Universitatea ”Dunărea de Jos” din Galați

Facultatea de Automatică, Calculatoare, Inginerie Electrică și Electronică

**Extragerea datelor biometrice din amprentele digitale**

Student: Hurduza Laurian Sorin

Anul I – Master

Specializarea: Tehnologii Informatice Avansate

**Cuprins**

[Descrierea imaginilor din sursele găsite 3](#_Toc135756621)

[Mediul pentru programare și bibliotecile utilizate 5](#_Toc135756622)

[Descrierea obiectivului proiectului 6](#_Toc135756623)

[Descrierea etapelor proiectului 6](#_Toc135756624)

[Schema generală de lucru 8](#_Toc135756625)

[Descrierea metodelor de procesare 9](#_Toc135756626)

[Clasificarea prin rețele neuronale 11](#_Toc135756627)

[Codul sursă 12](#_Toc135756628)

[Graficul modelului 27](#_Toc135756629)

# Descrierea imaginilor din sursele găsite

Imaginile din sursele găsite reprezintă amprente ale diferitelor persoane, captate cu ajutorul unor senzori specializați de amprentă. Fiecare amprentă are caracteristici unice, care o definesc și este utilizată pentru accesul controlat al persoanelor în diverse instituții, dispozitive etc. Aceste amprente conțin caracteristici biometrice unice, pe care le vom extrage și le vom analiza pe parcursul acestui proiect, în speranța identificării atributelor distinctive.

Imaginile sunt în format .bmp, iar setul de date a fost împărțit în trei secțiuni: setul pentru antrenare, setul pentru validare și setul pentru testare.

Setul pentru antrenare conține 1.155 de imagini cu amprente, care vor fi folosite în procesul de antrenare al modelului. Antrenarea este un pas esențial, deoarece prin acest proces rețeaua neuronală va învăța să clasifice corect diverse tipuri de amprente.

Setul pentru validare cuprinde 734 de imagini cu amprente, care nu vor fi utilizate în procesul de antrenare, ci vor fi rezervate pentru validarea rezultatelor obținute în timpul antrenamentului. Aceste imagini vor servi pentru a evalua capacitatea modelului de a generaliza la date noi și necunoscute.

Setul pentru testare conține 176 de imagini cu amprente, care nu au fost utilizate nici în procesul de antrenare, nici în procesul de validare. Aceste imagini vor fi folosite pentru a testa modelul și pentru a efectua predicții cu privire la clasa în care se încadrează fiecare imagine.

Structura acestui set de date este următoarea:

* dataset
  + test
    - test\_dataset
    - fake\_test\_dataset
  + train
    - original\_train
    - fake\_train
  + validation
    - real
    - fake

”dataset” – reprezintă directorul principal unde se află setul de date.

”test” – reprezintă folderul unde se află clasa cu imaginile pentru testarea modelului

”train” – reprezintă folderul unde se află clasa pentru antrenament

”validation” – reprezintă folderul udne se află clasa pentru validare

Observăm că, în fiecare folder, subfolderele sunt clasele pe care modelul se antrenează. În cazul nostru avem două clase pe care vom antrena modelul: real și fals.

Acesta este setul de date cu care vom lucra în antrenarea modelului nostru, însă pentru acest lucru va trebui să pregătim mediul de programare și să discutăm despre bibliotecile pe care le vom folosi.

# Mediul pentru programare și bibliotecile utilizate

PyCharm este mediul de programare care a fost utilizat pentru scrierea codului responsabil cu antrenarea modelului, acesta se poate descărca de pe site-ul oficial: <http://www.jetbrains.com/pycharm>.

Bibliotecile pe care le vom folosi sunt următoarele:

* **Keras**: Keras este o bibliotecă populară pentru construirea și antrenarea rețelelor neuronale. Oferă o interfață simplă și intuitivă pentru definirea modelelor de rețele neuronale și pentru executarea operațiilor de învățare profundă.
* **NumPy**: NumPy este o bibliotecă pentru Python care adaugă suport pentru lucrul cu matrice și tablouri multidimensionale. Este foarte utilizată în domeniul științei datelor și a învățării automate pentru manipularea și prelucrarea datelor.
* **layers din keras.applications.densenet**: Aceasta este o sub-bibliotecă din Keras care conține implementarea modelului DenseNet, o rețea neurală convoluțională (CNN) pre-antrenată. Aceste straturi de rețea pot fi utilizate pentru construirea și configurarea modelelor personalizate.
* **ImageDataGenerator din keras.preprocessing.image**: ImageDataGenerator este o clasă din Keras care permite generarea de date de imagine augmentate. Aceasta poate fi folosită pentru a prelucra și a augmenta seturile de date de imagini în timpul antrenării rețelei neurale.
* **pyplot din matplotlib**: Pyplot este un modul din biblioteca matplotlib, care oferă funcții pentru crearea de grafice și vizualizări. În codul dat, este utilizat pentru afișarea graficelor rezultatelor.
* **regularizers din tensorflow.python.keras**: Acest modul oferă funcționalitate pentru utilizarea de regularizatori în rețelele neuronale. Regularizatorii pot fi utilizați pentru a controla overfitting-ul în timpul antrenării rețelei.
* **tensorflow**: TensorFlow este o platformă de învățare automată open-source dezvoltată de Google. Este utilizată pentru construirea și antrenarea rețelelor neuronale și pentru realizarea operațiilor de învățare profundă.
* **Image din PIL**: Image este o clasă din biblioteca Python Imaging Library (PIL), care oferă funcționalitate pentru manipularea imaginilor. Este utilizată în codul dat pentru a realiza operații cu imagini.
* **sklearn.ensemble.RandomForestClassifier**: Această bibliotecă implementează algoritmul de pădure aleatorie (random forest) pentru clasificare. Pădurile aleatorii sunt o metodă de învățare în ansamblu (ensemble learning) care combină mai multe arbori de decizie pentru a obține rezultate mai precise și mai stabile în clasificare.
* **sklearn.metrics.accuracy\_score**: Această bibliotecă oferă o funcție pentru calcularea scorului de precizie (accuracy score). Scorul de precizie este o măsură a acurateței clasificatorului și reprezintă raportul dintre numărul de predicții corecte și numărul total de exemple.
* **sklearn.metrics.confusion\_matrix**: Această bibliotecă oferă o funcție pentru calcularea matricei de confuzie (confusion matrix). Matricea de confuzie este o reprezentare a rezultatelor clasificării și arată numărul de exemple clasificate corect și numărul de exemple clasificate greșit în funcție de clasele reale și predicțiile modelului.
* **sklearn.metrics.precision\_score**: Această bibliotecă oferă o funcție pentru calcularea scorului de precizie (precision score). Scorul de precizie este o măsură a proporției de exemple clasificate corect ca fiind pozitive din totalul exemplelor clasificate ca fiind pozitive.
* **sklearn.metrics.recall\_score**: Această bibliotecă oferă o funcție pentru calcularea scorului de recuperare (recall score). Scorul de recuperare este o măsură a proporției de exemple clasificate corect ca fiind pozitive din totalul de exemple pozitive reale.
* **sklearn.metrics.f1\_score**: Această bibliotecă oferă o funcție pentru calcularea scorului F1. Scorul F1 este o măsură a echilibrului între precizie și recuperare și este definit ca media armonică între cele două.
* **sklearn.model\_selection.train\_test\_split**: Această bibliotecă oferă o funcție pentru împărțirea setului de date într-un set de antrenare și un set de testare. Această împărțire este utilă pentru evaluarea performanței modelelor de clasificare, deoarece ne permite să antrenăm modelul pe datele de antrenare și să îl testăm pe datele de testare pentru a obține o estimare a performanței generale.
* **sklearn.preprocessing.LabelEncoder**: Această bibliotecă oferă o clasă pentru codificarea etichetelor categorice într-un format numeric. Acest lucru este util în clasificarea în care algoritmul de învățare automată are nevoie de date numerice pentru a putea fi antrenat și evaluat corect. LabelEncoder transformă etichetele categorice în valori numerice pentru a putea fi utilizate în algoritmul de clasificare.

# Descrierea obiectivului proiectului

Obiectivul proiectului este de a extrage date biometrice din amprentele digitale. Datele biometrice pe care le vom extrage sunt următoarele:

1. Keypoint-uri sau punctele cheie din amprentă care sunt unice de la amprentă la amprentă, iar acestea sunt identificate cu ajutorul algoritmului ORB (Oriented FAST and Rotated BRIEF).
2. Calcularea direcției liniei amprentei în fiecare punct cheie cu ajutorul gradienților.
3. Antrenarea unui model RandomForest care să clasifice datele obținute în două clase: real și fals.

# Descrierea etapelor proiectului

Etapele proiectului sunt următoarele:

1. Preprocesarea imaginilor

Această etapă cuprinde pregătirea setului de date pentru a putea fi utilizat în antrenarea modelului. Preprocesarea este un pas important pentru că de aceasta depinde acuratețea modelului. Ne dorim o acuratețe bună și din acest motiv vom aplica următoarele acțiuni pentru scopul dorit:

* Redimensionarea imaginii care va fi făcută la 80x80 pixeli
* Corecția contrastului
* Eliminarea zgomotului
* Umplerea golurilor din imagine
* Subțierea liniei amprentei
* Detectarea marginilor
* Binarizarea imaginii
* Transformări geometrice
* Segmentarea bazată pe praguri multiple
* Extragerea conturului
* Aplicarea filtrului Gabor
* Aplicarea algoritmului Watershed pentru segmentarea regiunilor de interes

1. Antrenarea modelului

Antrenarea modelului se va face utilizând biblioteca Tensorflow și Keras

1. Testarea modelului

Modelul va fi testat cu ajutorul unei imagini pentru a determina clasa din care face parte, în acest caz avem doar o clasă, deci o va prezice doar pe aceea.

1. Extragerea datelor biometrice

Extragem datele biometrice din amprentele digitale, acestea fiind punctele de interes și direcția liniei amprentei.

1. Antrenarea modelului RandomForest
2. Testarea modelului RandomForest

# Schema generală de lucru

PREPROCESARE

Redimensionare

Corecție contrast

Eliminare zgomot

Umplere goluri

Detectare margini

Binarizare

Transf. geometrice

Subțiere linie

Extragere contur

Segmentare

TESTARE

ANTRENARE

Watershed

Filtru Gabor

EXTRAGERE DATE

ANTRENARE RF

TESTARE RF

# Descrierea metodelor de procesare

* **Redimensionarea imaginii**: Aceasta implică modificarea dimensiunilor imaginii pentru a o face mai mică sau mai mare. În acest caz, imaginea va fi redimensionată la o rezoluție de 80x80 pixeli.
* **Corecția contrastului**: Această operație are ca scop ajustarea diferenței dintre nuanțele de lumină dintr-o imagine, pentru a accentua detalii și a îmbunătăți vizualizarea. Prin corecția contrastului, se poate îmbunătăți claritatea și vizibilitatea amprentei.
* **Eliminarea zgomotului**: Procesul de eliminare a zgomotului constă în reducerea sau eliminarea elementelor nedorite din imagine, cum ar fi pete sau pixeli neuniformi. Acest lucru se face în general prin aplicarea unor filtre de reducere a zgomotului.
* **Umplerea golurilor din imagine**: În cazul în care imaginea are goluri sau spații lipsă, se poate efectua o operație de umplere a acestora. Aceasta implică completarea zonelor lipsă cu informații bazate pe conținutul înconjurător.
* **Subțierea liniei amprentei**: Acest proces se referă la reducerea grosimii liniei amprentei, astfel încât să se obțină o reprezentare mai subțire a acesteia. Acesta poate fi realizat prin aplicarea unor tehnici specifice de procesare a imaginilor.
* **Detectarea marginilor**: Această etapă implică identificarea și evidențierea marginilor sau a contururilor obiectelor din imagine. Acest lucru se poate realiza prin diferite metode, cum ar fi utilizarea algoritmului Canny sau alți algoritmi de detecție a marginilor.
* **Binarizarea imaginii**: Binarizarea constă în transformarea imaginii într-o imagine alb-negru, în care fiecare pixel este atribuit unei culori (alb sau negru) în funcție de un anumit prag sau criteriu. Acest proces poate fi utilizat pentru a evidenția anumite caracteristici sau obiecte în imagine.
* **Transformări geometrice**: Acestea se referă la modificarea poziției, dimensiunilor, orientării sau deformării geometrice a imaginii. Aceste transformări pot include rotații, scalări, translații și inversări.
* **Segmentarea bazată pe praguri multiple**: Segmentarea imaginii implică împărțirea acesteia în regiuni sau obiecte mai mici, bazate pe anumite criterii sau praguri. Segmentarea bazată pe praguri multiple implică utilizarea a mai multor praguri pentru a diferenția diferitele regiuni sau obiecte în imagine.
* **Extragerea conturului**: Această etapă implică evidențierea și extragerea conturului sau a marginilor obiectelor din imagine. Aceasta poate fi realizată prin metode de detectare a marginilor sau prin aplicarea unor filtre specifice.
* **Aplicarea filtrului Gabor**: Filtrul Gabor este un filtru utilizat în procesarea imaginilor, care este folosit în special pentru detectarea texturilor și a contururilor. Acesta este bazat pe o funcție matematică complexă și este aplicat pentru a evidenția anumite caracteristici sau structuri din imagine.
* **Aplicarea algoritmului Watershed** pentru segmentarea regiunilor de interes: Algoritmul Watershed este utilizat pentru segmentarea imaginilor în regiuni sau obiecte, bazându-se pe caracteristicile topografice ale imaginii. Acesta este adesea folosit în cazul segmentării bazate pe regiuni și poate fi aplicat pentru a identifica și separa diferitele regiuni de interes în imagine.

# Clasificarea prin rețele neuronale

Avem un clasificator de imagini bazat pe o rețea neuronală convoluțională (CNN). Modelul utilizează arhitectura Sequential din biblioteca Keras pentru a construi rețeaua neuronală. Este utilizat un set de date preprocesat, care conține directoare pentru antrenare, validare și testare.

Setul de date este împărțit în două clase (binary) și imaginile sunt de dimensiune 80x80 pixeli și sunt în escala de gri (color\_mode='grayscale').

Modelul CNN este compus dintr-o serie de straturi convoluționale și straturi de max pooling, urmate de straturi Dense.

Pentru activarea straturilor convoluționale se utilizează funcții precum 'relu', 'sigmoid' și 'tanh'. Stratul final are o funcție de activare 'relu' și numărul de neuroni corespunde numărului de clase (în cazul nostru, 1 pentru clasificare binară).

Modelul este compilat cu un optimizator 'adam' și funcția de pierdere 'binary\_crossentropy'. Apoi, sunt create generatori de date pentru seturile de antrenare, validare și testare, folosind ImageDataGenerator pentru a redimensiona și scala valorile pixelilor.

După compilare și crearea generatorilor de date, modelul este antrenat cu ajutorul metodei fit() folosind datele de antrenare și validare. Ulterior, modelul este evaluat pe setul de testare pentru a obține pierderea și acuratețea testului.

La final, este afișată o grafică cu evoluția acurateței și pierderii în timpul antrenamentului, iar apoi este prezentat un exemplu de testare a modelului pe o imagine nouă, cu interpretarea rezultatului prin stabilirea clasei prezise în funcție de valoarea de tăiere 0.5.

# Codul sursă

Antrenarea și validarea modelului – train\_keras\_img.py

import itertools  
import os  
import re  
  
import numpy as np  
from matplotlib import pyplot as plt  
from sklearn.ensemble import RandomForestClassifier  
from sklearn.metrics import accuracy\_score, confusion\_matrix, precision\_score, recall\_score, f1\_score  
from sklearn.model\_selection import train\_test\_split  
from sklearn.preprocessing import LabelEncoder  
  
root\_dir = "keypoints\_orientation"  
classes = ['fake', 'real']  
directories = ['train', 'validation']  
  
data\_dict = {}  
  
# Parcurge fiecare director și încarcă datele  
for directory in directories:  
 for class\_name in classes:  
 class\_dir = os.path.join(root\_dir, directory, class\_name)  
 files = os.listdir(class\_dir)  
 for file in files:  
 if file.endswith('key\_orient.txt'):  
 file\_path = os.path.join(class\_dir, file)  
 with open(file\_path, 'r') as f:  
 lines = f.readlines()  
 for line in lines:  
 matches = re.findall(r'-?\d+\.?\d\*', line)  
 data\_dict[line] = class\_name  
  
texts = []  
labels = []  
  
# Parcurge dicționarul și adaugă linia și eticheta în vectori  
for line, label in data\_dict.items():  
 input\_values = line.strip().split(',')  
 input\_values = [float(value) for value in input\_values]  
 texts.append(input\_values)  
  
 if label == 'fake':  
 labels.append(0)  
 elif label == 'real':  
 labels.append(1)  
  
# Transformă vectorii în array-uri numpy  
texts = np.array(texts)  
labels = np.array(labels)  
  
# Împarte datele în set de antrenament și set de validare  
train\_texts, val\_texts, train\_labels, val\_labels = train\_test\_split(texts, labels, test\_size=0.2, random\_state=42)  
  
# Initializează un encoder pentru etichete  
label\_encoder = LabelEncoder()  
  
# Transformă etichetele în valori numerice  
encoded\_labels = label\_encoder.fit\_transform(labels.astype(str))  
  
# Transformă textele într-o reprezentare numerică  
numeric\_texts = texts.astype(float)  
  
# Transformă listele în array-uri numpy  
train\_texts = np.array(numeric\_texts)  
train\_labels = np.array(encoded\_labels)  
  
# Definirea și antrenarea modelului  
model = RandomForestClassifier(n\_estimators=600, random\_state=42)  
model.fit(train\_texts, train\_labels)  
  
# Realizează predicții pe setul de validare  
val\_predictions = model.predict(val\_texts)  
  
# Evaluarea performanței pe setul de validare  
val\_accuracy = accuracy\_score(val\_labels, val\_predictions)  
confusion\_mat = confusion\_matrix(val\_labels, val\_predictions)  
  
# Afișează performanța și matricea de confuzie  
print(f"Performanță pe setul de validare: Accuracy = {val\_accuracy:.4f}")  
print("Matricea de confuzie:")  
print(confusion\_mat)  
  
val\_precision = precision\_score(val\_labels, val\_predictions)  
val\_recall = recall\_score(val\_labels, val\_predictions)  
val\_f1\_score = f1\_score(val\_labels, val\_predictions)  
  
print(f"Set de validare - Precizie: {val\_precision:.4f}")  
print(f"Set de validare - Revocare: {val\_recall:.4f}")  
print(f"Set de validare - F1-Score: {val\_f1\_score:.4f}")  
  
  
# TESTARE  
  
data\_dict = {}  
test\_dir = os.path.join(root\_dir, "test")  
  
# Parcurge fiecare director și încarcă datele  
for directory in test\_dir:  
 for class\_name in classes:  
 class\_dir = os.path.join(test\_dir, class\_name)  
 files = os.listdir(class\_dir)  
 for file in files:  
 if file.endswith('key\_orient.txt'):  
 file\_path = os.path.join(class\_dir, file)  
 with open(file\_path, 'r') as f:  
 lines = f.readlines()  
 for line in lines:  
 matches = re.findall(r'-?\d+\.?\d\*', line)  
 data\_dict[line] = class\_name  
  
texts\_test = []  
labels\_test = []  
  
# Parcurge dicționarul și adaugă linia și eticheta în vectori  
for line, label in data\_dict.items():  
 input\_values = line.strip().split(',')  
 input\_values = [float(value) for value in input\_values]  
 texts\_test.append(input\_values)  
  
 if label == 'fake':  
 labels\_test.append(0)  
 elif label == 'real':  
 labels\_test.append(1)  
  
# Transformă vectorii în array-uri numpy  
test\_texts = np.array(texts\_test)  
test\_labels = np.array(labels\_test)  
  
# Transformă textele într-o reprezentare numerică  
numeric\_test\_texts = test\_texts.astype(float)  
  
# Preprocesează și codifică etichetele pentru setul de testare  
encoded\_test\_labels = label\_encoder.transform(test\_labels.astype(str))  
  
# Realizează predicții pe setul de testare  
test\_predictions = model.predict(numeric\_test\_texts)  
  
# Evaluarea performanței pe setul de testare  
test\_accuracy = accuracy\_score(encoded\_test\_labels, test\_predictions)  
test\_confusion\_mat = confusion\_matrix(encoded\_test\_labels, test\_predictions)  
  
# Afișează performanța și matricea de confuzie pentru setul de testare  
print(f"Performanță pe setul de testare: Accuracy = {test\_accuracy:.4f}")  
print("Matricea de confuzie pentru setul de testare:")  
print(test\_confusion\_mat)  
  
test\_precision = precision\_score(encoded\_test\_labels, test\_predictions)  
test\_recall = recall\_score(encoded\_test\_labels, test\_predictions)  
test\_f1\_score = f1\_score(encoded\_test\_labels, test\_predictions)  
  
print(f"Set de testare - Precizie: {test\_precision:.4f}")  
print(f"Set de testare - Revocare: {test\_recall:.4f}")  
print(f"Set de testare - F1-Score: {test\_f1\_score:.4f}")  
  
# Afișează performanța și matricea de confuzie pentru setul de validare  
print(f"Performanță pe setul de validare: Accuracy = {val\_accuracy:.4f}")  
print("Matricea de confuzie pentru setul de validare:")  
print(confusion\_mat)  
  
  
# Crearea graficului de clasificare  
def plot\_confusion\_matrix(cm, classes, normalize=False, title='Matrice de confuzie', cmap=plt.cm.Blues):  
 if normalize:  
 cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]  
 print("Matricea de confuzie normalizată")  
 else:  
 print('Matricea de confuzie, fără normalizare')  
  
 plt.imshow(cm, interpolation='nearest', cmap=cmap)  
 plt.title(title)  
 plt.colorbar()  
 tick\_marks = np.arange(len(classes))  
 plt.xticks(tick\_marks, classes, rotation=45)  
 plt.yticks(tick\_marks, classes)  
  
 fmt = '.2f' if normalize else 'd'  
 thresh = cm.max() / 2.  
 for i, j in itertools.product(range(cm.shape[0]), range(cm.shape[1])):  
 plt.text(j, i, format(cm[i, j], fmt), horizontalalignment="center", color="white" if cm[i, j] > thresh else "black")  
  
 plt.ylabel('Eticheta reală')  
 plt.xlabel('Eticheta prezisă')  
 plt.tight\_layout()  
  
# Afișarea matricei de confuzie pentru setul de validare  
plt.figure()  
plot\_confusion\_matrix(confusion\_mat, classes=classes, title='Matrice de confuzie - Validare')  
  
# Calcularea matricei de confuzie pentru setul de testare  
test\_confusion\_mat = confusion\_matrix(encoded\_test\_labels, test\_predictions)  
  
# Afișarea matricei de confuzie pentru setul de testare  
plt.figure()  
plot\_confusion\_matrix(test\_confusion\_mat, classes=classes, title='Matrice de confuzie - Testare')  
  
# Afișarea graficelor  
plt.show()  
  
# Definirea valorilor pentru predicție  
val = 31.0,36.0,31.0,38.0,32.0,31.0,33.0,34.0,-0.26331384175615524,-0.35197915712378447,-0.11069634694758786,-0.30010542117780975  
input\_values = [val]  
input\_values = np.array(input\_values).reshape(1, -1) # Reshape pentru a avea dimensiunea corectă  
  
# Realizarea predicției  
prediction = model.predict(input\_values)  
  
# Decodificarea predicției  
decoded\_prediction = label\_encoder.inverse\_transform(prediction)  
  
# Afișarea predicției  
print(f"Predicția pentru valorile furnizate: {decoded\_prediction}")

Testarea modelului – testing\_keras.py

import cv2  
import tensorflow as tf  
import numpy as np  
from PIL import Image as PILImage  
  
# Setarea parametrilor modelului  
input\_shape = (None, 80, 80, 1) # Assuming grayscale images  
  
image\_path = 'preprocessed\_dataset/test/FALS/fake\_fingerprint\_1.png'  
test\_image = PILImage.open(image\_path)  
test\_image = np.array(test\_image)  
test\_image = test\_image.reshape([1, 80, 80, 1])  
  
loaded\_model = tf.keras.models.load\_model('model.h5')  
# Realizați predicția utilizând modelul încărcat  
prediction = loaded\_model.predict(test\_image)  
  
# Extrageți valoarea scalară din tensorul de predicție  
prediction\_value = prediction[0][0]  
  
# Verificați clasa prezisă  
if prediction\_value > 0.5:  
 print("Imaginea de test aparține clasei pozitive - FALS")  
else:  
 print("Imaginea de test aparține clasei negative - REAL")

Preprocesarea imaginilor – preprocess.py

import cv2  
import numpy as np  
from matplotlib import pyplot as plt  
import os  
import uuid  
  
# Directorul cu imagini de intrare  
input\_dir = 'dataset/train/original\_train/'  
input\_dir\_2 = 'dataset/validation/real/'  
input\_dir\_3 = 'dataset/test/test\_dataset/'  
  
# Directorul de ieșire pentru rezultate  
output\_dir = 'preprocessed\_dataset/train/REAL/'  
output\_dir\_2 = 'preprocessed\_dataset/validation/REAL/'  
output\_dir\_3 = 'preprocessed\_dataset/test/REAL/'  
  
# Directorul pentru salvarea graficelor  
save\_dir = 'graphs/train/'  
save\_dir\_2 = 'graphs/validation/'  
save\_dir\_3 = 'graphs/test/'  
  
# Listarea imaginilor din directorul de intrare  
image\_files = os.listdir(input\_dir)  
image\_files\_2 = os.listdir(input\_dir\_2)  
image\_files\_3 = os.listdir(input\_dir\_3)  
  
# Parcurgerea fiecărei imagini din directorul de antrenare  
for image\_file in image\_files:  
 # Calea completă către imaginea de procesat  
 image\_path = os.path.join(input\_dir, image\_file)  
  
 # Incarcare imagine  
 image = cv2.imread(image\_path, cv2.IMREAD\_GRAYSCALE)  
  
 # PREPROCESARE  
  
 # Redimensionare imagine  
 resized\_image = cv2.resize(image, (80, 80))  
  
 # Corectie de contrast  
 equalized\_image = cv2.equalizeHist(resized\_image)  
  
 # Eliminare zgomot  
 denoised\_image = cv2.GaussianBlur(equalized\_image, (1, 1), 0)  
  
 # Umplerea golurilor din imagine  
 kernel = np.ones((5, 5), np.uint8) # filtrul prin care imaginea va fi trecuta  
 opened\_image = cv2.morphologyEx(denoised\_image, cv2.MORPH\_OPEN, kernel)  
 closed\_image = cv2.morphologyEx(opened\_image, cv2.MORPH\_CLOSE, kernel)  
  
 # Subtierea liniei amprentei  
 skeletonized\_image = cv2.morphologyEx(closed\_image, cv2.MORPH\_HITMISS, kernel)  
  
 # Detectarea marginilor  
 edges = cv2.Canny(skeletonized\_image, threshold1=50, threshold2=150)  
  
 # Binarizarea imaginii  
 \_, binary\_image = cv2.threshold(edges, 0, 255, cv2.THRESH\_BINARY + cv2.THRESH\_OTSU)  
  
 # Normalizarea imaginii  
 normalized\_image = binary\_image.astype(np.float32) / 255.0  
  
 # Transformari geometrice  
 height, width = normalized\_image.shape[:2] # obtinerea dimensiunilor imaginii  
 M = cv2.getRotationMatrix2D((width / 2, height / 2), 30, 1) # rotire la 30 grade  
 transformed\_image = cv2.warpAffine(normalized\_image, M, (width, height))  
  
 # Segmentarea bazata pe praguri multiple  
 threshold\_lower = 12  
 threshold\_upper = 5000  
 segmented\_image = cv2.inRange(binary\_image, threshold\_lower, threshold\_upper)  
  
 # Conturul  
 contours, \_ = cv2.findContours(binary\_image, cv2.RETR\_EXTERNAL, cv2.CHAIN\_APPROX\_SIMPLE) # gasirea contururilor  
 contour\_image = np.zeros\_like(transformed\_image) # crearea unei imagini pentru afisarea contururlor  
 cv2.drawContours(contour\_image, contours, -1, (255, 255, 255), thickness=1) # desenarea conturului  
  
 # Filtrul Gabor ( detectarea liniilor amprentei )  
 # parametrii  
 ksize = 31 # Dimensiunea kernelului  
 sigma = 3 # Deviația standard a gaussianei  
 theta = 0 # Unghiul de orientare al filtrului  
 lambd = 10 # Lungimea de undă a sinusoidului  
 gamma = 0.5 # Aspectul (raportul dintre lungimea de undă și deviația standard)  
  
 # generare filtru gabor  
 kernel = cv2.getGaborKernel((ksize, ksize), sigma, theta, lambd, gamma)  
 # aplicare filtru gabor  
 filtered\_image = cv2.filter2D(contour\_image, cv2.CV\_64F, kernel)  
  
 # Aplicare Watershed pentru segmentarea regiunilor de interes  
 dist\_transform = cv2.distanceTransform(segmented\_image, cv2.DIST\_L2, 3)  
 \_, sure\_fg = cv2.threshold(dist\_transform, 0.1 \* dist\_transform.max(), 255, 0)  
 sure\_fg = np.uint8(sure\_fg)  
 filtered\_image = filtered\_image.astype(np.uint8) # Conversie la tipul de date uint8  
 sure\_fg = sure\_fg.astype(np.uint8) # Conversie la tipul de date uint8  
 unknown = cv2.subtract(filtered\_image, sure\_fg)  
  
 \_, markers = cv2.connectedComponents(sure\_fg)  
  
 markers += 1  
 markers[unknown == 255] = 0  
  
 transformed\_image = transformed\_image.astype(np.uint8)  
 markers = markers.astype(np.int32)  
 transformed\_image\_color = cv2.cvtColor(transformed\_image, cv2.COLOR\_GRAY2BGR)  
 filtered = cv2.watershed(transformed\_image\_color, markers)  
 filtered\_image[filtered == -1] = 255  
  
 # Salvează imaginea preprocesată în directorul corespunzător  
 output\_path = os.path.join(output\_dir, image\_file)  
 cv2.imwrite(output\_path, filtered\_image)  
  
 # Creaza un plot și afisează imaginile  
 fig, axes = plt.subplots(nrows=4, ncols=4, figsize=(12, 8))  
  
 axes[0, 0].imshow(image)  
 axes[0, 1].imshow(resized\_image)  
 axes[0, 2].imshow(equalized\_image)  
 axes[0, 3].imshow(denoised\_image)  
 axes[1, 0].imshow(opened\_image)  
 axes[1, 1].imshow(closed\_image)  
 axes[1, 2].imshow(skeletonized\_image)  
 axes[1, 3].imshow(binary\_image)  
 axes[2, 0].imshow(normalized\_image)  
 axes[2, 1].imshow(transformed\_image)  
 axes[2, 2].imshow(segmented\_image)  
 axes[2, 3].imshow(filtered\_image)  
  
 plt.tight\_layout()  
 plt.axis('off')  
  
 # Crează un nume de fișier unic folosind modulul uuid  
 unique\_filename = str(uuid.uuid4())  
  
 # Concatenează numele fișierului unic cu extensia dorită  
 filename = f"plot\_image\_{unique\_filename}.png"  
  
 # Creați calea completă către fișierul de salvare  
 save\_path = os.path.join(save\_dir, filename)  
  
 # Salvează imaginea utilizând calea completă  
 plt.savefig(save\_path)  
  
 # Închide figura  
 plt.close(fig)  
  
print("Preprocess for train images done")  
  
# Parcurgerea fiecărei imagini din directorul de validare  
for image\_file in image\_files\_2:  
 # Calea completă către imaginea de procesat  
 image\_path = os.path.join(input\_dir\_2, image\_file)  
  
 # Incarcare imagine  
 image = cv2.imread(image\_path, cv2.IMREAD\_GRAYSCALE)  
  
 # PREPROCESARE  
  
 # Redimensionare imagine  
 resized\_image = cv2.resize(image, (80, 80))  
  
 # Corectie de contrast  
 equalized\_image = cv2.equalizeHist(resized\_image)  
  
 # Eliminare zgomot  
 denoised\_image = cv2.GaussianBlur(equalized\_image, (1, 1), 0)  
  
 # Umplerea golurilor din imagine  
 kernel = np.ones((5, 5), np.uint8) # filtrul prin care imaginea va fi trecuta  
 opened\_image = cv2.morphologyEx(denoised\_image, cv2.MORPH\_OPEN, kernel)  
 closed\_image = cv2.morphologyEx(opened\_image, cv2.MORPH\_CLOSE, kernel)  
  
 # Subtierea liniei amprentei  
 skeletonized\_image = cv2.morphologyEx(closed\_image, cv2.MORPH\_HITMISS, kernel)  
  
 # Detectarea marginilor  
 edges = cv2.Canny(skeletonized\_image, threshold1=50, threshold2=150)  
  
 # Binarizarea imaginii  
 \_, binary\_image = cv2.threshold(edges, 0, 255, cv2.THRESH\_BINARY + cv2.THRESH\_OTSU)  
  
 # Normalizarea imaginii  
 normalized\_image = binary\_image.astype(np.float32) / 255.0  
  
 # Transformari geometrice  
 height, width = normalized\_image.shape[:2] # obtinerea dimensiunilor imaginii  
 M = cv2.getRotationMatrix2D((width / 2, height / 2), 30, 1) # rotire la 30 grade  
 transformed\_image = cv2.warpAffine(normalized\_image, M, (width, height))  
  
 # Segmentarea bazata pe praguri multiple  
 threshold\_lower = 12  
 threshold\_upper = 5000  
 segmented\_image = cv2.inRange(binary\_image, threshold\_lower, threshold\_upper)  
  
 # Conturul  
 contours, \_ = cv2.findContours(binary\_image, cv2.RETR\_EXTERNAL, cv2.CHAIN\_APPROX\_SIMPLE) # gasirea contururilor  
 contour\_image = np.zeros\_like(transformed\_image) # crearea unei imagini pentru afisarea contururlor  
 cv2.drawContours(contour\_image, contours, -1, (255, 255, 255), thickness=1) # desenarea conturului  
  
 # Filtrul Gabor ( detectarea liniilor amprentei )  
 # parametrii  
 ksize = 31 # Dimensiunea kernelului  
 sigma = 3 # Deviația standard a gaussianei  
 theta = 0 # Unghiul de orientare al filtrului  
 lambd = 10 # Lungimea de undă a sinusoidului  
 gamma = 0.5 # Aspectul (raportul dintre lungimea de undă și deviația standard)  
  
 # generare filtru gabor  
 kernel = cv2.getGaborKernel((ksize, ksize), sigma, theta, lambd, gamma)  
 # aplicare filtru gabor  
 filtered\_image = cv2.filter2D(contour\_image, cv2.CV\_64F, kernel)  
  
 # Aplicare Watershed pentru segmentarea regiunilor de interes  
 dist\_transform = cv2.distanceTransform(segmented\_image, cv2.DIST\_L2, 3)  
 \_, sure\_fg = cv2.threshold(dist\_transform, 0.1 \* dist\_transform.max(), 255, 0)  
 sure\_fg = np.uint8(sure\_fg)  
 filtered\_image = filtered\_image.astype(np.uint8) # Conversie la tipul de date uint8  
 sure\_fg = sure\_fg.astype(np.uint8) # Conversie la tipul de date uint8  
 unknown = cv2.subtract(filtered\_image, sure\_fg)  
  
 \_, markers = cv2.connectedComponents(sure\_fg)  
  
 markers += 1  
 markers[unknown == 255] = 0  
  
 transformed\_image = transformed\_image.astype(np.uint8)  
 markers = markers.astype(np.int32)  
 transformed\_image\_color = cv2.cvtColor(transformed\_image, cv2.COLOR\_GRAY2BGR)  
 filtered = cv2.watershed(transformed\_image\_color, markers)  
 filtered\_image[filtered == -1] = 255  
  
 # Salvează imaginea preprocesată în directorul corespunzător  
 output\_path = os.path.join(output\_dir\_2, image\_file)  
 cv2.imwrite(output\_path, filtered\_image)  
  
 # Creaza un plot și afisează imaginile  
 fig, axes = plt.subplots(nrows=4, ncols=4, figsize=(12, 8))  
  
 axes[0, 0].imshow(image)  
 axes[0, 1].imshow(resized\_image)  
 axes[0, 2].imshow(equalized\_image)  
 axes[0, 3].imshow(denoised\_image)  
 axes[1, 0].imshow(opened\_image)  
 axes[1, 1].imshow(closed\_image)  
 axes[1, 2].imshow(skeletonized\_image)  
 axes[1, 3].imshow(binary\_image)  
 axes[2, 0].imshow(normalized\_image)  
 axes[2, 1].imshow(transformed\_image)  
 axes[2, 2].imshow(segmented\_image)  
 axes[2, 3].imshow(filtered\_image)  
  
 plt.tight\_layout()  
 plt.axis('off')  
  
 # Crează un nume de fișier unic folosind modulul uuid  
 unique\_filename = str(uuid.uuid4())  
  
 # Concatenează numele fișierului unic cu extensia dorită  
 filename = f"plot\_image\_{unique\_filename}.png"  
  
 # Creați calea completă către fișierul de salvare  
 save\_path = os.path.join(save\_dir\_2, filename)  
  
 # Salvează imaginea utilizând calea completă  
 plt.savefig(save\_path)  
  
 # Închide figura  
 plt.close(fig)  
  
print("Preprocess for validation images done")  
  
# Parcurgerea fiecărei imagini din directorul de test  
for image\_file in image\_files\_3:  
 # Calea completă către imaginea de procesat  
 image\_path = os.path.join(input\_dir\_3, image\_file)  
  
 # Incarcare imagine  
 image = cv2.imread(image\_path, cv2.IMREAD\_GRAYSCALE)  
  
 # PREPROCESARE  
  
 # Redimensionare imagine  
 resized\_image = cv2.resize(image, (80, 80))  
  
 # Corectie de contrast  
 equalized\_image = cv2.equalizeHist(resized\_image)  
  
 # Eliminare zgomot  
 denoised\_image = cv2.GaussianBlur(equalized\_image, (1, 1), 0)  
  
 # Umplerea golurilor din imagine  
 kernel = np.ones((5, 5), np.uint8) # filtrul prin care imaginea va fi trecuta  
 opened\_image = cv2.morphologyEx(denoised\_image, cv2.MORPH\_OPEN, kernel)  
 closed\_image = cv2.morphologyEx(opened\_image, cv2.MORPH\_CLOSE, kernel)  
  
 # Subtierea liniei amprentei  
 skeletonized\_image = cv2.morphologyEx(closed\_image, cv2.MORPH\_HITMISS, kernel)  
  
 # Detectarea marginilor  
 edges = cv2.Canny(skeletonized\_image, threshold1=50, threshold2=150)  
  
 # Binarizarea imaginii  
 \_, binary\_image = cv2.threshold(edges, 0, 255, cv2.THRESH\_BINARY + cv2.THRESH\_OTSU)  
  
 # Normalizarea imaginii  
 normalized\_image = binary\_image.astype(np.float32) / 255.0  
  
 # Transformari geometrice  
 height, width = normalized\_image.shape[:2] # obtinerea dimensiunilor imaginii  
 M = cv2.getRotationMatrix2D((width / 2, height / 2), 30, 1) # rotire la 30 grade  
 transformed\_image = cv2.warpAffine(normalized\_image, M, (width, height))  
  
 # Segmentarea bazata pe praguri multiple  
 threshold\_lower = 12  
 threshold\_upper = 5000  
 segmented\_image = cv2.inRange(binary\_image, threshold\_lower, threshold\_upper)  
  
 # Conturul  
 contours, \_ = cv2.findContours(binary\_image, cv2.RETR\_EXTERNAL, cv2.CHAIN\_APPROX\_SIMPLE) # gasirea contururilor  
 contour\_image = np.zeros\_like(transformed\_image) # crearea unei imagini pentru afisarea contururlor  
 cv2.drawContours(contour\_image, contours, -1, (255, 255, 255), thickness=1) # desenarea conturului  
  
 # Filtrul Gabor ( detectarea liniilor amprentei )  
 # parametrii  
 ksize = 31 # Dimensiunea kernelului  
 sigma = 3 # Deviația standard a gaussianei  
 theta = 0 # Unghiul de orientare al filtrului  
 lambd = 10 # Lungimea de undă a sinusoidului  
 gamma = 0.5 # Aspectul (raportul dintre lungimea de undă și deviația standard)  
  
 # generare filtru gabor  
 kernel = cv2.getGaborKernel((ksize, ksize), sigma, theta, lambd, gamma)  
 # aplicare filtru gabor  
 filtered\_image = cv2.filter2D(contour\_image, cv2.CV\_64F, kernel)  
  
 # Aplicare Watershed pentru segmentarea regiunilor de interes  
 dist\_transform = cv2.distanceTransform(segmented\_image, cv2.DIST\_L2, 3)  
 \_, sure\_fg = cv2.threshold(dist\_transform, 0.1 \* dist\_transform.max(), 255, 0)  
 sure\_fg = np.uint8(sure\_fg)  
 filtered\_image = filtered\_image.astype(np.uint8) # Conversie la tipul de date uint8  
 sure\_fg = sure\_fg.astype(np.uint8) # Conversie la tipul de date uint8  
 unknown = cv2.subtract(filtered\_image, sure\_fg)  
  
 \_, markers = cv2.connectedComponents(sure\_fg)  
  
 markers += 1  
 markers[unknown == 255] = 0  
  
 transformed\_image = transformed\_image.astype(np.uint8)  
 markers = markers.astype(np.int32)  
 transformed\_image\_color = cv2.cvtColor(transformed\_image, cv2.COLOR\_GRAY2BGR)  
 filtered = cv2.watershed(transformed\_image\_color, markers)  
 filtered\_image[filtered == -1] = 255  
  
 # Salvează imaginea preprocesată în directorul corespunzător  
 output\_path = os.path.join(output\_dir\_3, image\_file)  
 cv2.imwrite(output\_path, filtered\_image)  
  
 # Creaza un plot și afisează imaginile  
 fig, axes = plt.subplots(nrows=4, ncols=4, figsize=(12, 8))  
  
 axes[0, 0].imshow(image)  
 axes[0, 1].imshow(resized\_image)  
 axes[0, 2].imshow(equalized\_image)  
 axes[0, 3].imshow(denoised\_image)  
 axes[1, 0].imshow(opened\_image)  
 axes[1, 1].imshow(closed\_image)  
 axes[1, 2].imshow(skeletonized\_image)  
 axes[1, 3].imshow(binary\_image)  
 axes[2, 0].imshow(normalized\_image)  
 axes[2, 1].imshow(transformed\_image)  
 axes[2, 2].imshow(segmented\_image)  
 axes[2, 3].imshow(filtered\_image)  
  
 plt.tight\_layout()  
 plt.axis('off')  
  
 # Crează un nume de fișier unic folosind modulul uuid  
 unique\_filename = str(uuid.uuid4())  
  
 # Concatenează numele fișierului unic cu extensia dorită  
 filename = f"plot\_image\_{unique\_filename}.png"  
  
 # Creați calea completă către fișierul de salvare  
 save\_path = os.path.join(save\_dir\_3, filename)  
  
 # Salvează imaginea utilizând calea completă  
 plt.savefig(save\_path)  
  
 # Închide figura  
 plt.close(fig)  
  
print("Preprocess for test images done")

Extragerea datelor biometrice – iterator\_biometrics.py

import os  
  
import cv2  
import numpy as np  
from matplotlib import pyplot as plt  
  
  
# Funcție pentru identificarea punctelor de interes din amprentă  
def detect\_keypoints(image):  
 # Aplică algoritmul ORB pentru detectarea punctelor de interes  
 orb = cv2.ORB\_create()  
 keypoints = orb.detect(image, None)  
 return keypoints  
  
  
# Funcție pentru calcularea direcției liniei ridge în fiecare punct  
def compute\_orientation(image, keypoints):  
 # Calculează direcția liniei ridge în fiecare punct cheie  
 orientations = []  
 for keypoint in keypoints:  
 x, y = keypoint.pt  
 # Extrage o fereastră vecină în jurul punctului cheie  
 window\_size = 16  
 window = image[int(y) - window\_size:int(y) + window\_size, int(x) - window\_size:int(x) + window\_size]  
 # Calculează gradientul imaginii în fereastra vecină  
 gradient\_x = cv2.Sobel(window, cv2.CV\_64F, 1, 0, ksize=3)  
 gradient\_y = cv2.Sobel(window, cv2.CV\_64F, 0, 1, ksize=3)  
 # Calculează direcția liniei ridge utilizând gradientul  
 orientation = np.arctan2(gradient\_y.mean(), gradient\_x.mean())  
 orientations.append(orientation)  
 return orientations  
  
  
# Imaginea prelucrată a amprentei  
input\_dir1 = "preprocessed\_dataset/train/REAL/"  
input\_dir2 = "preprocessed\_dataset/validation/REAL/"  
input\_dir3 = "preprocessed\_dataset/test/REAL/"  
all\_keypoints = []  
all\_orientations = []  
image\_files1 = os.listdir(input\_dir1)  
image\_files2 = os.listdir(input\_dir2)  
image\_files3 = os.listdir(input\_dir3)  
  
all\_keypoints\_orientations = []  
for image\_file1 in image\_files1:  
 image\_path = os.path.join(input\_dir1, image\_file1)  
  
 # Incarcarea imaginilor  
 preprocessed\_image = cv2.imread(image\_path, 0)  
 if preprocessed\_image is None:  
 print(f"Nu s-a putut încărca imaginea {image\_path}!")  
 else:  
 print(f"S-a incarcat imaginea {image\_path}")  
  
 # Identificarea keypoints  
 keypoints = detect\_keypoints(preprocessed\_image)  
  
 # Calcularea directiei liniei ridge in fiecare punct  
 orientations = compute\_orientation(preprocessed\_image, keypoints)  
  
 keypoints\_orientations = list(zip(keypoints, orientations))  
 keypoints\_orientations = sorted(keypoints\_orientations, key=lambda x: x[0].pt[0])  
  
 # Salvarea primelor 4 keypoints și orientări într-un array  
 keypoints\_array = []  
 orientations\_array = []  
 for keypoint, orientation in keypoints\_orientations[:4]:  
 keypoints\_array.append([keypoint.pt[0], keypoint.pt[1]])  
 orientations\_array.append(orientation)  
  
 all\_keypoints\_orientations.append((keypoints\_array, orientations\_array))  
  
for image\_file2 in image\_files2:  
 image\_path = os.path.join(input\_dir2, image\_file2)  
  
 # Incarcarea imaginilor  
 preprocessed\_image = cv2.imread(image\_path, 0)  
 if preprocessed\_image is None:  
 print(f"Nu s-a putut încărca imaginea {image\_path}!")  
 else:  
 print(f"S-a incarcat imaginea {image\_path}")  
  
 # Identificarea keypoints  
 keypoints = detect\_keypoints(preprocessed\_image)  
  
 # Calcularea directiei liniei ridge in fiecare punct  
 orientations = compute\_orientation(preprocessed\_image, keypoints)  
  
 keypoints\_orientations = list(zip(keypoints, orientations))  
 keypoints\_orientations = sorted(keypoints\_orientations, key=lambda x: x[0].pt[0])  
  
 # Salvarea primelor 4 keypoints și orientări într-un array  
 keypoints\_array = []  
 orientations\_array = []  
 for keypoint, orientation in keypoints\_orientations[:4]:  
 keypoints\_array.append([keypoint.pt[0], keypoint.pt[1]])  
 orientations\_array.append(orientation)  
  
 all\_keypoints\_orientations.append((keypoints\_array, orientations\_array))  
  
for image\_file3 in image\_files3:  
 image\_path = os.path.join(input\_dir3, image\_file3)  
  
 # Incarcarea imaginilor  
 preprocessed\_image = cv2.imread(image\_path, 0)  
 if preprocessed\_image is None:  
 print(f"Nu s-a putut încărca imaginea {image\_path}!")  
 else:  
 print(f"S-a incarcat imaginea {image\_path}")  
  
 # Identificarea keypoints  
 keypoints = detect\_keypoints(preprocessed\_image)  
  
 # Calcularea directiei liniei ridge in fiecare punct  
 orientations = compute\_orientation(preprocessed\_image, keypoints)  
  
 keypoints\_orientations = list(zip(keypoints, orientations))  
 keypoints\_orientations = sorted(keypoints\_orientations, key=lambda x: x[0].pt[0])  
  
 # Salvarea primelor 4 keypoints și orientări într-un array  
 keypoints\_array = []  
 orientations\_array = []  
 for keypoint, orientation in keypoints\_orientations[:4]:  
 keypoints\_array.append([keypoint.pt[0], keypoint.pt[1]])  
 orientations\_array.append(orientation)  
  
 all\_keypoints\_orientations.append((keypoints\_array, orientations\_array))  
  
# Salvarea keypoints și orientărilor într-un fișier text  
keypoints\_orientations\_path1 = 'keypoints\_orientation/train/real/key\_orient.txt'  
keypoints\_orientations\_path2 = 'keypoints\_orientation/validation/real/key\_orient.txt'  
keypoints\_orientations\_path3 = 'keypoints\_orientation/test/real/key\_orient.txt'  
  
with open(keypoints\_orientations\_path1, 'w') as f:  
 for keypoints\_array, orientations\_array in all\_keypoints\_orientations:  
 keypoint\_str = ",".join([f"{keypoint[0]},{keypoint[1]}" for keypoint in keypoints\_array])  
 if len(keypoints\_array) == 4 and len(orientations\_array) == 4:  
 f.write(keypoint\_str + ",")  
 orientation\_str = ",".join([str(orientation) for orientation in orientations\_array])  
 f.write(orientation\_str + "\n")  
  
with open(keypoints\_orientations\_path2, 'w') as f:  
 for keypoints\_array, orientations\_array in all\_keypoints\_orientations:  
 keypoint\_str = ",".join([f"{keypoint[0]},{keypoint[1]}" for keypoint in keypoints\_array])  
 if len(keypoints\_array) == 4 and len(orientations\_array) == 4:  
 f.write(keypoint\_str + ",")  
 orientation\_str = ",".join([str(orientation) for orientation in orientations\_array])  
 f.write(orientation\_str + "\n")  
  
with open(keypoints\_orientations\_path3, 'w') as f:  
 for keypoints\_array, orientations\_array in all\_keypoints\_orientations:  
 keypoint\_str = ",".join([f"{keypoint[0]},{keypoint[1]}" for keypoint in keypoints\_array])  
 if len(keypoints\_array) == 4 and len(orientations\_array) == 4:  
 f.write(keypoint\_str + ",")  
 orientation\_str = ",".join([str(orientation) for orientation in orientations\_array])  
 f.write(orientation\_str + "\n")

Antrenarea modelului RandomForest – train\_randomForest.py

import itertools  
import os  
import re  
  
import numpy as np  
from matplotlib import pyplot as plt  
from sklearn.ensemble import RandomForestClassifier  
from sklearn.metrics import accuracy\_score, confusion\_matrix, precision\_score, recall\_score, f1\_score  
from sklearn.model\_selection import train\_test\_split  
from sklearn.preprocessing import LabelEncoder  
  
root\_dir = "keypoints\_orientation"  
classes = ['fake', 'real']  
directories = ['train', 'validation']  
  
data\_dict = {}  
  
# Parcurge fiecare director și încarcă datele  
for directory in directories:  
 for class\_name in classes:  
 class\_dir = os.path.join(root\_dir, directory, class\_name)  
 files = os.listdir(class\_dir)  
 for file in files:  
 if file.endswith('key\_orient.txt'):  
 file\_path = os.path.join(class\_dir, file)  
 with open(file\_path, 'r') as f:  
 lines = f.readlines()  
 for line in lines:  
 matches = re.findall(r'-?\d+\.?\d\*', line)  
 data\_dict[line] = class\_name  
  
texts = []  
labels = []  
  
# Parcurge dicționarul și adaugă linia și eticheta în vectori  
for line, label in data\_dict.items():  
 input\_values = line.strip().split(',')  
 input\_values = [float(value) for value in input\_values]  
 texts.append(input\_values)  
  
 if label == 'fake':  
 labels.append(0)  
 elif label == 'real':  
 labels.append(1)  
  
# Transformă vectorii în array-uri numpy  
texts = np.array(texts)  
labels = np.array(labels)  
  
# Împarte datele în set de antrenament și set de validare  
train\_texts, val\_texts, train\_labels, val\_labels = train\_test\_split(texts, labels, test\_size=0.2, random\_state=42)  
  
# Initializează un encoder pentru etichete  
label\_encoder = LabelEncoder()  
  
# Transformă etichetele în valori numerice  
encoded\_labels = label\_encoder.fit\_transform(labels.astype(str))  
  
# Transformă textele într-o reprezentare numerică  
numeric\_texts = texts.astype(float)  
  
# Transformă listele în array-uri numpy  
train\_texts = np.array(numeric\_texts)  
train\_labels = np.array(encoded\_labels)  
  
# Definirea și antrenarea modelului  
model = RandomForestClassifier(n\_estimators=600, random\_state=42)  
model.fit(train\_texts, train\_labels)  
  
# Realizează predicții pe setul de validare  
val\_predictions = model.predict(val\_texts)  
  
# Evaluarea performanței pe setul de validare  
val\_accuracy = accuracy\_score(val\_labels, val\_predictions)  
confusion\_mat = confusion\_matrix(val\_labels, val\_predictions)  
  
# Afișează performanța și matricea de confuzie  
print(f"Performanță pe setul de validare: Accuracy = {val\_accuracy:.4f}")  
print("Matricea de confuzie:")  
print(confusion\_mat)  
  
val\_precision = precision\_score(val\_labels, val\_predictions)  
val\_recall = recall\_score(val\_labels, val\_predictions)  
val\_f1\_score = f1\_score(val\_labels, val\_predictions)  
  
print(f"Set de validare - Precizie: {val\_precision:.4f}")  
print(f"Set de validare - Revocare: {val\_recall:.4f}")  
print(f"Set de validare - F1-Score: {val\_f1\_score:.4f}")  
  
  
# TESTARE  
  
data\_dict = {}  
test\_dir = os.path.join(root\_dir, "test")  
  
# Parcurge fiecare director și încarcă datele  
for directory in test\_dir:  
 for class\_name in classes:  
 class\_dir = os.path.join(test\_dir, class\_name)  
 files = os.listdir(class\_dir)  
 for file in files:  
 if file.endswith('key\_orient.txt'):  
 file\_path = os.path.join(class\_dir, file)  
 with open(file\_path, 'r') as f:  
 lines = f.readlines()  
 for line in lines:  
 matches = re.findall(r'-?\d+\.?\d\*', line)  
 data\_dict[line] = class\_name  
  
texts\_test = []  
labels\_test = []  
  
# Parcurge dicționarul și adaugă linia și eticheta în vectori  
for line, label in data\_dict.items():  
 input\_values = line.strip().split(',')  
 input\_values = [float(value) for value in input\_values]  
 texts\_test.append(input\_values)  
  
 if label == 'fake':  
 labels\_test.append(0)  
 elif label == 'real':  
 labels\_test.append(1)  
  
# Transformă vectorii în array-uri numpy  
test\_texts = np.array(texts\_test)  
test\_labels = np.array(labels\_test)  
  
# Transformă textele într-o reprezentare numerică  
numeric\_test\_texts = test\_texts.astype(float)  
  
# Preprocesează și codifică etichetele pentru setul de testare  
encoded\_test\_labels = label\_encoder.transform(test\_labels.astype(str))  
  
# Realizează predicții pe setul de testare  
test\_predictions = model.predict(numeric\_test\_texts)  
  
# Evaluarea performanței pe setul de testare  
test\_accuracy = accuracy\_score(encoded\_test\_labels, test\_predictions)  
test\_confusion\_mat = confusion\_matrix(encoded\_test\_labels, test\_predictions)  
  
# Afișează performanța și matricea de confuzie pentru setul de testare  
print(f"Performanță pe setul de testare: Accuracy = {test\_accuracy:.4f}")  
print("Matricea de confuzie pentru setul de testare:")  
print(test\_confusion\_mat)  
  
test\_precision = precision\_score(encoded\_test\_labels, test\_predictions)  
test\_recall = recall\_score(encoded\_test\_labels, test\_predictions)  
test\_f1\_score = f1\_score(encoded\_test\_labels, test\_predictions)  
  
print(f"Set de testare - Precizie: {test\_precision:.4f}")  
print(f"Set de testare - Revocare: {test\_recall:.4f}")  
print(f"Set de testare - F1-Score: {test\_f1\_score:.4f}")  
  
# Afișează performanța și matricea de confuzie pentru setul de validare  
print(f"Performanță pe setul de validare: Accuracy = {val\_accuracy:.4f}")  
print("Matricea de confuzie pentru setul de validare:")  
print(confusion\_mat)  
  
  
# Crearea graficului de clasificare  
def plot\_confusion\_matrix(cm, classes, normalize=False, title='Matrice de confuzie', cmap=plt.cm.Blues):  
 if normalize:  
 cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]  
 print("Matricea de confuzie normalizată")  
 else:  
 print('Matricea de confuzie, fără normalizare')  
  
 plt.imshow(cm, interpolation='nearest', cmap=cmap)  
 plt.title(title)  
 plt.colorbar()  
 tick\_marks = np.arange(len(classes))  
 plt.xticks(tick\_marks, classes, rotation=45)  
 plt.yticks(tick\_marks, classes)  
  
 fmt = '.2f' if normalize else 'd'  
 thresh = cm.max() / 2.  
 for i, j in itertools.product(range(cm.shape[0]), range(cm.shape[1])):  
 plt.text(j, i, format(cm[i, j], fmt), horizontalalignment="center", color="white" if cm[i, j] > thresh else "black")  
  
 plt.ylabel('Eticheta reală')  
 plt.xlabel('Eticheta prezisă')  
 plt.tight\_layout()  
  
# Afișarea matricei de confuzie pentru setul de validare  
plt.figure()  
plot\_confusion\_matrix(confusion\_mat, classes=classes, title='Matrice de confuzie - Validare')  
  
# Calcularea matricei de confuzie pentru setul de testare  
test\_confusion\_mat = confusion\_matrix(encoded\_test\_labels, test\_predictions)  
  
# Afișarea matricei de confuzie pentru setul de testare  
plt.figure()  
plot\_confusion\_matrix(test\_confusion\_mat, classes=classes, title='Matrice de confuzie - Testare')  
  
# Afișarea graficelor  
plt.show()  
  
# Definirea valorilor pentru predicție  
val = 31.0,36.0,31.0,38.0,32.0,31.0,33.0,34.0,-0.26331384175615524,-0.35197915712378447,-0.11069634694758786,-0.30010542117780975  
input\_values = [val]  
input\_values = np.array(input\_values).reshape(1, -1) # Reshape pentru a avea dimensiunea corectă  
  
# Realizarea predicției  
prediction = model.predict(input\_values)  
  
# Decodificarea predicției  
decoded\_prediction = label\_encoder.inverse\_transform(prediction)  
  
# Afișarea predicției  
print(f"Predicția pentru valorile furnizate: {decoded\_prediction}")

Generarea amprentelor false – fake\_fingerprints.py

import os  
  
import numpy as np  
import cv2  
import glob  
  
  
def generate\_fake\_fingerprints(real\_fingerprints, output\_directory):  
 fake\_fingerprints = []  
  
 # Creează directorul de ieșire dacă nu există deja  
 os.makedirs(output\_directory, exist\_ok=True)  
  
 for i, real\_fingerprint in enumerate(real\_fingerprints):  
 img = cv2.imread(real\_fingerprint, 0)  
 if img is not None:  
 # Adaugă zgomot aleatoriu în pixeli  
 noise = np.random.normal(0, 25, img.shape).astype(np.uint8)  
 noisy\_img = cv2.add(img, noise)  
  
 # Aplică o tehnică de preprocesare sau modificare suplimentară a imaginii  
 # De exemplu, puteți aplica un filtru pentru a îmbunătăți claritatea amprentei  
  
 # Exemplu: Aplică un filtru Gaussian  
 filtered\_img = cv2.GaussianBlur(noisy\_img, (5, 5), 0)  
  
 # Generează calea de salvare a amprentei false  
 filename = f"fake\_fingerprint\_{i + 1}.png"  
 save\_path = os.path.join(output\_directory, filename)  
  
 # Salvează amprenta falsă  
 cv2.imwrite(save\_path, filtered\_img)  
  
 # Adaugă calea de salvare a amprentei false în lista rezultatelor  
 fake\_fingerprints.append(save\_path)  
 else:  
 print(f"Nu s-a putut încărca imaginea: {real\_fingerprint}")  
  
 return fake\_fingerprints  
  
  
#real\_fingerprints\_directory = "dataset/train/original\_train/"  
#real\_fingerprints\_directory = "dataset/validation/real/"  
real\_fingerprints\_directory = "dataset/test/test\_dataset/"  
  
#output\_directory = "dataset/train/fake\_train/"  
#output\_directory = "dataset/validation/fake/"  
output\_directory = "dataset/test/fake\_test\_dataset/"  
  
# Obține lista fișierelor din directorul amprentelor reale  
real\_fingerprints = glob.glob(os.path.join(real\_fingerprints\_directory, "\*.bmp"))  
  
# Generează amprente false pentru fiecare fișier  
fake\_fingerprints = generate\_fake\_fingerprints(real\_fingerprints, output\_directory)

# Rezultate

# Rezultate Keras

Epoch 50/50

57/57 [==============================] - 4s 68ms/step - loss: 0.1556 - **accuracy: 0.9354** - val\_loss: 1.1399 - **val\_accuracy: 0.6236**

Test Loss: 0.8663791418075562

**Test Accuracy: 0.6846590638160706**

**Prediction**: [[1. 0.]]

The test image belongs to the positive class

FALS

Matricea de confuzie:

[[ 75 101]

[ 68 108]]

A picture containing text, diagram, plot, line

Description automatically generated

Rezultate RandomForest

Performanță pe **setul de validare**: Accuracy = 1.0000

Matricea de confuzie:

[[413 0]

[ 0 360]]

Set de validare - Precizie: 1.0000

Set de validare - Revocare: 1.0000

Set de validare - F1-Score: 1.0000

Performanță pe setul de testare: Accuracy = 1.0000

Matricea de confuzie pentru **setul de testare**:

[[1996 0]

[ 0 1868]]

Set de testare - Precizie: 1.0000

Set de testare - Revocare: 1.0000

Set de testare - F1-Score: 1.0000

Performanță pe setul de validare: Accuracy = 1.0000

**Predicția** pentru valorile furnizate: ['1']

A picture containing text, screenshot, diagram, rectangle

Description automatically generated

A picture containing text, screenshot, diagram, number

Description automatically generated