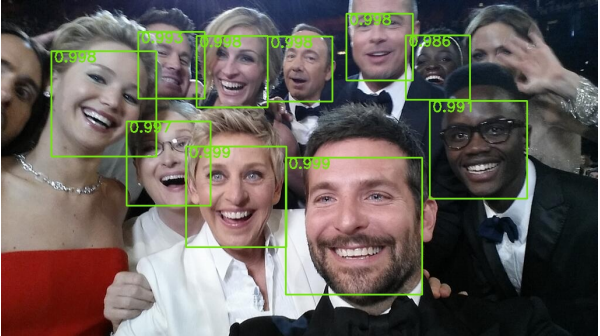
**Face Detection Models: A Comparative Analysis**

**Project 3: Gianmarco Iannario**



1)Description

# Introduction

Face detection is an AI-based computer technology that can identify and locate the presence of human faces in digital photos and videos.

It can be regarded as a special case of object-class detection, where the task is to find the locations and specify the sizes of all the objects that belong to a given class - in this case, faces - within a specific image or images. Face detection technology can be applied to various fields - including security, biometrics, law enforcement, entertainment and personal safety - to provide surveillance and tracking of people in real time. Face detection models often represents the basis for other computer vision approaches, such as face recognition, age estimation or face expression prediction.

In this project, my goal is to draw bounding boxes on faces using pre-trained models like Haar cascades, MTCNN, cvlib and a Caffe model using OpenCV’s DNN module. Then I will compare them to find out which works the best for real-time applications.

# Description

For this project, my goal is to draw bounding boxes on faces using pre-trained models like Haar cascades, MTCNN, cvlib and a Caffe model using OpenCV’s DNN module. Then I will compare them to find out which works the best for real-time applications.

These approaches are tested on 27 pictures, where the first 26 are taken from the WIDER Validation Dataset (<http://shuoyang1213.me/WIDERFACE/>). Each picture represents a different real-world situation, for example:

- 0.jpg: Parade

- 1.jpg: Handshaking

... (Check the first 26 directory of the WIDER dataset)

- 27.jpg: No faces picture (does not belong to WIDER)

Here is a short description of the different models:

* **Haar Cascade**

Introduced in 2001 by Paul Viola and Michael Jones through their seminal work, 'Rapid Object Detection using a Boosted Cascade of Simple Features', this model stands out for its speed and efficiency. Similar to a basic CNN, it sifts through images to extract a plethora of features. The most effective features are chosen using the Adaboost algorithm. However, deploying all these features across a sliding window would still demand considera ble time. To address this, they devised a Cascade of Classifiers approach where features are categorized into groups. It's important to notice that the model is better at finding frontal faces.

* **MTCNN**

Introduced by Kaipeng Zhang, et al. in 2016 in their paper, *Joint Face Detection and Alignment Using Multi-task Cascaded Convolutional Networks*. It not only detects the face but also detects five key points as well. It uses a cascade structure with three stages of CNN. First, a fully convolutional network is used to obtain candidate windows and their bounding box regression vectors. Next, these candidates are passed to another CNN which rejects a large number of false positives and performs calibration of bounding boxes. In the final stage, the facial landmark detection is performed.

* **cvlib**

cvlib stands as a straightforward, accessible, and high-level open-source Computer Vision library designed for Python. Crafted with the intention to facilitate swift and straightforward experimentation, cvlib draws significant inspiration from the well-regarded Keras library—a deep learning framework that operates atop TensorFlow. This influence is evident in cvlib’s guiding principles, emphasizing user-friendliness and rapid testing in the realm of computer vision.

* **DNN Face Detector in OpenCV**

This detector utilizes a Caffe model grounded in the Single Shot Multibox Detector (SSD) framework and employs the ResNet-10 architecture as its foundational backbone. Introduced in the aftermath of OpenCV 3.3, it marks a significant enhancement within its deep neural network (DNN) module. Notably, this model excels in achieving a balance between speed and accuracy, making it suitable for real-time applications. Its integration into OpenCV's DNN module simplifies the process of implementing advanced face detection capabilities without the need for external dependencies, thereby democratizing access to powerful computer vision technology

2)Results & Analysis

For each picture and for each model, the number of faces detected by the model are plotted in an hystogram. Then, the number of faces detected by the models are compared to the real number of faces in the picture (extracted from WIDER labels).

Immagine che contiene Diagramma, linea, diagramma, schermata

Descrizione generata automaticamente

Figure 1 - Histograms for the first 13 images

Immagine che contiene schermata, Diagramma, linea, diagramma

Descrizione generata automaticamente

Figure 2 Histograms for the last 14 images

To properly compare performances of face detection methods we should consider also the position of the faces detected, and so the bounding boxes coordinates. In fact, the number of faces detected alone is not sufficient to prove that a model is good.

However, since the final goal is to use the detection model to perform further operations with the faces detected, we can also compare the models by simply observing the results obtained for different pictures.

**Orange boxes : Haar - Green boxes : MTCNN - Red boxes : cvlib - Purple boxes : DNN**

Immagine che contiene persona

Descrizione generata automaticamenteImmagine che contiene persona

Descrizione generata automaticamenteImmagine che contiene persona, interni

Descrizione generata automaticamenteImmagine che contiene persona

Descrizione generata automaticamente

Immagine che contiene aria aperta, vestiti, persona, albero

Descrizione generata automaticamenteImmagine che contiene albero, persona, esterni, cielo

Descrizione generata automaticamenteImmagine che contiene persona, esterni, albero, cielo

Descrizione generata automaticamenteImmagine che contiene persona, esterni, albero, cielo

Descrizione generata automaticamente

Immagine che contiene persona, vestiti, Viso umano, conforto

Descrizione generata automaticamenteImmagine che contiene persona

Descrizione generata automaticamenteImmagine che contiene persona, vestiti, Viso umano, conforto

Descrizione generata automaticamenteImmagine che contiene persona, interni

Descrizione generata automaticamente

Immagine che contiene persona, edificio, persone, gruppo

Descrizione generata automaticamenteImmagine che contiene persona, vestiti, Viso umano, uomo

Descrizione generata automaticamenteImmagine che contiene persona, esterni, persone, gruppo

Descrizione generata automaticamenteImmagine che contiene vestiti, persona, Viso umano, donna

Descrizione generata automaticamente

For each approach, we can state a sort of Accuracy: this measure is obtained by summing the total number of faces detected by each model and divided by the real number of faces in every picture. Obviously, I am not considering that there might be false positives that can raise this measure. However, by doing this I can have a clue on which method can localize the highest number of faces.

--- Accuracies ---

* Haar: 0.7515151515151515
* MTCNN: 0.9636363636363636
* cvlib: 0.3696969696969697
* DNN: 0.7757575757575758

It’s interesting also to measure the time required by each method to perform the computation of all the pictures:

--- Time ---

* Haar: 30.2 s
* MTCNN: 55.3 s
* cvlib: 2.9 s
* DNN: 8.52 s

Accuracy Graph:

Immagine che contiene testo, schermata, diagramma, Diagramma

Descrizione generata automaticamente

Time:

Immagine che contiene testo, schermata, diagramma, Rettangolo

Descrizione generata automaticamente

3)Flow Chart

Immagine che contiene testo, Biglietto Post-it, schermata, design

Descrizione generata automaticamente

The process is very linear and sequetial. For each of the classifiers the scheme is similar, and it follows this type of sequence:

Once I load the dataset, for each image:

for i in range(dataset\_size):

Firstly, I convert the image to a grayscale image:

    img = cv2.imread("images/"+str(i)+".jpg")

    gray = cv2.cvtColor(img, cv2.COLOR\_BGR2GRAY)

Then, taking as example the Haar cascade classifier, that I load with:

face\_cascade = cv2.CascadeClassifier('models/haarcascade\_frontalface\_default.xml')

I can classify the image as it follows:

    face\_info = face\_cascade.detectMultiScale(gray, 1.1, 4)

Drawing the rectangle:

    haar\_numfaces[i] = len(face\_info)

    # For each face detected we draw a bounding box to localize it

    for faces in face\_info:

        x,y,width,height = faces

        cv2.rectangle(img,(x,y),(x+width,y+height),(0, 165, 255),5)

And finally returning the new Image:

    cv2.imwrite("haar\_output/haar\_"+str(i)+".jpg",img)

4)Code

This time the code is not too long, so I can just Paste it here:

!pip install cvlib

!pip install mtcnn

import cv2

import numpy as np

import cvlib as cv

from mtcnn import MTCNN

import matplotlib.pyplot as plt

import pandas as pd

from random import randrange

# Number of pictures used to compare performance

dataset\_size = 27

#inizializing arrays for each classifier

haar\_numfaces = np.zeros(dataset\_size, dtype=int)

mtcnn\_numfaces = np.zeros(dataset\_size, dtype=int)

cvlib\_numfaces = np.zeros(dataset\_size, dtype=int)

dnn\_numfaces = np.zeros(dataset\_size, dtype=int)

#uploading Images,and documents I need (as Haarcascade\_frontalface\_default.xml and deploy.prototxt.txt and res\_\_caffemodel

from google.colab import files

uploaded = files.upload()

# The model is initialized

face\_detector = MTCNN()

%%time

print("MTCNN processing...")

for i in range(dataset\_size):

    # Image is read and converted to RGB

    img = cv2.cvtColor(cv2.imread(str(i)+".jpg"), cv2.COLOR\_BGR2RGB)

    # Face detection is performed

    face\_info = face\_detector.detect\_face s(img)

    # print("Image "+str(i)+" processing...")

    mtcnn\_numfaces[i] = len(face\_info)

    # For each face detected we draw a bounding box to localize it

    for faces in face\_info:

        x,y,width,height = faces["box"]

        cv2.rectangle(img,(x,y),(x+width,y+height),(50,205,50),5)

    img = cv2.cvtColor(img, cv2.COLOR\_RGB2BGR)

    # The image with bounding boxes is saved

    cv2.imwrite("mtcnn\_output/mtcnn\_"+str(i)+".jpg",img)

# Load weights of the cascade of classifiers from xml file

face\_cascade = cv2.CascadeClassifier('haarcascade\_frontalface\_default.xml')

%%time

print("Haar Cascade processing...")

for i in range(dataset\_size):

    img = cv2.imread(str(i)+".jpg")

    if img is not None and img.size > 0:

      gray = cv2.cvtColor(img, cv2.COLOR\_BGR2GRAY)

    else:

      print(f" {i}.jpg unredable")

    # Face detection is started with scaleFactor = 1.1 and minNeighbors = 3

    face\_info = face\_cascade.detectMultiScale(gray, 1.1, 4)

    # print("Image "+str(i)+" processing...")

    haar\_numfaces[i] = len(face\_info)

    # For each face detected we draw a bounding box to localize it

    for faces in face\_info:

        x,y,width,height = faces

        cv2.rectangle(img,(x,y),(x+width,y+height),(0, 165, 255),5)

    # The image with bounding boxes is saved

    if img is not None and img.size > 0:

      cv2.imwrite("haar\_output/haar\_"+str(i)+".jpg", img)

    else:

      print(f"{i} empty")

%%time

print("cvlib processing...")

for i in range(dataset\_size):

    # Image is read

    img = cv2.imread(str(i)+".jpg")

    # Face detection is performed

    face\_info, \_ = cv.detect\_face(img)

    # print("Image "+str(i)+" processing...")

    cvlib\_numfaces[i] = len(face\_info)

    # For each face detected we draw a bounding box to localize it

    for faces in face\_info:

        x,y,z,k = faces

        cv2.rectangle(img,(x,y),(z,k),(0,0,255),5)

    # The image with bounding boxes is saved

    cv2.imwrite("cvlib\_output/cvlib\_"+str(i)+".jpg",img)

# The network is loaded using cv2.dnn.readNetFromCaffe and the model's layers and weights as passed its arguments.

modelFile = "res10\_300x300\_ssd\_iter\_140000.caffemodel"

configFile = "deploy.prototxt.txt"

net = cv2.dnn.readNetFromCaffe(configFile, modelFile)

%%time

print("DNN Face Detector processing...")

for i in range(dataset\_size):

    # Image is read

    img = cv2.imread(str(i)+".jpg")

    # Height and width of the image are extracted

    h, w = img.shape[:2]

    # To achieve the best accuracy I ran the model on BGR images resized to 300x300

    # applying mean subtraction of values (104, 177, 123) for each blue, green and red channels correspondingly.

    blob = cv2.dnn.blobFromImage(cv2.resize(img, (300, 300)), 1.0,(600, 600), (104.0, 117.0, 123.0))

    net.setInput(blob)

    # Face detection is performed

    faces = net.forward()

    # print("Image "+str(i)+" processing...")

    # For each face detected we draw a bounding box to localize it

    for j in range(faces.shape[2]):

        confidence = faces[0, 0, j, 2]

        if confidence > 0.5:

            dnn\_numfaces[i] = dnn\_numfaces[i] + 1

            box = faces[0, 0, j, 3:7] \* np.array([w, h, w, h])

            (x, y, x1, y1) = box.astype("int")

            cv2.rectangle(img, (x, y), (x1, y1), (177,36,199), 5)

    # The image with bounding boxes is saved

    cv2.imwrite("dnn\_output/dnn\_"+str(i)+".jpg",img)

# Real number of faces are extracted from the text file

real\_numfaces = np.loadtxt("real\_labels.txt",dtype="int")

# In these dataframes are stored the number of faces detected by the models (and the real number too)

# For the first 13 images

df1 = pd.DataFrame(

    {

        "Correct": real\_numfaces[0:13],

        "Haar": haar\_numfaces[0:13],

        "MTCNN": mtcnn\_numfaces[0:13],

        "cvlib": cvlib\_numfaces[0:13],

        "DNN": dnn\_numfaces[0:13]

    }

)

# For the last 14 images

df2 = pd.DataFrame(

    {

        "Correct": real\_numfaces[13:],

        "Haar": haar\_numfaces[13:],

        "MTCNN": mtcnn\_numfaces[13:],

        "cvlib": cvlib\_numfaces[13:],

        "DNN": dnn\_numfaces[13:]

    }

)

df2.index = range(13, 13 + len(df2))

df1.plot(kind='bar', figsize=(15, 5))

plt.title('Face Detection Comparison for Images 0-12')

plt.show()

df2.plot(kind='bar', figsize=(15, 5))

plt.title('Face Detection Comparison for Images 13-26')

plt.show()

total\_faces = np.sum(real\_numfaces)

haar\_acc = np.sum(haar\_numfaces)/total\_faces

mtcnn\_acc = np.sum(mtcnn\_numfaces)/total\_faces

cvlib\_acc = np.sum(cvlib\_numfaces)/total\_faces

dnn\_acc = np.sum(dnn\_numfaces)/total\_faces

print("--- Accuracies ---")

print("Haar: "+str(haar\_acc))

print("MTCNN: "+str(mtcnn\_acc))

print("cvlib: "+str(cvlib\_acc))

print("DNN: "+str(dnn\_acc))

labels = ['Haar', 'MTCNN', 'CVLib', 'DNN']

values = [haar\_acc, mtcnn\_acc, cvlib\_acc, dnn\_acc]

plt.bar(labels, values)

plt.xlabel('methods')

plt.ylabel('Accuracy')

plt.title('Face detection accuracy by method')

plt.show()

plt.bar(labels, values)

plt.xlabel('methods')

plt.ylabel('Time')

plt.title('Face detection time by method')

plt.show()

def showImagesHorizontally(list\_of\_files):

    fig = plt.figure(figsize=(20, 20), dpi=80)

    number\_of\_files = len(list\_of\_files)

    for i in range(number\_of\_files):

        a=fig.add\_subplot(1,number\_of\_files,i+1)

        image = plt.imread(list\_of\_files[i])

        print(image)

        plt.imshow(image)

        plt.axis('off')

r = randrange(27)

showImagesHorizontally(["/content/haar\_output/haar\_"+str(r)+".jpg","mtcnn\_output/mtcnn\_"+str(r)+".jpg","cvlib\_output/cvlib\_"+str(r)+".jpg","dnn\_output/dnn\_"+str(r)+".jpg"])