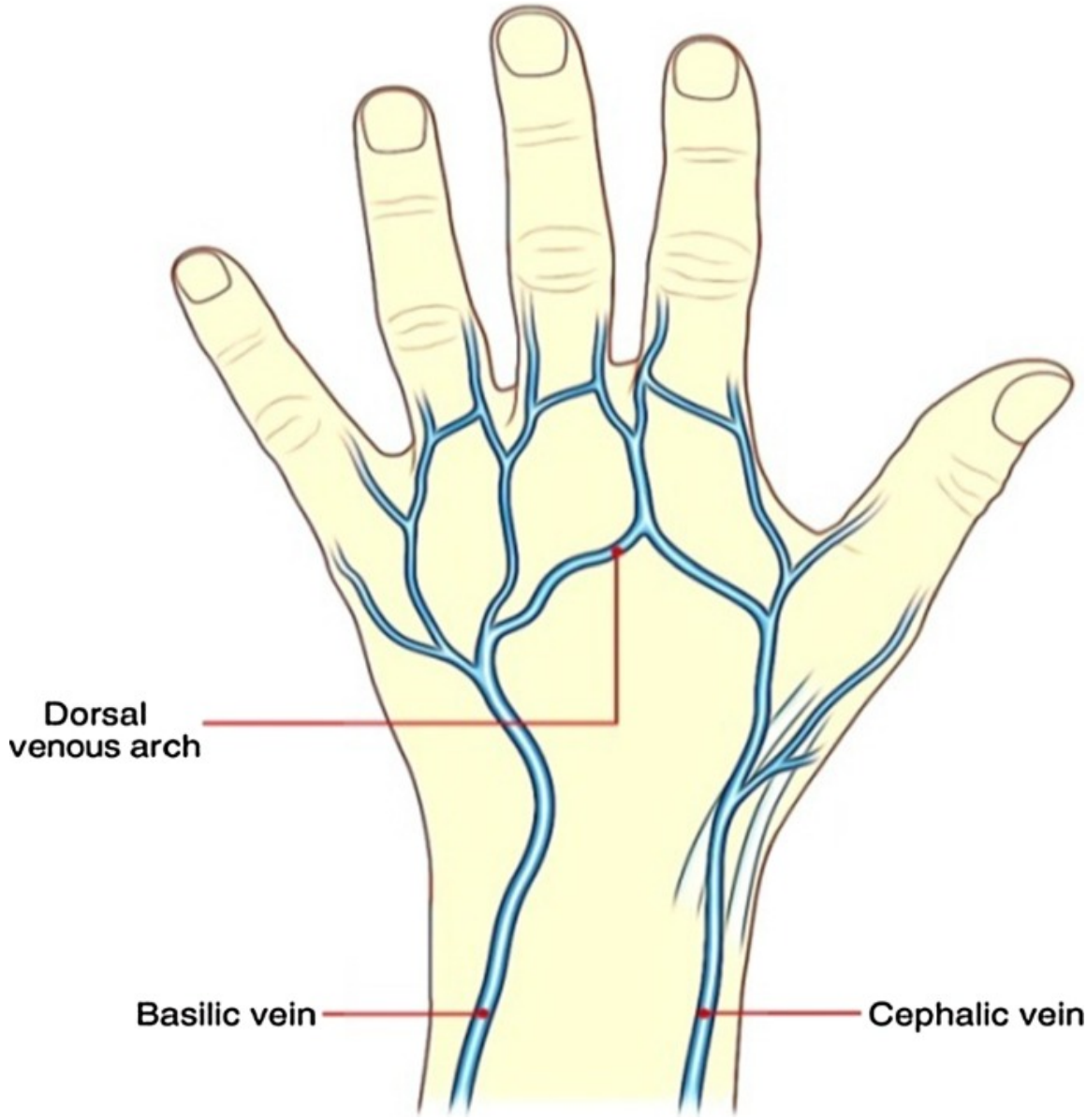
BCI3006 - BIOMETRICS (G2 SLOT - TEAM 3)

VEIN BIOMETRIC RECOGNITION SYSTEM
USING DORSAL VEIN



Introduction

As the security system in society increases, vein recognition has become a precise method for biometrics-based identification systems. Biometrics is a field of statistical analysis of the biological characteristics of an individual. Here, the dorsal hand vein is used for biometric identification.

The dorsal hand vein biometric system is one of the emerging biometric technologies to determine or authenticate an individual's identity.

The dorsal hand vein biometric is an emerging biometric due to its uniqueness. Since the vein pattern is a series of blood vessels placed right under the person's hand, it is tough to make unauthorised alterations.

Member wise contribution

Mrinal Sharma (20BCI0247)

After the completion of feature extraction, a machine learning model has been written. This is a CNN Model. This is a typical ML model which has been trained and tested using Kernel Fischer Analysis. Using KFA, we generate the training data randomly and also generate the training data manually. Prototype:

```
model = perform_kfa_PhD(X, ids, kernel_type, kernel_args,n);
```

After training the ML model, I proceeded towards the testing of the model. Then I tested and used euclidean distance to check the error. If the Euclidean distance is less than 100, access will not be granted to the person, else access will be granted.

To obtain cancelability, I applied template transformation on the features using the XOR-based technique. The XOR Function-based technique is applied on extracted dorsal veins features by the following steps:-

1) Let Fv is the original features of dimension [100,100]

2) I generated a random grid(RG) of the dimension [100,100] the same size as the original feature.

3) Performed bitwise XOR operation between original feature and RG

4) Performed median filter on distorted features, to provide non-invertibility.

Rohan Gupta (20BCI0260)

Created the function minmax, which normalizes the input data using a min-max normalization technique.

In this code, along with the basic CNN model, the Bayesian Optimization Technique is also used, which is an approach that uses the Bayes Theorem to direct the search in order to find the minimum and maximum of an objective function. It is an approach that is mostly used for objective functions that are complex to evaluate.

Then using the append() function, I stored the path of all the sample vein patterns into a list.

Along with this, I researched a few research papers related to vein biometric systems and the Bayesian Optimization Technique.

Research papers referred:

- M. El-Ghandour, M. I. Obayya, B. Yousef and N. F. Areed, "Palm vein Recognition Using Block-Based WLD Histogram of Gabor Feature Maps and Deep Neural Network With Bayesian Optimization," in IEEE Access, vol. 9, pp. 97337-97353, 2021, doi: 10.1109/ACCESS.2021.3093343.

Harsh Rajpal (20BCI0271)

I helped in writing the code for loading the weight matrix. Coded to read and generate a key matrix for matching with input data. In the pre-processing state on dorsal skin, I built a function to load sample data images and clear all noise like hair. Defined a function to extract all required features and filled it in a matrix that corresponded to the key matrix.

I investigated numerous strategies for loading and removing loaded data. Various good qualities of extracted data were researched and extracted as though it should be effortlessly feasible.

Extraction methods such as second-order statistics (line, edge detection) that include the local maximum curvature feature detector, line detection, and first-order statistics such as thresholding operations.

The extraction of features involves the extraction of vein centre locations, the connection of vein centre positions, picture labelling, and vein centre connection.

Kenil Patel (20BCI0277)

Contributed to code for loading the weight matrix. Declared variables for rows and columns, then defined KeyMatrix in which I appended our 'key.csv' files which stored our data and then I converted the matrix into a NumPy array and then normalised the key matrix which I appended to a list.

```
for row in reader:
        for i in range(0, len(row)):
                row[i] = float(row[i])
        Keymatrix.append(row)
Keymatrix = np.array(Keymatrix)
Keymatrix = normalize(Keymatrix)
Keymatrix1 = []
```

(Appending DATA stored in CSV files to an array and then normalizing key matrix)
I defined another keymatrix1 in which I append our 'key2.csv' files. then I converted to a NumPy array and then normalised keymatrix1 which I appended to a list. Hence now both the matrices are loaded.

Reviewed various research papers for the project :

- Singh, Sanchit & Ramalho, Mauricio & Correia, Paulo & Soares, Luís. (2011). Biometric identification through the palm and dorsal hand vein patterns. EUROCON 2011 - International Conference on Computer as a Tool - Joint with Conftele 2011. 1-4. 10.1109/EUROCON.2011.5929297.
- Wei Jia, Wei Xia, Bob Zhang, Yang Zhao, Lunke Fei, Wenxiong Kang, Di Huang, Guodong Guo, A survey on dorsal hand vein biometrics, Pattern Recognition, Volume 120,2021,108122,ISSN0031-3203,https://doi.org/10.1016/j.patcog.2021.108122.(https://www.sciencedirect.com/science/article/pii/S0031320321003095)

Code:

```
print('\n')
print('Load images from a database and compute it\'s features.')
print('During this we also transform feature using random distance.')
print('This may take a while.')

import csv # to csv -> to read the given data
import os # -> to maintain the paths of the files
import warnings # To handle so the the program will be error free
import cv2 # image processing
import numpy as np # we can use list in python but as its a complex -> np arrays for fast
from scipy.spatial import distance # e, h, -> which is used while calculating the errors
from sklearn.preprocessing import normalize # to normalize the given data for more accuracy
from cancellability.feature_rdmtransform import transformMeximumCurvatureRDM as MCRG # part algo
from classification import KFA # algo
from classification import Projection #algo

warnings.filterwarnings("ignore")

# normalizing the input data using a minmax normalization tech...
def minmax(X, low, high, minX=None, maxX=None, dtype=np.float):
    X = np.asarray(X)
    if minX is None:
        minX = np.min(X)
    if maxX is None:
        maxX = np.max(X)

    X = X - float(minX)
    X = X / float((maxX - minX))
    return np.asarray(X, dtype=dtype)
```

```
input_folder_path = './sample dataset/veinpattern/'
input_folder_content = (os.listdir(input_folder_path))
print("Dataset contain", input_folder_content)
NumberOfSub = len(input_folder_content)
print("Number of input subfolder: ", NumberOfSub)
S_Path = []
for subfolderNo in range(0, NumberOfSub):
    S_Path.append(os.path.join(input_folder_path, input_folder_content[subfolderNo]))

sub_folder_content = []

for subfolderNo in range(0, NumberOfSub):
    sub_folder_content.append(os.listdir(S_Path[subfolderNo]))
```

```
# loading the weight matrix
R = len(sub_folder_content)
C = len(sub_folder_content[0])
# R = 5
# C = 3
k = 100
N = 100
Keymatrix = []

with open('./create_csv/Key01.csv') as File:
    reader = csv.reader(File, delimiter=',', quotechar=',', quoting=csv.QUOTE_MINIMAL)
    for row in reader:
        for i in range(0, len(row)):
            row[i] = float(row[i])
        Keymatrix.append(row)

print('Key Matrix Loaded')
Keymatrix = np.array(Keymatrix)
Keymatrix = normalize(Keymatrix)
Keymatrix1 = []

with open('./create_csv/Key02.csv') as File:
    reader = csv.reader(File, delimiter=',', quotechar=',', quoting=csv.QUOTE_MINIMAL)
    for row in reader:
        for i in range(0, len(row)):
            row[i] = float(row[i])
        Keymatrix1.append(row)

print('Key Matrix 2 Loaded')
Keymatrix1 = np.array(Keymatrix1)
Keymatrix1 = normalize(Keymatrix1)
```

```
#Importing data in the program and extracting features
# images using cv2 # from opencv
fvs3 = []

ID = []
for x in range(0, R):
    print('Subject: ', x + 1)
    for y in range(0, C):
        ID.append(x)
        ImgPath = S_Path[x] + '\\' + sub_folder_content[x][y]
        Img = cv2.imread(ImgPath, 0)
        Img = np.asarray(Img, dtype=np.float64)
        Img = cv2.resize(Img, (k, N), interpolation=cv2.INTER_CUBIC)
        Key = Keymatrix[x]
        Key1 = Keymatrix1[x]
        Key = Key.reshape(k * N, 1)
        Key1 = Key1.reshape(k * N, 1)
        transformedFeatureVector, fvs = MCRG(Img, Key, Key1)
        if x == 0 and y == 0:
            dataMatrixRG = np.column_stack((transformedFeatureVector)).T
        else:
            dataMatrixRG = np.column_stack((dataMatrixRG, transformedFeatureVector))
```

```
print('Feature Extraction completed')
print('============fv train=======================')
data = dataMatrixRG.T
print('feature : ', data)
print('==================================')
ids_train = []
cnt = 0
for i in range(0, R):
    for j in range(0, C):
        ids_train.append(ID[cnt])
        cnt += 1
train_data = ((np.array(data)).T)

model = KFA.perform_kfa_PhD(train_data, ids_train, 'fpp', len(ids_train))
# compiling and traing the model
fvs2 = []
k = 100
N = 100
R1 = 1
C1 = 1
ids_test = 0
for x in range(0, R1):
    for y in range(0, C1):
        ImgPath = S_Path[x] + '\\' + sub_folder_content[x][5]
        Img2 = cv2.imread(ImgPath, 0)
        Img2 = np.asarray(Img2, dtype=float)
        Img2 = cv2.resize(Img2, (k, N), interpolation=cv2.INTER_CUBIC)
        Key = Keymatrix[ids_test]
        Key1 = Keymatrix1[ids_test]
        Key = Key.reshape(k * N, 1)
        Key1 = Key1.reshape(k * N, 1)
        transformedFeatureVector1, fvs1 = MCRG(Img2, Key, Key1)  # (100x100,1)
        if x == 0 and y == 0:
            dataMatrixRG1 = np.column_stack((transformedFeatureVector1)).T
        else:
            dataMatrixRG1 = np.column_stack((dataMatrixRG1, transformedFeatureVector1))

print('Feature Extraction completed')
print('============fv test=======================')
data1 = np.array(dataMatrixRG1).T
print('feature : ', data1)
test_data = ((np.array(data1)).T)
```

```
testfeature = Projection.nonlinear_subspace_projection_PhD(test_data, model)
dt = model.train
print(testfeature) # testing and using euclidean distace to check the error
for x in range(0, len(ids_train)):
    u = np.array(dt[:, x])
    v = np.array(testfeature)
    di = distance.euclidean(v, u)
    print('value of dist=  ', di)
    if di <= 100:
        print('acess granted for id no : ', ids_train[x] + 1)
    else:
        print('sorry: ', ids_train[x] + 1)
```