1
```

5) Create feature extraction and key point detector objects.



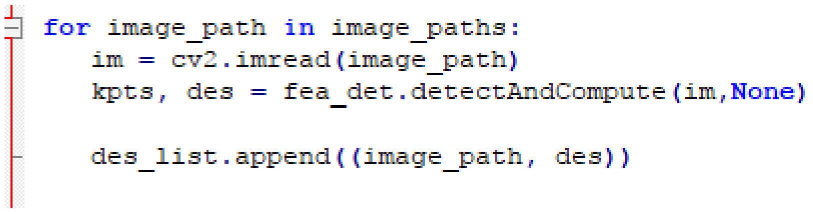
```
fea_det = cv2.xfeatures2d.SIFT_create()
```

6) List where all the descriptors are stored.



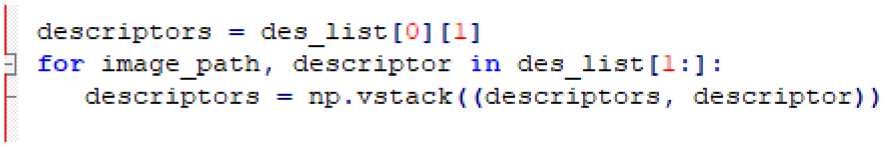
```
des_list = []
```

7) List all images with their descriptor.



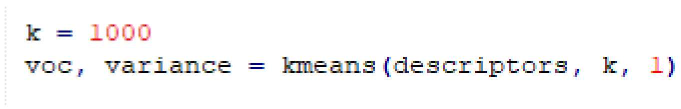
```
for image_path in image_paths:
    im = cv2.imread(image_path)
    kpts, des = fea_det.detectAndCompute(im,None)

    des_list.append((image_path, des))
```

8) Stack all the descriptors vertically in a numpy array.



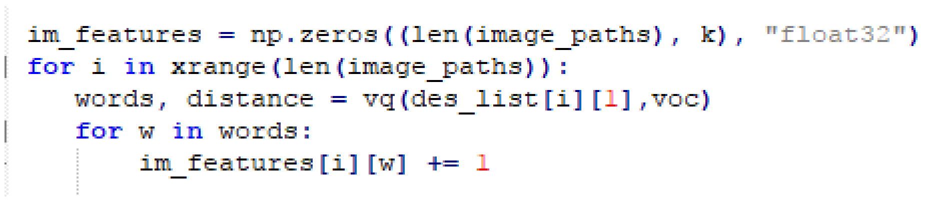
```
descriptors = des_list[0][1]
for image_path, descriptor in des_list[1:]:
    descriptors = np.vstack((descriptors, descriptor))
```

9) Perform k-means clustering.



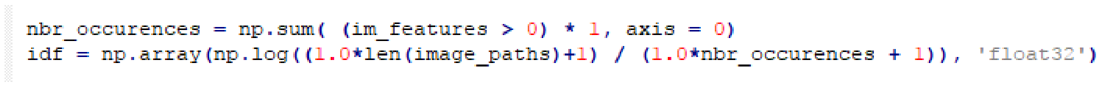
```
k = 1000
voc, variance = kmeans(descriptors, k, 1)
```

10) Calculate the histogram of features.



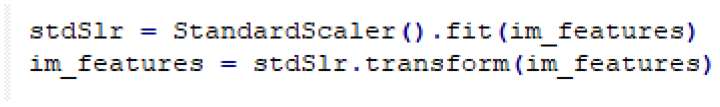
```
im_features = np.zeros((len(image_paths), k), "float32")
for i in xrange(len(image_paths)):
    words, distance = vq(des_list[i][1],voc)
    for w in words:
        im_features[i][w] += 1
```

11) Perform Tf-Idf vectorization.



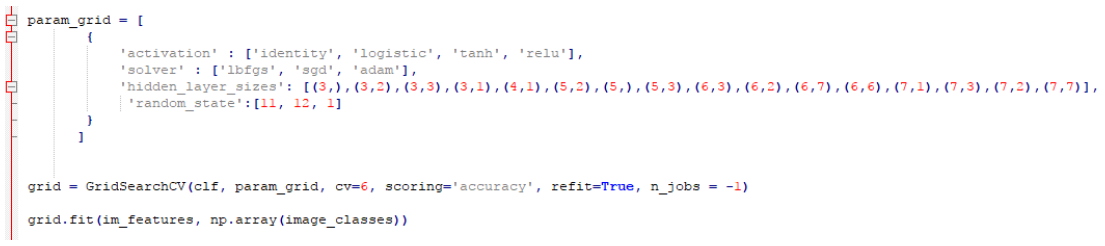
```
nbr_occurences = np.sum( (im_features > 0) * 1, axis = 0)
idf = np.array(np.log((1.0*len(image_paths)+1) / (1.0*nbr_occurences + 1)), 'float32')
```

12) Scaling the words.



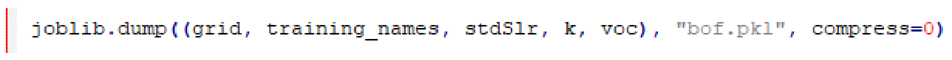
```
stdSlr = StandardScaler().fit(im_features)
im_features = stdSlr.transform(im_features)
```

13) Try all the possible parameters for (activation, solver, hidden_layer_sizes and random_state) to get the fit parameter grid.



```
param_grid = [
        {
            'activation' : ['identity', 'logistic', 'tanh', 'relu'],
            'solver' : ['lbfgs', 'sgd', 'adam'],
            'hidden_layer_sizes': [(3,),(3,2),(3,3),(3,1),(4,1),(5,2),(5,),(5,3),(6,3),(6,2),(6,7),(6,6),(7,1),(7,3),(7,2),(7,7)],
             'random_state':[11, 12, 1]
        }
       ]


grid = GridSearchCV(clf, param_grid, cv=6, scoring='accuracy', refit=True, n_jobs = -1)

grid.fit(im_features, np.array(image_classes))
```

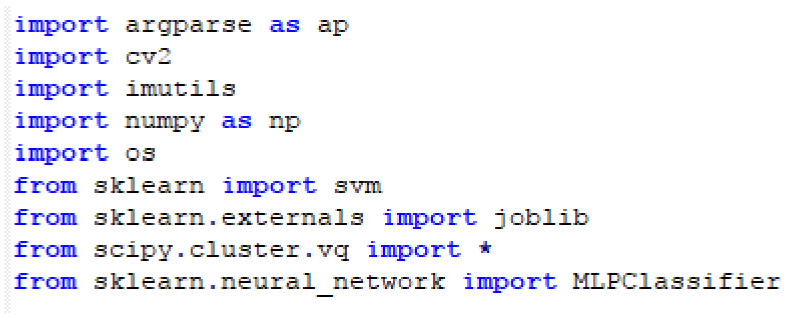
14) Save the MLP model as "bof.pkl".



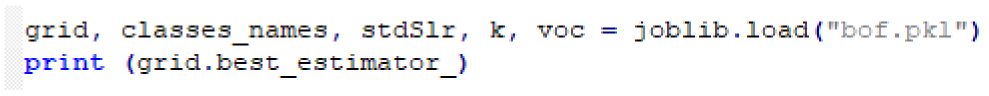
```
joblib.dump((grid, training_names, stdSlr, k, voc), "bof.pkl", compress=0)
```

Testing code.

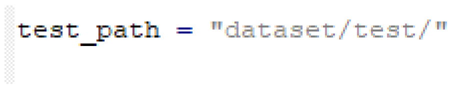
1) Import libraries.


```
import argparse as ap
import cv2
import imutils
import numpy as np
import os
from sklearn import svm
from sklearn.externals import joblib
from scipy.cluster.vq import *
from sklearn.neural_network import MLPClassifier
```

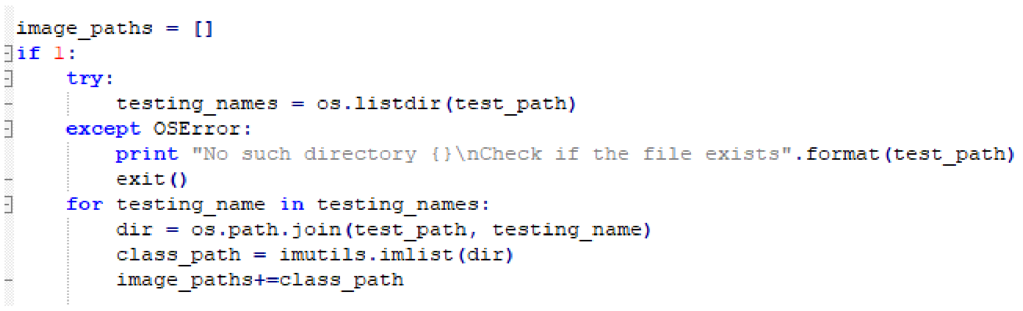
2) Load the classifier, class names, scale, number of clusters and vocabulary for MLP.


```
grid, classes_names, stdSlr, k, voc = joblib.load("bof.pkl")
print (grid.best_estimator_)
```

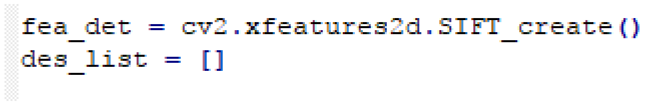
3) Get the path of the testing set.


```
test_path = "dataset/test/"
```

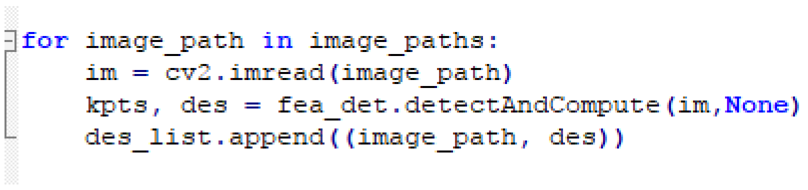
4) Get the path of the testing image(s) and store them in a list.


```
image_paths = []
if 1:
    try:
        testing_names = os.listdir(test_path)
    except OSError:
        print "No such directory {}\nCheck if the file exists".format(test_path)
        exit()
    for testing_name in testing_names:
        dir = os.path.join(test_path, testing_name)
        class_path = imutils.imlist(dir)
        image_paths+=class_path
```

5) Create feature extraction and key point detector objects and list where all the descriptors are stored.

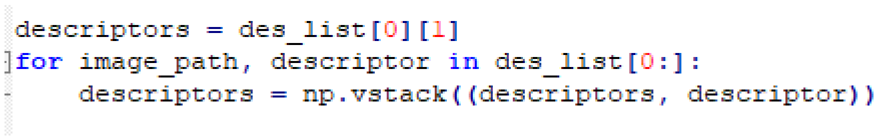

```
fea_det = cv2.xfeatures2d.SIFT_create()
des_list = []
```

6) List all images with their descriptor.



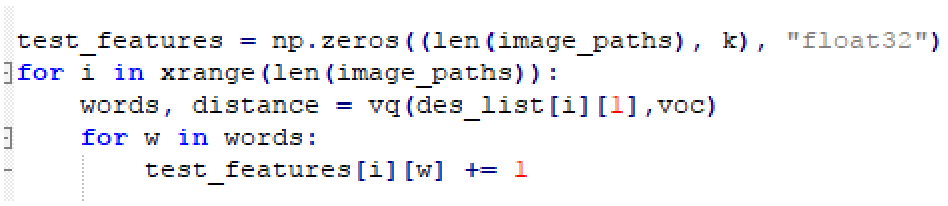
```
for image_path in image_paths:
    im = cv2.imread(image_path)
    kpts, des = fea_det.detectAndCompute(im,None)
    des_list.append((image_path, des))
```

7) Stack all the descriptors vertically in a numpy array.



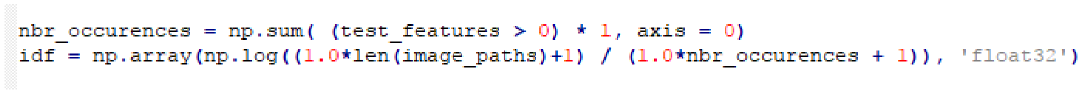
```
descriptors = des_list[0][1]
for image_path, descriptor in des_list[0:]:
    descriptors = np.vstack((descriptors, descriptor))
```

8) Test features.



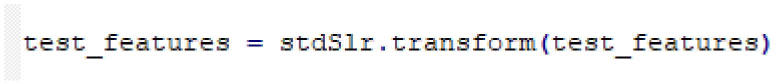
```
test_features = np.zeros((len(image_paths), k), "float32")
for i in xrange(len(image_paths)):
    words, distance = vq(des_list[i][1],voc)
    for w in words:
        test_features[i][w] += 1
```

9) Perform Tf-Idf vectorization.



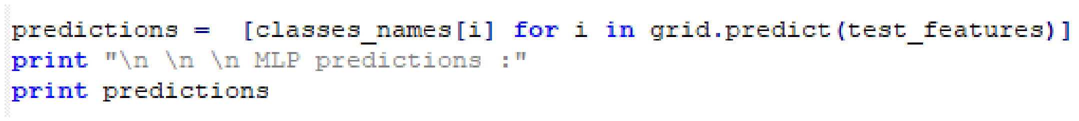
```
nbr_occurences = np.sum( (test_features > 0) * 1, axis = 0)
idf = np.array(np.log((1.0*len(image_paths)+1) / (1.0*nbr_occurences + 1)), 'float32')
```

10) Scale the features.



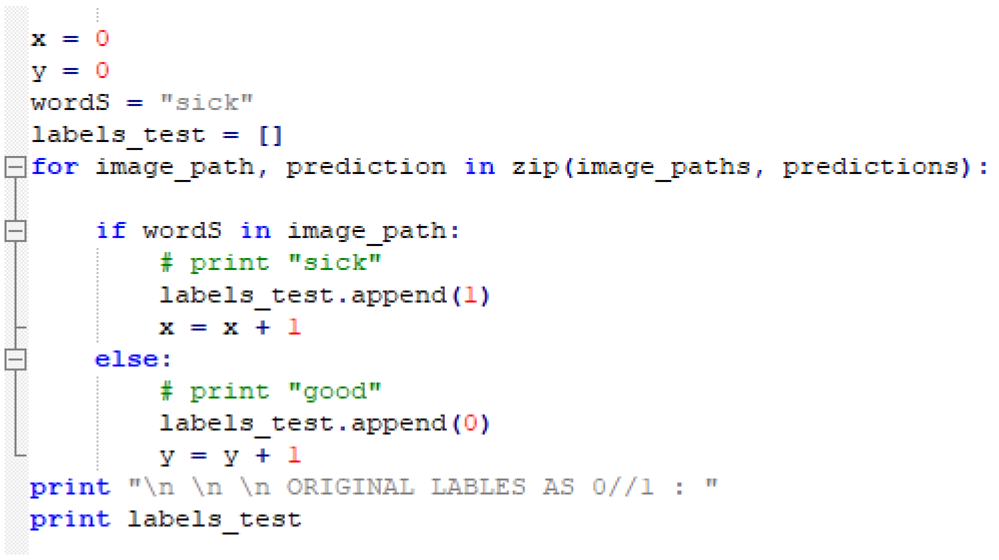
```
test_features = stdSlr.transform(test_features)
```

11) For testing MLP model performance (visualize the results as "good" and "sick").



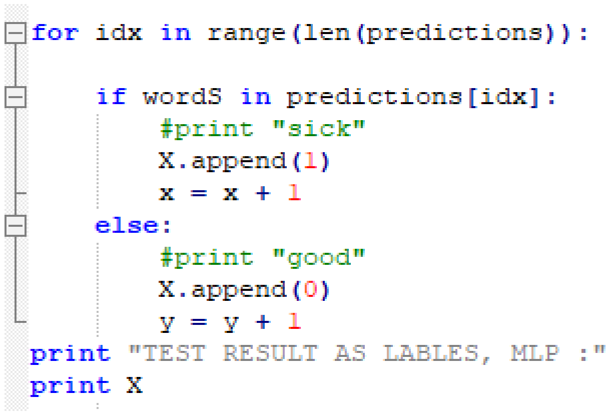
```
predictions =  [classes_names[i] for i in grid.predict(test_features)]
print "\n \n \n MLP predictions :"
print predictions
```

12) Visualize the original labels as 0 (for good) and 1 (for sick).



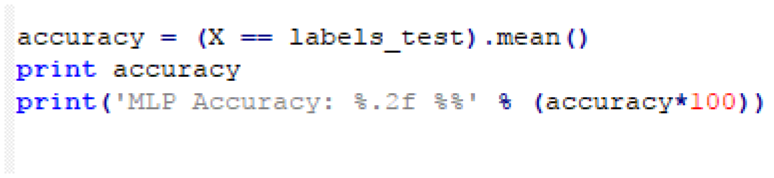
```
x = 0
y = 0
wordS = "sick"
labels_test = []
for image_path, prediction in zip(image_paths, predictions):

    if wordS in image_path:
        # print "sick"
        labels_test.append(1)
        x = x + 1
    else:
        # print "good"
        labels_test.append(0)
        y = y + 1
print "\n \n \n ORIGINAL LABLES AS 0//1 : "
print labels_test
```

13) Visualize the test results.



```
for idx in range(len(predictions)):

    if wordS in predictions[idx]:
        #print "sick"
        X.append(1)
        x = x + 1
    else:
        #print "good"
        X.append(0)
        y = y + 1
print "TEST RESULT AS LABLES, MLP :"
print X
```

14) Calculate the accuracy.



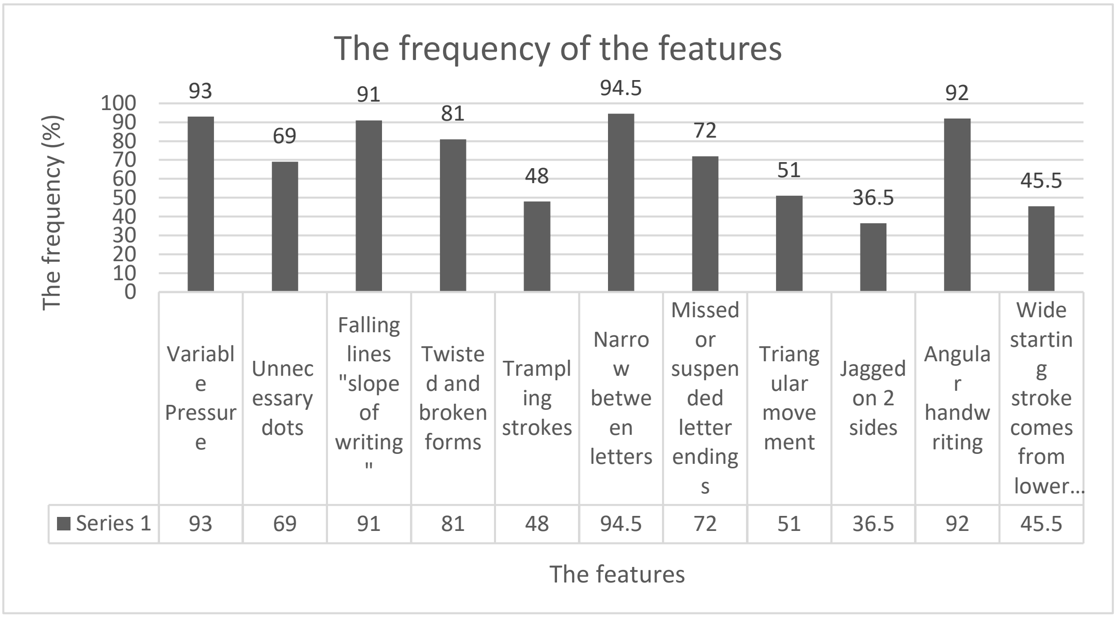
```
accuracy = (X == labels_test).mean()
print accuracy
print('MLP Accuracy: %.2f %%' % (accuracy*100))
```

Results

Features extracting from patients' samples

The next table and graph shows the frequency of each feature in the patients' handwriting samples:

Feature	Frequency	Percent
Variable Pressure	186	93%
Unnecessary dots	138	69%
Falling lines "slope of writing"	182	91%
Twisted and broken forms	162	81%
Trampling strokes	96	48%
Narrow between letters	189	94.5%
Missed or suspended letter endings	144	72%
Triangular movement	102	51%
Jagged on 2 sides	73	36.5%
Angular handwriting	184	92%
Wide starting stroke comes from lower zone to the left	91	45.5%



	Variable Pressure	Unnecessary dots	Falling lines "slope of writing"	Twisted and broken forms	Trampling strokes	Narrow between letters	Missed or suspended letter endings	Triangular movement	Jagged on 2 sides	Angular handwriting	Wide starting stroke comes from lower…
■ Series 1	93	69	91	81	48	94.5	72	51	36.5	92	45.5

It’s obvious from the previous graph that the most frequently appearing features are:

Variable pressure, falling lines, narrow between letters and angular handwriting.

Efficiency of the program

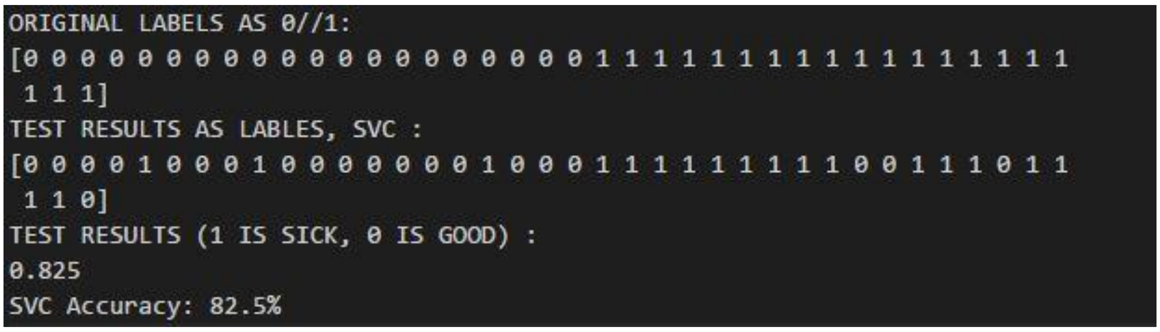
As we mentioned in test plan, we had 2 designs for the program (design 1 and redesign).

Below, are shown the results of the efficiency test of the first design.

Machine learning (supervised algorithm).

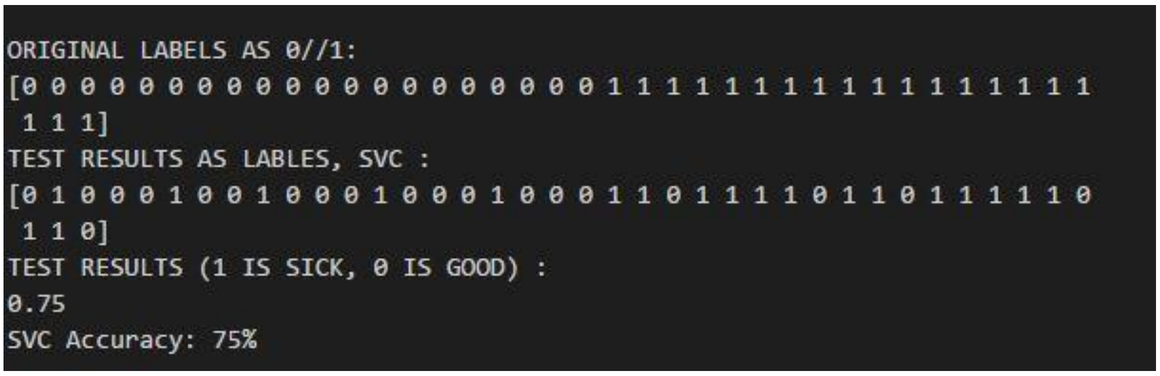
As we mentioned that we divided the testing data into 4 sub-groups, the results of each sub-group are shown below.

The first sub-group:



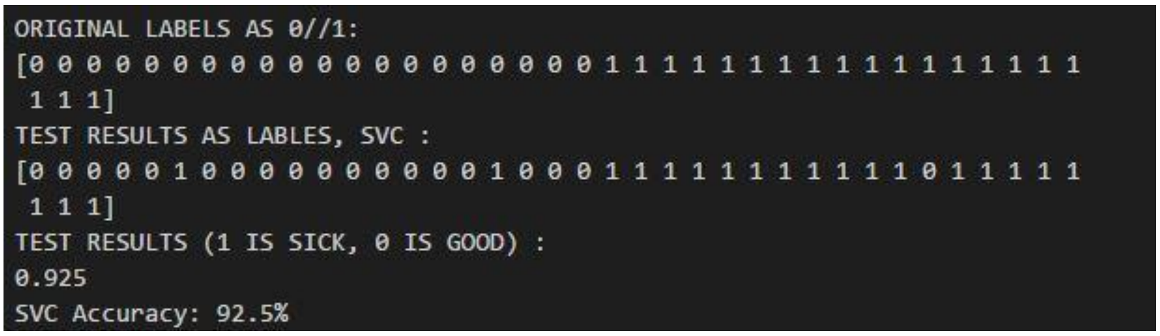
```
ORIGINAL LABELS AS 0//1:
[0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1]
TEST RESULTS AS LABLES, SVC :
[0 0 0 0 1 0 0 0 1 0 0 0 0 0 0 0 1 0 0 0 1 1 1 1 1 1 1 1 1 0 0 1 1 1 0 1 1
 1 1 0]
TEST RESULTS (1 IS SICK, 0 IS GOOD) :
0.825
SVC Accuracy: 82.5%
```

The second sub-group:



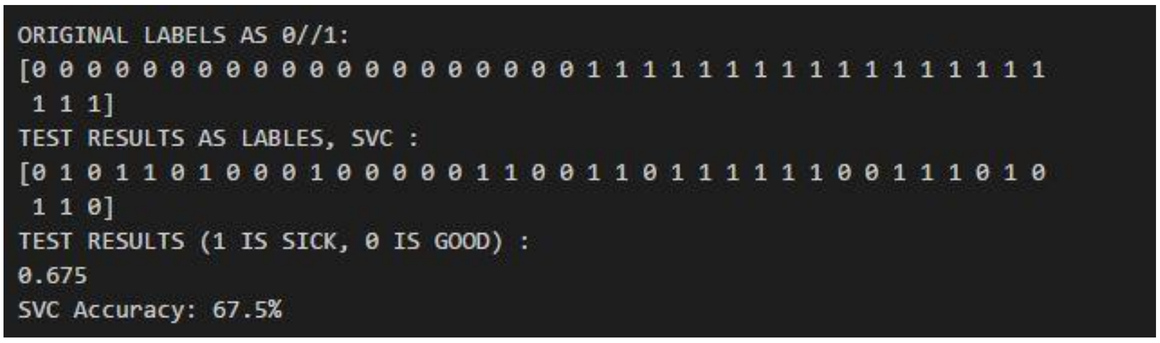
```
ORIGINAL LABELS AS 0//1:
[0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1]
TEST RESULTS AS LABLES, SVC :
[0 1 0 0 0 1 0 0 1 0 0 0 1 0 0 0 1 0 0 0 1 1 0 1 1 1 1 0 1 1 0 1 1 1 1 1 0
 1 1 0]
TEST RESULTS (1 IS SICK, 0 IS GOOD) :
0.75
SVC Accuracy: 75%
```

The third sub-group:



```
ORIGINAL LABELS AS 0//1:
[0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1]
TEST RESULTS AS LABLES, SVC :
[0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 1 0 0 0 1 1 1 1 1 1 1 1 1 1 1 0 1 1 1 1 1
 1 1 1]
TEST RESULTS (1 IS SICK, 0 IS GOOD) :
0.925
SVC Accuracy: 92.5%
```

The forth sub-group:



```
ORIGINAL LABELS AS 0//1:
[0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1]
TEST RESULTS AS LABLES, SVC :
[0 1 0 1 1 0 1 0 0 0 1 0 0 0 0 0 1 1 0 0 1 1 0 1 1 1 1 1 1 0 0 1 1 1 0 1 0
 1 1 0]
TEST RESULTS (1 IS SICK, 0 IS GOOD) :
0.675
SVC Accuracy: 67.5%
```

The next table summarizes the resulted efficiencies from the 4 sub-groups:

Sub-group	Group 1	Group 2	Group 3	Group 4

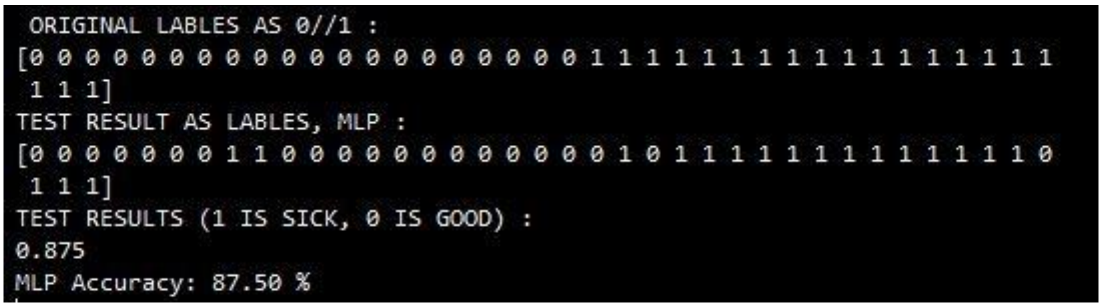
Number of matching samples	33	30	37	27
Efficiency	82.5%	75%	92.5%	67.5%

From the previous table, the average efficiency resulted from testing samples after machine learning was = Summation of efficiencies / number of efficiencies = 79.4 %

It wasn’t satisfying, so we redesigned the program using deep learning (unsupervised algorithm).

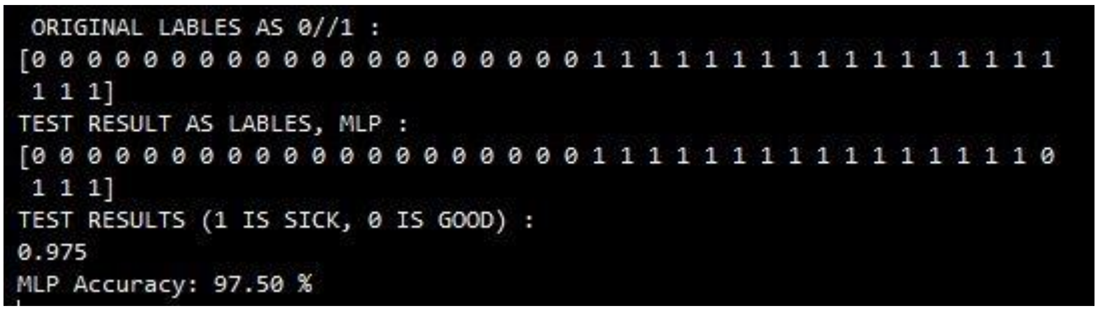
Here are the results of the second design (Deep Learning).

The first sub-group results:



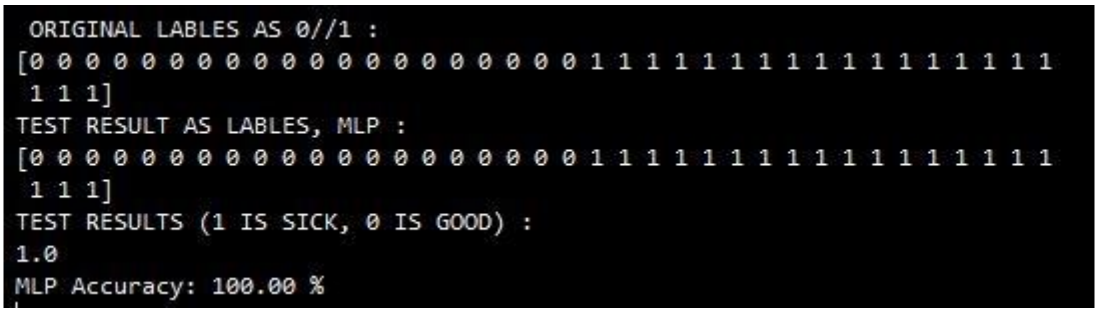
```
 ORIGINAL LABLES AS 0//1 :
[0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1]
TEST RESULT AS LABLES, MLP :
[0 0 0 0 0 0 0 1 1 0 0 0 0 0 0 0 0 0 0 0 0 1 0 1 1 1 1 1 1 1 1 1 1 1 1 1 0
 1 1 1]
TEST RESULTS (1 IS SICK, 0 IS GOOD) :
0.875
MLP Accuracy: 87.50 %
```

The second sub-group results:



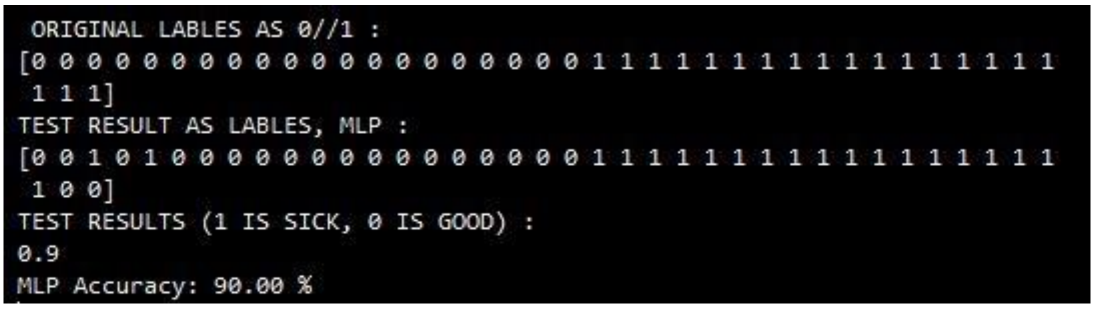
```
 ORIGINAL LABLES AS 0//1 :
[0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1]
TEST RESULT AS LABLES, MLP :
[0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0
 1 1 1]
TEST RESULTS (1 IS SICK, 0 IS GOOD) :
0.975
MLP Accuracy: 97.50 %
```

The third sub-group results:



```
 ORIGINAL LABLES AS 0//1 :
[0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1]
TEST RESULT AS LABLES, MLP :
[0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1]
TEST RESULTS (1 IS SICK, 0 IS GOOD) :
1.0
MLP Accuracy: 100.00 %
```

The forth sub-group results:



```
 ORIGINAL LABLES AS 0//1 :
[0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1]
TEST RESULT AS LABLES, MLP :
[0 0 1 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 0 0]
TEST RESULTS (1 IS SICK, 0 IS GOOD) :
0.9
MLP Accuracy: 90.00 %
```

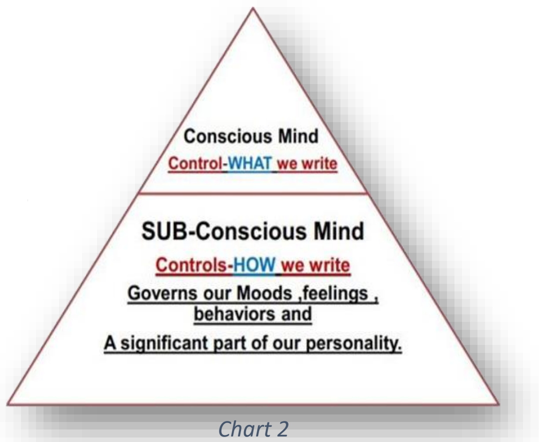
The next table summarizes the resulted efficiencies from the 4 sub-groups.

Sub-group	**Group 1**	**Group 2**	**Group 3**	**Group 4**
Number of matching samples	35	39	40	36
Efficiency	87.5%	97.5%	100%	90%

According to the previous table, the average efficiency was reached by the second design was 93.75%.

Discussion

Changing your handwriting means changing your life, in addition, what we write is controlled by the conscious mind, but how we write is controlled with sub-conscious mind. Cancer -or tumor- makes pressure on neurons that affects motor nervous system, controller of handwriting, which causes some dramatic changes in the patient's handwriting. Hence, we worked on early detection of cancer through handwriting.


Conscious Mind
Control-WHAT we write
SUB-Conscious Mind
Controls-HOW we write
Governs our Moods ,feelings , behaviors and
A significant part of our personality.
Chart 2

In the previous tries to early detect cancer, graphologist used their naked eye to detect the features, accordingly, they need a lot of time, effort, and money. Since nothing had escaped the information technology revolution especially in health, so our detection method was made using a program which saves time, money and lives.

At first, the program was trained on specific features using "Supervised Machine Learning". After testing the program, the efficiency didn't meet our target. That happen because for designing the program in supervised machine learning we depended on just four features which made the detection process more difficult and decreased the ability of the program to distinguish the patients from the healthy ones.

That's why we redesigned the program using "Deep Learning" which extracts all the features automatically from all the samples and depend on them in the training instead of train the program limited number of features. As a result, the efficiency of the program increased. The reached efficiency in the final test was 93.5%. Assuredly, this efficiency declares the high accuracy of the program to analyze the foreign samples and display the true status of the person. Consequently, the program has achieved our main objective.

Recommendation

Future researchers can focus on:

- ✓ Determining the type of cancer. It is important in order to facilitate the examination process for the doctor and increase the chance for relieving. As the handwriting technique is an indicator of the existence of cancer, it won't be as efficient -for the checkup to indicate the existence of cancer- as narrowing the area of illness.
- ✓ Converting the program into a mobile app to increase its availability.
- ✓ Make an electronic pen that has a sensor that will be able to display the writing directly on the screen instead of scanning the text, sending it to the computer and uploading it to the program.
- ✓ Familiarize the program with Arabic language to be available to those who cannot use English.

Conclusion

Changing your handwriting means changing your life, in addition, what we write is controlled by the conscious mind, but how we write is controlled with sub-conscious mind. Cancer –or tumor- makes pressure on neurons that affects motor nervous system, controller of handwriting, which causes some dramatic changes in the patient's handwriting. Hence, we worked on early detection of cancer through handwriting. In the previous trials to detect cancer early, graphologist used their naked eye to detect the features, accordingly, they need a lot of time, effort, and money. Since nothing had escaped the information technology revolution especially in health, so our detection method was made using a program which saves time, money and lives. At first, the program was trained on specific features using "Machine Learning". After testing the program, the efficiency didn't meet our target. That's why we redesigned the program using "Deep Learning" which extracts the features automatically. As a result, the efficiency of the program increased. The reached efficiency in the final test was 93.75%. Assuredly, this efficiency declares the high accuracy of the program to analyze the foreign samples and display the true status of the person. Consequently, the program has achieved our main objective.

References

1) Cancer. (n.d.). Retrieved October 4, 2017, from http://www.who.int/mediacentre/factsheets/fs297/ar/

2) Cancer. (n.d.). Retrieved October 4, 2017, from http://www.who.int/mediacentre/factsheets/fs297/en/

3) OpenCV: Contour Features. (n.d.). Retrieved October 20, 2017, from https://docs.opencv.org/3.1.0/dd/d49/tutorial_py_contour_features.html

4) Morphological Transformations. (n.d.). Retrieved October 21, 2017, from https://docs.opencv.org/3.0-beta/doc/py_tutorials/py_imgproc/py_morphological_ops/py_morphological_ops.html

5) Support vector machine (Svm classifier) implemenation in python with Scikit-learn. (2017, February 19). Retrieved November 7, 2017, from http://dataaspirant.com/2017/01/25/svm-classifier-implemenation-python-scikit-learn/

6) Statistical learning: the setting and the estimator object in scikit-learn. (n.d.). Retrieved November 19, 2017, from http://scikit-learn.org/stable/tutorial/statistical_inference/settings.html

7) Supervised learning: predicting an output variable from high-dimensional observations. (n.d.). Retrieved November 20, 2017, from http://scikit-learn.org/stable/tutorial/statistical_inference/supervised_learning.html

8) Plot different SVM classifiers in the iris dataset. (n.d.). Retrieved November 20, 2017, from http://scikit-learn.org/stable/auto_examples/svm/plot_iris.html#sphx-glr-auto-examples-svm-plot-iris-py

9) Logistic function. (n.d.). Retrieved November 21, 2017, from http://scikit-learn.org/stable/auto_examples/linear_model/plot_logistic.html

10) Model selection: choosing estimators and their parameters. (n.d.). Retrieved November 22, 2017, from http://scikitlearn.org/stable/tutorial/statistical_inference/model_selection.html

11) Unsupervised learning: seeking representations of the data. (n.d.). Retrieved November 24, 2017, from http://scikitlearn.org/stable/tutorial/statistical_inference/unsupervised_learning.html

12) Sklearn.model_selection.train_test_split. (n.d.). Retrieved November 24, 2017, from http://scikitlearn.org/stable/modules/generated/sklearn.model_selection.train_test_split.html

13) Regression vs Classification. (n.d.). Retrieved November 20, 2017, from https://math.stackexchange.com/questions/141381/regression-vs-classification

14) 1.17. Neural network models (supervised). (n.d.). Retrieved November 25, 2017, from http://scikit-learn.org/stable/modules/neural_networks_supervised.html

15) Compare Stochastic learning strategies for MLPClassifier. (n.d.). Retrieved November 25, 2017, from http://scikitlearn.org/stable/auto_examples/neural_networks/plot_mlp_training_curves.html#sphx-glr-auto-examples-neural-networks-plot-mlp-training-curves-py

16) Visualization of MLP weights on MNIST. (n.d.). Retrieved November 26, 2017, from http://scikitlearn.org/stable/auto_examples/neural_networks/plot_mnist_filters.html#sphx-glr-auto-examples-neural-networks-plot-mnist-filters-py

17) Varying regularization in Multi-layer Perceptron. (n.d.). Retrieved November 26, 2017, from http://scikitlearn.org/stable/auto_examples/neural_networks/plot_mlp_alpha.html#sphx-glr-auto-examples-neural-networks-plot-mlp-alpha-py

18) Convolutional Neural Networks - CS231n Convolutional Neural Networks for visual recognition . (n.d.). Retrieved November 27, 2017, from http://cs231n.github.io/convolutional-networks/

19) How to Tune Algorithm Parameters with Scikit-Learn. (2017, January 03). Retrieved November 27, 2017, from https://machinelearningmastery.com/how-to-tune-algorithm-parameters-with-scikit-learn/

20) Sklearn.model_selection.GridSearchCV. (n.d.). Retrieved November 28, 2017, from http://scikitlearn.org/stable/modules/generated/sklearn.model_selection.GridSearchCV.html

21) Introduction to SIFT (Scale-Invariant Feature Transform). (n.d.). Retrieved November 27, 2017, from https://docs.opencv.org/3.3.1/da/df5/tutorial_py_sift_intro.html

22) VBoW Pt 2 - Image Classification in Python with Visual Bag of Words (VBoW) was published on May 11, 2016. (n.d.). VBoW Pt 2 - Image Classification in Python with Visual Bag of Words (VBoW). Retrieved November 28, 2017, from https://ianlondon.github.io/blog/visual-bag-of-words/

23) Bag of Words Models for visual categorization. (2016, April 05). Retrieved November 29, 2017, from https://gilscvblog.com/2013/08/23/bag-of-words-models-for-visual-categorization/

24) B. (2015, July 15). Bikz05/bag-of-words. Retrieved November 29, 2017, from https://github.com/bikz05/bag-of-words

25) T. (2013, March 13). Let's Learn Python - Basics #1 of 8 - Integers, Floats and Maths. Retrieved October 14, 2017, from https://www.youtube.com/watch?v=D48iCw3WWpI&list=PL82YdDfxhWsDJTq5f0Ae7M7yGcA26wevJ

26) Introduction to OpenCV-Python Tutorials. (n.d.). Retrieved October 17, 2017, from http://opencv-python-tutroals.readthedocs.io/en/latest/py_tutorials/py_setup/py_intro/py_intro.html

27) Getting Started with Images. (n.d.). Retrieved October 17, 2017, from http://opencv-python-tutroals.readthedocs.io/en/latest/py_tutorials/py_gui/py_image_display/py_image_display.html

28) Basic Operations on Images. (n.d.). Retrieved October 18, 2017, from http://opencv-python-tutroals.readthedocs.io/en/latest/py_tutorials/py_core/py_basic_ops/py_basic_ops.html

29) Understanding k-Nearest Neighbour. (n.d.). Retrieved October 20, 2017, from http://opencv-python-tutroals.readthedocs.io/en/latest/py_tutorials/py_ml/py_knn/py_knn_understanding/py_knn_understanding.html

30) OCR of Hand-written Data using kNN. (n.d.). Retrieved October 20, 2017, from http://opencv-python-tutroals.readthedocs.io/en/latest/py_tutorials/py_ml/py_knn/py_knn_opencv/py_knn_opencv.html

31) Caring for the Symptoms of Cancer and its Treatment. (2017, November 03). Retrieved December 12, 2017, from http://www.cancer.net/navigating-cancer-care/how-cancer-treated/palliative-care/caring-symptoms-cancer-and-its-treatment

32) Nervous System Side Effects. (2017, May 23). Retrieved December 15, 2017, from http://www.cancer.net/navigating-cancer-care/side-effects/nervous-system-side-effects

33) Palliative Care. (2017, November 02). Retrieved December 10, 2017, from http://www.cancer.net/navigating-cancer-care/how-cancer-treated/palliative-care

34) Peripheral Neuropathy. (2017, June 22). Retrieved December 20, 2017, from http://www.cancer.net/navigating-cancer-care/side-effects/peripheral-neuropathy

35) Nervous system damage and chemotherapy - Canadian Cancer Society. (n.d.). Retrieved October 9, 2017, from http://www.cancer.ca/en/cancer-information/diagnosis-and-treatment/chemotherapy-and-other-drug-therapies/chemotherapy/side-effects-of-chemotherapy/nervous-system-damage-and-chemotherapy/?region=on

36) Peripheral neuropathy. (n.d.). Retrieved October 6, 2017, from http://www.macmillan.org.uk/information-and-support/coping/side-effects-and-symptoms/other-side-effects/peripheral-neuropathy.html

37) Temperton, J. (2017, May 18). This vibrating hair clip could help deaf people sense sounds. Retrieved October 10, 2017, from http://www.wired.co.uk/article/fujitsu-ontenna-deaf-people-vibration-clip

38) Early cancer diagnosis saves lives, cuts treatment costs. (n.d.). Retrieved October 5, 2017, from http://www.who.int/mediacentre/news/releases/2017/early-cancer-costs/en/

39) Home. (n.d.). Retrieved October 9, 2017, from https://horus.tech/?l=en_us

40) Beckett, J. (2017, August 31). Wearable Device for Blind People Could be a Life Changer | NVIDIA Blog. Retrieved October 12, 2017, from https://blogs.nvidia.com/blog/2016/10/27/wearable-device-for-blind-visually-impaired/

41) Pardes, A. (2017, July 20). The Wearables Giving Computer Vision to the Blind. Retrieved October 9, 2017, from https://www.wired.com/story/wearables-for-the-blind/

42) Egyptian cancer resources. (n.d.). Retrieved October 9, 2017, from http://www.cancerindex.org/Egypt

43) Wearable Tech for the Disabled. (2016, December 07). Retrieved October 11, 2017, from http://thirdwavefashion.com/2016/06/wearable-tech-for-the-disabled/

44) Toyota develops wearable device for blind people. (2016, March 08). Retrieved October 11, 2017, from http://www.bbc.com/news/technology-35753978

45) Early cancer diagnosis saves lives, cuts treatment costs. (n.d.). Retrieved October 5, 2017, from http://www.who.int/mediacentre/news/releases/2017/early-cancer-costs/ar/

46) Egypt Health > Diseases Stats. (n.d.). Retrieved October 4, 2017, from http://www.nationmaster.com/country-info/profiles/Egypt/Health/Diseases

47) Egypt. (n.d.). Retrieved October 2, 2017, from http://www.healthdata.org/egypt

48) A Guide to Egypt's Challenges: Healthcare & Hepatitis. (n.d.). Retrieved October 1, 2017, from http://english.ahram.org.eg/NewsContent/1/0/49610/Egypt/Healthcare--Hepatitis.aspx

49) (n.d.). Retrieved September 29, 2017, from https://photius.com/rankings/healthranks.html

50) HEALTH PROFILE EGYPT. (n.d.). Retrieved September 29, 2017, from http://www.worldlifeexpectancy.com/country-health-profile/egypt

51) Neuromuscular complications of cancer. (n.d.). Retrieved October 7, 2017, from https://now.aapmr.org/neuromuscular-complications-of-cancer/

52) Frank Matozza Follow. (2009, March 01). Handwriting analysis in cancer patients. Retrieved October 20, 2017, from https://www.slideshare.net/frankmatozza/HRA-ADOBE-english-presentation

Acknowledgement

We want to show our gratitude to everyone in STEM staff for their continuous support and guidance in our project especially: Mr. Fahmi Farag (Capstone Leader), Mr. Maytham El-Desouky (Capstone Teacher. Besides that, we want to present special thanks to “Eng.Ahmed Hossam” for his time and effort, Cancer hospitals (“57357 Tanta branch”, “Oncology Center Mansoura University”, “Oncology Center Tanta University”, “Tanta Cancer Center”) for their help in taking the samples from the patients.