**2D FACE DETECTION**

**A. Dataset**

The dataset used is sourced from Kaggle and contains 133 images with anywhere from a few to many faces in a single frame. This dataset provides an objective basis for evaluating the face detection capabilities of the models under various conditions, including different lighting, angles, and face counts.

**B. Models**

The models are all used from the deepface library. DeepFace AI is the most lightweight face recognition and facial attribute analysis library for Python. The open-sourced DeepFace library includes all leading-edge AI models for modern face recognition and automatically handles all procedures for facial recognition in the background. In this report, I use 3 models are OpenCv, MtCnn and Yolo.

**I. OpenCv**

OpenCV (short for Open Source Computer Vision Library) is an open source library specialized in image processing and computer vision. The technology provides tools and libraries to analyze and process images and videos, from identifying objects in images to detecting faces or tracking other movements. The OpenCV system provides powerful algorithms and tools used to detect and analyze images. In addition, the tool is also widely used in many fields related to computer vision, robotics and automated vehicles... OpenCV is supported on many different programming languages ​​such as C++, Python, Java and MATLAB. It runs under the most popular operating systems, such as GNU/Linux, OS X, Windows, Android, iOS, etc. This allows developers to use OpenCV in many different programming environments, depending on their preferences and specific project requirements. The OpenCV library contains over 2500 algorithms, extensive documentation, source code, and sample code for real-time computer vision. Developers using the Python package and Python libraries can integrate OpenCV into their projects with commands like “python opencv.” Package managers facilitate this integration, which provides a simple process for installation and version control.

OpenCV uses the Haar Cascade Classifier for face detection. This method relies on machine learning to train a cascade function with a set of positive and negative images.

1. Haar features

Haar features are simple rectangular patterns used to describe the light intensity of different areas in an image. Each feature is represented by the difference between the total pixel intensities in the light and dark regions.

A math equations and formulas

Description automatically generated with medium confidence

**Optimization Using Integral Image:**

To speed up calculations, Haar Cascade uses an "Integral Image" (IIIIII) to quickly compute pixel intensity sums over any region.

A math equations on a white background

Description automatically generated

2. Sliding window

The **Sliding Window** algorithm is used to systematically search for objects within an image. The algorithm slides a fixed-size window across the entire image, applying a classifier to each window to determine whether it contains the object of interest. The process begins by initializing parameters such as the window size and stride. The window size is defined by its width and height, and the stride determines how far the window moves between each step. The input image has a defined width and height.

At each step, the window moves horizontally and vertically across the image. The position of the window is updated according to a set step size. The window moves across the entire image, checking each possible location. The specific positions of the window are calculated using the following formulas:

A white sheet of paper with black text

Description automatically generated

At each position, a sub-region of the image corresponding to the current window is extracted. This sub-region is then passed to a classifier, such as a Haar classifier, to determine whether it contains the object of interest. If the classifier detects the object with a probability greater than a predefined threshold, a rectangle is drawn around the window to highlight the detection. This step is represented by:

A white background with black text

Description automatically generated

3. **Adaboost Algorithm: Weight Adjustment**

Haar Cascade uses the Adaboost algorithm to select the most important Haar features from a large set.

A math equations on a white background

Description automatically generated

A math equations on a white background

Description automatically generated

4. **Cascade Classifier: Multi-Stage Detection**

A Cascade Classifier consists of a series of classifiers arranged in stages. The idea is to quickly discard regions unlikely to contain a face at early stages and refine detection at later stages.

**General Formula:**

A math equation with black text

Description automatically generated

A math equation with black text

Description automatically generated

5. The OpenCV Implementation in deepface

- **Cascade Classifier Initialization**:

* The \_\_build\_cascade method loads the Haar Cascade models.
* Two classifiers are used: haarcascade\_frontalface\_default.xml (face detection) and haarcascade\_eye.xml (eye detection).

- **Face Detection**:

* The detectMultiScale3 method applies the Haar Cascade classifier to the image, identifying bounding boxes of faces.

A black text on a white background

Description automatically generated

**- Eye Alignment**:

* The algorithm detects eyes within the identified face bounding box.
* The center of each eye is calculated for alignment purposes.

**6. Advantages:**

- Lightweight and computationally efficient.

- Suitable for real-time face detection on low-power devices.

**7. Limitations:**

- Sensitivity to lighting conditions and non-frontal faces.

- Lower accuracy compared to deep learning-based methods

**II. MtCnn**

MTCNN (Multi-task Cascaded Convolutional Networks) là một trong những phương pháp phổ biến để nhận diện khuôn mặt trong các bài toán thị giác máy tính. Nó sử dụng một mạng nơ-ron tích chập đa nhiệm (multi-task) để thực hiện ba nhiệm vụ chính: phát hiện khuôn mặt, xác định các điểm đặc trưng của khuôn mặt (ví dụ như mắt, mũi, miệng), và căn chỉnh khuôn mặt sao cho các điểm này đều được xác định chính xác.

Thuật toán MTCNN hoạt động thông qua ba bước chính:

**1. P-Net (Proposal Network)**

P-Net là mạng đầu tiên trong MTCNN, mục đích chính của nó là tìm ra các vùng có khả năng chứa khuôn mặt. P-Net thực hiện hai tác vụ chính:

* Phân loại: Kiểm tra xem một khu vực có phải là khuôn mặt hay không.
* Hồi quy (Bounding box regression): Điều chỉnh vùng phát hiện khuôn mặt sao cho chính xác hơn.

**Công thức và tác vụ:**

* Phân loại (Classification): P-Net sử dụng một mạng nơ-ron tích chập để phân loại mỗi vùng ứng với khuôn mặt hoặc không phải khuôn mặt.
  + Đầu vào: Một ảnh con của vùng được quét.
  + Đầu ra: Một xác suất p, nếu p > threshold, thì vùng đó là khuôn mặt.

A white background with black text

Description automatically generated

**Hồi quy bounding box (Bounding Box Regression)**: Điều chỉnh bounding box sao cho phù hợp hơn với khuôn mặt. Công thức hồi quy bao gồm việc học cách điều chỉnh các tọa độ của bounding box từ các thông số (x, y, width, height).



Sau khi tính toán được các hiệu chỉnh (delta), ta có thể điều chỉnh các tọa độ bounding box của đối tượng theo:

A black and white text

Description automatically generated with medium confidence

**2. R-Net (Refine Network)**

R-Net nhận các đề xuất từ P-Net và làm nhiệm vụ tinh chỉnh các bounding box, đồng thời phân loại lại các vùng và dự đoán lại các điểm đặc trưng (keypoints) của khuôn mặt.

**Công thức và tác vụ:**

* **Phân loại lại**: R-Net sử dụng một lớp phân loại để xác định lại khả năng khuôn mặt.

A black and white text

Description automatically generated with medium confidence

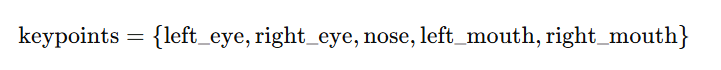
Giống như P-Net, R-Net cũng sử dụng hàm kích hoạt sigmoid để phân loại lại các khu vực phát hiện khuôn mặt.

**Hồi quy bounding box (Bounding Box Regression)**: Giống như P-Net, R-Net cũng sẽ tinh chỉnh các bounding box đã được đề xuất từ P-Net.

A math equations and formulas

Description automatically generated with medium confidence

**Dự đoán các điểm đặc trưng**: R-Net cũng dự đoán các điểm đặc trưng của khuôn mặt (ví dụ: mắt, mũi, miệng). Công thức này thực hiện hồi quy để dự đoán vị trí chính xác của các điểm đặc trưng:



**3. O-Net (Output Network)**

O-Net là mạng cuối cùng trong MTCNN, có nhiệm vụ đưa ra các kết quả cuối cùng về bounding box và các điểm đặc trưng của khuôn mặt. O-Net thực hiện các tác vụ giống như R-Net nhưng ở mức độ cao hơn với độ chính xác và chi tiết hơn.

**Công thức và tác vụ:**

* **Phân loại lại**: O-Net phân loại lại khuôn mặt, giống như các mạng trước.

A white paper with black text and symbols

Description automatically generated

4. **Multi-task Loss**

Combines face classification, bounding box regression, and landmark localization into a single optimization process, enhancing efficiency and accuracy.

5. Advantages

- High precision in detecting faces and landmarks, even in complex images.

- Robust against variations in pose, lighting, and occlusion.

6. Limitations

- Requires significant computational resources.

- Slower than single-stage detectors like YOLO.

**III. Yolo**

YOLO (You Only Look Once) là một trong những phương pháp phát hiện đối tượng mạnh mẽ và nhanh chóng trong lĩnh vực thị giác máy tính, được sử dụng rộng rãi trong nhận diện khuôn mặt và các tác vụ nhận dạng đối tượng. Trong mã nguồn từ DeepFace, YOLO được sử dụng để phát hiện khuôn mặt trong các hình ảnh với các phiên bản khác nhau của mô hình YOLO (V8N, V11N, V11S, V11M). Dưới đây là sự giải thích chi tiết về cách YOLO được tích hợp và hoạt động trong DeepFace.

**1. Các phiên bản của YOLO trong DeepFace**

DeepFace sử dụng nhiều phiên bản YOLO khác nhau để nhận diện khuôn mặt, bao gồm:

* V8N (YOLOv8n-face): Phiên bản YOLOv8 nhỏ gọn, được tối ưu hóa cho việc phát hiện khuôn mặt với tốc độ nhanh.
* V11N (YOLOv11n-face): Phiên bản YOLOv11 nhỏ gọn.
* V11S (YOLOv11s-face): Phiên bản YOLOv11 trung bình.
* V11M (YOLOv11m-face): Phiên bản YOLOv11 với mô hình lớn hơn, có khả năng phát hiện khuôn mặt chính xác hơn trong các điều kiện phức tạp.

**2. Quy trình phát hiện khuôn mặt với YOLO**

Trong mã của DeepFace, lớp YoloDetectorClient được sử dụng để phát hiện khuôn mặt trong hình ảnh bằng cách sử dụng các mô hình YOLO đã được huấn luyện sẵn.

**2.1 Tải mô hình YOLO**

Mô hình YOLO sẽ được tải thông qua phương thức build\_model, sử dụng ultralytics.YOLO để tải mô hình YOLO và trọng số từ các URL cụ thể. Trọng số của mô hình sẽ được tải về nếu chưa có sẵn trong hệ thống.

**2.2 Phát hiện khuôn mặt**

YOLO phân loại một ô lưới (grid cell) trên ảnh thành các đối tượng. Với mỗi ô lưới, YOLO dự đoán:

* C1: Xác suất của một đối tượng thuộc lớp (trong trường hợp này là "khuôn mặt").
* C2: Xác suất của đối tượng thuộc lớp "mặt" trong trường hợp có mặt.

**A white text with black text

Description automatically generated**

**Công thức hồi quy bounding box:**

Đối với mỗi ô lưới, YOLO sẽ dự đoán một bounding box, được mô tả bởi các tham số sau:

**A white paper with black text

Description automatically generated**

**2.3 Trích xuất thông tin từ kết quả**

Sau khi YOLO phát hiện các khuôn mặt, kết quả sẽ bao gồm thông tin về bounding box, điểm đặc trưng của khuôn mặt (như mắt trái và mắt phải) và độ tin cậy của mỗi phát hiện.

* Bounding box: Lấy các giá trị x, y, w, h từ kết quả phát hiện để xác định vị trí và kích thước của khuôn mặt.
* Điểm đặc trưng: Nếu mô hình phát hiện được điểm đặc trưng của khuôn mặt (như mắt), các tọa độ của mắt trái và mắt phải sẽ được lấy ra và chuyển thành kiểu dữ liệu tuple của số nguyên.

**A screen shot of a computer code

Description automatically generated**

**2.4 Tạo đối tượng FacialAreaRegion**

Sau khi thu thập thông tin về bounding box và điểm đặc trưng, một đối tượng FacialAreaRegion sẽ được tạo ra để lưu trữ các thông tin này. Đối tượng này sẽ chứa các thuộc tính như:

* Vị trí và kích thước của khuôn mặt.
* Vị trí của mắt trái và mắt phải (nếu có).
* Độ tin cậy của phát hiện.

**3. Advantages**:

- Fast, suitable for real-time applications.

- Performs well on images with large or well-defined faces.

4. **Limitations**:

- Struggles with detecting small or partially occluded faces.

- Lower accuracy compared to multi-stage methods like MTCNN.

**IV. Coparation**

three popular face detection models were compared: OpenCV, MTCNN, and YOLO. Each model was evaluated based on three key metrics: accuracy, average processing time per image, and total execution time across the entire dataset.

Độ chính xác của mô hình được đánh giá bằng cách so sánh số lượng khuôn mặt mà mô hình phát hiện với nhãn thực tế trong dataset. Mỗi ảnh trong dataset sẽ được mô hình dự đoán số lượng khuôn mặt và so sánh với số lượng khuôn mặt thực tế có trong file PersonData.txt.

****

- all\_labels là danh sách chứa nhãn thực tế (số khuôn mặt trong ảnh, được lưu trữ trong ground\_truth).

- all\_predictions là danh sách chứa số lượng khuôn mặt mà mô hình dự đoán được.

Thời gian xử lý mỗi ảnh được tính bằng cách đo thời gian từ khi mô hình bắt đầu phát hiện khuôn mặt cho đến khi hoàn thành. Thời gian này được ghi nhận trong biến elapsed\_time. Tổng thời gian cho toàn bộ dataset được cộng dồn trong biến total\_time.

**A white background with black text

Description automatically generated**

**-** start\_time lưu lại thời điểm bắt đầu đo thời gian.

- elapsed\_time là thời gian mà mô hình mất để phát hiện khuôn mặt trong một ảnh, tính bằng cách lấy thời điểm hiện tại trừ đi start\_time.

Tổng thời gian được tính bằng cách cộng dồn elapsed\_time của mỗi ảnh trong dataset:

****

Cuối cùng, thời gian trung bình mỗi ảnh được tính bằng cách chia total\_time cho tổng số ảnh:

****

**OpenCV Model**

* Accuracy: 0.46
* Average time per image: 0.6387 seconds
* Total time: 84.9494 seconds

The OpenCV model achieved the lowest accuracy, at 46%. However, its processing time per image is quite fast, averaging 0.6387 seconds. The total execution time for the entire dataset was 84.9494 seconds, indicating that although its accuracy is lower, OpenCV can process images quickly.

**MTCNN Model**

* Accuracy: 0.80
* Average time per image: 0.6590 seconds
* Total time: 87.6488 seconds

MTCNN achieved a significantly higher accuracy of 80%, compared to OpenCV. Its average processing time per image is 0.6590 seconds, which is quite close to OpenCV. However, the total execution time for the entire dataset is 87.6488 seconds, slightly higher than OpenCV. Nonetheless, MTCNN demonstrates a substantial improvement in accuracy, despite similar processing times.

**YOLO Model**

* Accuracy: 0.81
* Average time per image: 0.2184 seconds
* Total time: 29.0447 seconds

YOLO achieved the highest accuracy at 81%. The major advantage of YOLO is its exceptionally fast processing time per image, averaging only 0.2184 seconds, significantly faster than both OpenCV and MTCNN. The total execution time for the entire dataset is 29.0447 seconds, which is much lower than the other two models. This shows that YOLO not only offers high accuracy but also processes images much faster.

**V. Summary**

In this experiment, three face detection models OpenCV, MTCNN, and YOLO were evaluated based on accuracy and processing speed. The OpenCV model achieved the lowest accuracy of 46%, but it processed images relatively quickly, with an average time of 0.6387 seconds per image. In comparison, the MTCNN model demonstrated significantly better accuracy at 80%, though its processing time was slightly higher, averaging 0.6590 seconds per image. Finally, the YOLO model outperformed the others in both accuracy and speed, with an accuracy of 81% and an average processing time of just 0.2184 seconds per image. Overall, YOLO emerged as the best-performing model, offering a good balance between accuracy and speed, making it the most suitable choice for real-time face detection applications.