Use of advanced ai

# 🚨 Forensic AI Toolkit

import numpy as np

import cv2

import matplotlib.pyplot as plt

from sklearn.metrics.pairwise import cosine\_similarity as cos\_sim

from sklearn.preprocessing import MinMaxScaler

from sklearn.cluster import KMeans

from sklearn.ensemble import GradientBoostingClassifier

import tensorflow as tf

from tensorflow.keras.applications import ResNet50

from tensorflow.keras.models import Model

from tensorflow.keras.layers import Dense, GlobalAveragePooling2D

from tensorflow.keras.preprocessing.image import img\_to\_array, load\_img

from google.colab import files

# ======================================

# 🔍 Fingerprint Feature Encoder (CNN)

# ======================================

class FingerprintFeatureEncoder:

def \_\_init\_\_(self):

base\_net = ResNet50(weights='imagenet', include\_top=False)

gap\_layer = GlobalAveragePooling2D()(base\_net.output)

embedding = Dense(128, activation='relu')(gap\_layer)

self.extractor = Model(inputs=base\_net.input, outputs=embedding)

print("[SYSTEM] Feature extractor initialized successfully.")

def preprocess\_image(self, img\_file):

img = load\_img(img\_file, target\_size=(224, 224))

arr = img\_to\_array(img)

arr = np.expand\_dims(arr, axis=0)

return tf.keras.applications.resnet50.preprocess\_input(arr)

def get\_embedding(self, img\_file):

arr = self.preprocess\_image(img\_file)

vector = self.extractor.predict(arr)[0]

return vector / np.linalg.norm(vector)

def compare\_fingerprints(self, query\_file, db\_files):

query\_vector = self.get\_embedding(query\_file).reshape(1, -1)

scores = []

for sid, file in db\_files.items():

db\_vector = self.get\_embedding(file).reshape(1, -1)

similarity = cos\_sim(query\_vector, db\_vector)[0][0]

scores.append((sid, similarity))

return sorted(scores, key=lambda x: x[1], reverse=True)

# ======================================

# 🩸 Blood Pattern Recognition

# ======================================

def identify\_blood\_patterns(scene\_file):

image = cv2.imread(scene\_file)

hsv\_image = cv2.cvtColor(image, cv2.COLOR\_BGR2HSV)

red\_mask1 = cv2.inRange(hsv\_image, (0,50,50), (10,255,255))

red\_mask2 = cv2.inRange(hsv\_image, (170,50,50), (180,255,255))

combined\_mask = cv2.bitwise\_or(red\_mask1, red\_mask2)

output = cv2.bitwise\_and(image, image, mask=combined\_mask)

plt.figure(figsize=(12,6))

plt.subplot(1,2,1)

plt.title("Original Scene")

plt.imshow(cv2.cvtColor(image, cv2.COLOR\_BGR2RGB))

plt.subplot(1,2,2)

plt.title("Detected Blood Patterns")

plt.imshow(cv2.cvtColor(output, cv2.COLOR\_BGR2RGB))

plt.show()

# ======================================

# 💥 Projectile Trajectory Simulation

# ======================================

def calculate\_projectile():

print("\n[INPUT] Provide firearm parameters")

try:

velocity = float(input("Muzzle Velocity (m/s): "))

theta = float(input("Firing Angle (degrees): "))

except ValueError:

print("[ERROR] Invalid values, defaulting to preset parameters.")

velocity, theta = 700, 40

gravity = 9.81

t\_max = (2 \* velocity \* np.sin(np.radians(theta))) / gravity

t\_points = np.linspace(0, t\_max, 500)

x\_coords = velocity \* np.cos(np.radians(theta)) \* t\_points

y\_coords = velocity \* np.sin(np.radians(theta)) \* t\_points - 0.5 \* gravity \* t\_points\*\*2

plt.plot(x\_coords, y\_coords, color='purple')

plt.title("Projectile Path Simulation")

plt.xlabel("Distance (m)")

plt.ylabel("Height (m)")

plt.grid(True)

plt.show()

# ======================================

# 🏗️ Crime Scene 3D Mapping

# ======================================

def map\_crime\_scene():

evidence\_points = np.random.uniform(0, 50, size=(60,3))

clusterer = KMeans(n\_clusters=5, random\_state=99)

cluster\_labels = clusterer.fit\_predict(evidence\_points)

fig = plt.figure(figsize=(8,6))

ax = fig.add\_subplot(111, projection='3d')

for cluster\_id in np.unique(cluster\_labels):

ax.scatter(evidence\_points[cluster\_labels==cluster\_id,0],

evidence\_points[cluster\_labels==cluster\_id,1],

evidence\_points[cluster\_labels==cluster\_id,2],

label=f"Cluster-{cluster\_id+1}")

ax.set\_title("3D Evidence Mapping")

ax.set\_xlabel("X-axis")

ax.set\_ylabel("Y-axis")

ax.set\_zlabel("Z-axis")

ax.legend()

plt.show()

# ======================================

# 🎯 Integrated Demo

# ======================================

def launch\_demo():

print("\n🚨 Forensic AI System: Interactive Demo")

# Fingerprint Matching

print("\n🖐 Upload Reference Fingerprint")

uploaded\_query = files.upload()

query\_img = list(uploaded\_query.keys())[0]

print("\n🖐 Upload Database Fingerprints")

uploaded\_db = files.upload()

db\_images = {f"Suspect-{i+1:02}": img for i, img in enumerate(uploaded\_db.keys())}

encoder = FingerprintFeatureEncoder()

result\_scores = encoder.compare\_fingerprints(query\_img, db\_images)

print("\n✅ Similarity Results:")

for sid, score in result\_scores:

print(f"{sid}: {score\*100:.2f}%")

# Blood Detection

print("\n🩸 Upload Crime Scene Photo for Blood Detection")

uploaded\_scene = files.upload()

scene\_img = list(uploaded\_scene.keys())[0]

identify\_blood\_patterns(scene\_img)

# Ballistics Simulation

print("\n💥 Simulating Projectile Motion")

calculate\_projectile()

# 3D Scene Mapping

print("\n🏗️ Reconstructing 3D Crime Scene")

map\_crime\_scene()

if \_\_name\_\_ == "\_\_main\_\_":

launch\_demo()