Face Recognition Using MTCNN

Name : Shaik Anwar
Email : shaik_anwar@srmap.edu.in

Contents:

1. Abstract

This project gives an ideal way for Face detection and Face recognition by using Python Programming Language, OpenCV (CV: Computer Vision), and the Deep Learning technique CNN algorithm. A face recognition system is a complex image-processing problem in real-world applications with complex effects of illumination, occlusion, and imaging condition on live images. This model is usually applied and preferred for people and security cameras in metropolitan life, for crime prevention, video surveillance, person verification, and similar security activities. We have implemented MTCNN Algorithm in Convolutional Neural Network [CNN]. This model recognizes various features of image or video frames. CNN model can detect and recognize faces more accurately. Finally, in this report, we have discussed how the MTCNN model will recognize a face.

Keywords: Face detection and Face recognition, CNN, OpenCV, MTCNN Algorithm.

2. Introduction

A face recognition system is a powerful amalgamation of detection and recognition techniques that captures facial images and transforms them into unique face prints. This system proves invaluable for various applications, including security systems, person verification, and enhancing security measures in public spaces to identify potential criminals and prevent criminal activities.

In this context, we'll explore the role of MTCNN (Multi-task Cascaded Convolutional Networks) as the primary detection algorithm, which efficiently locates and isolates faces within a given image. Following this, we delve into the recognition phase, which utilizes recognition algorithms to classify these detected images based on distinctive facial features – a fundamental element in many computer vision applications.

Compared to single detection or recognition approaches, this combination of detection and recognition adds complexity to the system, as detection algorithms must pinpoint faces and extract facial components like eyes, eyebrows, nose, and mouth.

The face recognition system typically follows a sequential process, starting with image acquisition from a camera and a dataset. The system's initial step involves the automatic detection of faces within the acquired image using the MTCNN algorithm. Once faces are successfully detected, the next step is face recognition, wherein the system processes the extracted facial images. The final stage involves identifying the individuals through the recognition process.

To initiate the face recognition system, the first step is acquiring digital images from a camera and transferring them to a computer. Subsequently, these images are processed through a frame grabber and presented to the face detection algorithm. Numerous face detection strategies are available, but in this context, we emphasize the utilization of the MTCNN algorithm, known for its exceptional performance in detecting faces within images.

Upon successful detection of faces, the recognition phase comes into play, where Convolutional Neural Networks (CNN), specifically configured with the MTCNN algorithm, play a pivotal role. There are various neural network libraries available for this purpose, such as TensorFlow, TFlearn, and Keras. In this model, we focus on leveraging MTCNN, TensorFlow, and TFlearn to construct a robust face recognition system.

3. Proposed Scheme

Convolutional Neural Networks (CNNs):

The goal of a CNN is to learn higher-order features in the data via convolutions. They are well suited to object recognition with images and consistently top image classification competitions. They can identify faces, individuals, street signs, platypuses, and many other aspects of visual data. CNN overlap with text analysis via optical character recognition, but they are also useful when analysing words as discrete textual units. They're also good at analysing sound. The efficacy of CNNs in image recognition is one of the main reasons why the world recognizes the power of deep learning. CNN are good at building position and rotation invariant features from raw image data. CNN are powering major advances in machine vision, which has obvious applications for self-driving cars, robotics, drones, and treatments for the visually impaired

CNN and Structure in Data:

CNN’s tend to be most useful when there is some structure to the input data. An example would be how images and audio data that have a specific set of repeating patterns and input values next to each other are related spatially. Conversely, the columnar data exported from a relational database management system (RDBMS) tends to have no structural relationships spatially. Columns next to one another just happen to be materialized that way in the database exported materialized view.

CNN Architecture Overview :

CNN transforms the input data from the input layer through all connected layers into a set of class scores given by the output layer. There are many variations of the CNN architecture, but they are based on the pattern of layers.

1. **Convolutional Layer(Kernels):** Convolutional filters (also known as kernels) are small 2D matrices that slide over the input image. Each filter performs a weighted sum of pixel values in its receptive field to extract specific features. YOLOv4 employs multiple convolutional filters with varying sizes and depths to capture features at different scales and complexities.
2. **Pooling Layer:** These layers find several features in the images and progressively construct higher-order features. This corresponds directly to the ongoing theme in deep learning by which features are automatically learned as opposed to traditionally hand engineered. Finally, we have the classification layers in which we have one or more fully connected layers to take the higher-order features and produce class probabilities or scores. The output of these layers produces typically a two-dimensional output of the dimensions [b x N], where b is the number of examples in the mini-batch and N is the number of classes we're interested in scoring
3. **Activation Functions:** Activation functions like ReLU (Rectified Linear Unit) are applied after convolution operations to introduce non-linearity to the network. ReLU is commonly used to introduce non-linearity and facilitate feature learning.
4. **Max-Pooling Layers:** Max-pooling layers down sample feature maps by selecting the maximum value within a fixed-size window. Max-pooling helps reduce the spatial dimensions of the feature maps while retaining essential information.
5. **Fully Connected Layer:** A fully Connected Layer is simply, a feed-forward neural network. Fully Connected Layers form the last few layers in the network. The input to the fully connected layer is the output from the final Pooling or Convolutional Layer, which is flattened and then fed into the fullyconnected layer.

4. Theory behind the project

Facial recognition hinges on three fundamental principles:

- Dataset Preparation
- Training of the Model
- Testing of the Model

This model empowers facial recognition in two distinct modes, with a primary focus on live face recognition using the device's webcam.

In live face recognition, the system interfaces with the device's webcam to process real-time input. It swiftly detects faces, delineates them with bounding boxes, and extracts essential facial features. These features, represented by 128 nodes, serve as reference points for identifying individuals. The next step involves comparing these reference images to those stored in the dataset, thereby identifying the person in question.

To achieve this identification, the model relies on a robust set of comparisons, scrutinizing the input image against the dataset to ascertain identity. Notably, the model employs the MTCNN algorithm for face detection, renowned for its accuracy in real-time face recognition applications. In addition, it utilizes popular deep learning techniques, including CNN, in conjunction with libraries like Haar Cascades and TensorFlow to enhance its performance.

In summary, this model adheres to three core principles—dataset preparation, model training, and model testing—and excels in live face recognition by employing the MTCNN algorithm for face detection.

Working of MTCNN:

MTCNN consists of three cascaded networks, each responsible for specific tasks.

Stage 1: Proposal Network (P-Net):

First, the image is resized multiple times to detect faces of different sizes. Then the P-network (Proposal) scans images, performing first detection. It has a low threshold for detection and therefore detects many false positives, even after NMS (Non-Maximum Suppression), but works like this on purpose.

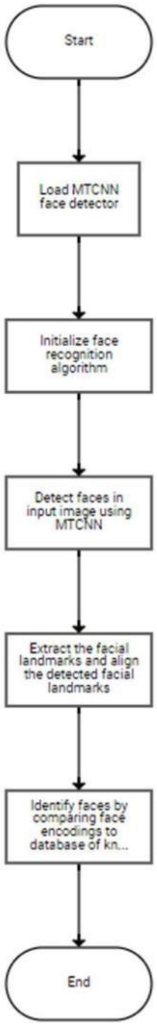
Stage 2: Refinement Network (R-Net):

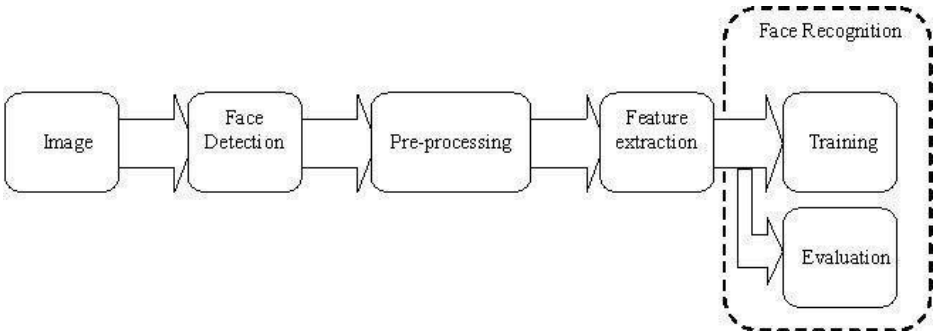
The proposed regions (containing many false positives) are input for the second network, the R-network (Refine), which, as the name suggests, filters detections (also with NMS) to obtain quite precise bounding boxes.

Stage 3: Output Network (O-Net):

The final stage, the O-network (Output) performs the final refinement of the bounding boxes. This way not only faces are detected, but bounding boxes are very right and precise.

5. Algorithm and Flowchart





6) Program:

```
import os
import cv2
import time
from facenet_pytorch import MTCNN, InceptionResnetV1
import torch
from PIL import Image
import skfuzzy as fuzz
import numpy as np

# Initialize MTCNN and InceptionResnetV1
mtcnn0 = MTCNN(image_size=240, margin=0, keep_all=False, min_face_size=40)
mtcnn = MTCNN(image_size=240, margin=0, keep_all=True, min_face_size=40)
resnet = InceptionResnetV1(pretrained='vggface2').eval()

# Initialize empty lists to store embeddings and names
embedding_list = []
name_list = []

# Process images in the photos folder
photos_folder = "photos"  # Path to the folder containing images
for filename in os.listdir(photos_folder):
    image_path = os.path.join(photos_folder, filename)

    if image_path.lower().endswith(('.png', '.jpg', '.jpeg')):
        # Process each image for embeddings
        img = Image.open(image_path)
        face, prob = mtcnn0(img, return_prob=True)

        if face is not None and prob > 0.92:
            emb = resnet(face.unsqueeze(0))
            embedding_list.append(emb.detach())
            name_list.append(filename.split('.')[0])  # Use the filename (without extension) as the name

# Save the data (embeddings and names)
data = [embedding_list, name_list]
torch.save(data, 'data.pt')  # Save data.pt file

# Fuzzy Logic Membership Functions for Distance and Confidence
def fuzzy_distance(distance):
    dist_range = np.linspace(0, 2, 100)

    # Distance fuzzy sets (Close, Medium, Far)
    close = fuzz.trimf(dist_range, [0, 0, 0.5])
    medium = fuzz.trimf(dist_range, [0, 0.5, 1.0])
    far = fuzz.trimf(dist_range, [0.5, 1.0, 2.0])

    # Calculate the membership values for the given distance
    close_membership = fuzz.interp_membership(dist_range, close, distance)
```

```
    medium_membership = fuzz.interp_membership(dist_range, medium, distance)
    far_membership = fuzz.interp_membership(dist_range, far, distance)

    return close_membership, medium_membership, far_membership

def fuzzy_confidence(confidence):
    conf_range = np.linspace(0, 1, 100)

    # Confidence fuzzy sets (Low, Medium, High)
    low = fuzz.trimf(conf_range, [0, 0, 0.5])
    medium = fuzz.trimf(conf_range, [0, 0.5, 1.0])
    high = fuzz.trimf(conf_range, [0.5, 1.0, 1.0])

    # Calculate the membership values for the given confidence
    low_membership = fuzz.interp_membership(conf_range, low, confidence)
    medium_membership = fuzz.interp_membership(conf_range, medium, confidence)
    high_membership = fuzz.interp_membership(conf_range, high, confidence)

    return low_membership, medium_membership, high_membership

# Load the saved data
load_data = torch.load('data.pt')
embedding_list = load_data[0]
name_list = load_data[1]

# Open the webcam
cam = cv2.VideoCapture(0)

while True:
    ret, frame = cam.read()
    if not ret:
        print("Failed to grab frame, try again")
        break

    # Convert frame to image for MTCNN processing
    img = Image.fromarray(frame)
    img_cropped_list = None  # Initialize to default
    match_probability = 0.0  # Initialize to default

    try:
        img_cropped_list, prob_list = mtcnn(img, return_prob=True)
    except Exception as e:
        print(f"Error during face detection: {e}")

     if img_cropped_list is not None:
         boxes, _ = mtcnn.detect(img)

        for i, prob in enumerate(prob_list):
            if prob > 0.90:
                emb = resnet(img_cropped_list[i].unsqueeze(0)).detach()
```

```
            # List of distances for face recognition
            dist_list = [torch.dist(emb, emb_db).item() for emb_db in embedding_list]

            min_dist = min(dist_list)  # Get minimum distance
            min_dist_idx = dist_list.index(min_dist)  # Get index of minimum distance
            name = name_list[min_dist_idx]  # Get name corresponding to the minimum distance

            # Get bounding box for detected face
            box = boxes[i]
            original_frame = frame.copy()  # Store the frame before drawing on it

            # Apply fuzzy logic to determine the matching probability
            close_membership, medium_membership, far_membership = fuzzy_distance(min_dist)
            low_membership, medium_conf_membership, high_membership =
fuzzy_confidence(prob)

            # Compute weighted average of memberships
            match_probability = (close_membership * high_membership) + \
                        (medium_membership * medium_conf_membership) + \
                        (far_membership * low_membership)

            # Scale the match probability to be in the range of [0, 1]
            match_probability = np.clip(match_probability, 0, 1)

            # Draw the name and match probability on the frame
            frame = cv2.putText(frame, f'{name} Match Probability: {match_probability:.2f}',
                        (int(box[0]), int(box[1])),
                        cv2.FONT_HERSHEY_SIMPLEX, 1, (0, 255, 0), 2, cv2.LINE_AA)

            # Draw the bounding box around the face
            frame = cv2.rectangle(frame, (int(box[0]), int(box[1])),
                         (int(box[2]), int(box[3])), (0, 255, 0), 2)

    # Display the frame
    cv2.imshow("Output", frame)

    k = cv2.waitKey(1)
    if k % 256 == 27:  # ESC key to exit
        break

    # Save the image of the recognized person if match probability is high
    if match_probability > 0.8 and 'name' in locals() and not os.path.exists(f'photos/{name}'):
        os.mkdir(f'photos/{name}')

        img_name = f"photos/{name}/{int(time.time())}.jpeg"
        cv2.imwrite(img_name, original_frame)
        print(f"Saved: {img_name}")

# Release the webcam and close windows
cam.release()
cv2.destroyAllWindows()
```

!pip install facenet-pytorch

!pip install opencv-python

7) Output:

MTCNN is very accurate and robust. It properly detects faces even with different sizes, lighting and strong rotations. It achieves about 13 FPS on the full HD videos, and even up to45 FPS on rescaled.

In this Project we have also achieved 6–8 FPS on the CPU for full HD, so real-time processingis very much possible with MTCNN.

8)Reference:

- https://towardsdatascience.com/robust-face-detection-with-mtcnn-400fa81adc2e
- https://arsfutura.com/magazine/face-recognition-with-facenet-and-mtcnn
- https://www.kaggle.com/code/msripooja/face-s-detection-using-mtcnn
- https://www.aimspress.com/article/doi/10.3934/mbe.2021329?viewType=HTML
- https://www.semanticscholar.org/paper/Face-Recognition-Based-on-MTCNN-and-FaceNet- Jin-Li/24787e7ad8e7c5e8ce7d8bf5c3fd79c2be7b418b
- https://medium.com/@iselagradilla94/multi-task-cascaded-convolutional-networks-mtcnn- for-face-detection-and-facial-landmark-alignment-7c21e8007923