Face Detection using InsightFace (SCRFD Model)

1. Introduction

This project demonstrates the use of the InsightFace SCRFD model to detect faces in both static images and video files. By leveraging ONNX-based pre-trained models and OpenCV, we developed a complete pipeline capable of detecting human faces in real-time from a video file, or uploaded image. The project is ideal for applications in security, media analytics, and smart surveillance systems.

2. Libraries and Tools Used

- **OpenCV (cv2)**: Used for reading, writing, and manipulating images. It also supports drawing functions to annotate detected faces.
- **NumPy**: Provides support for numerical operations and array manipulations.
- **Matplotlib**: Used to visualize the output image with detected faces.
- **InsightFace**: A powerful deep learning library designed for face analysis tasks. It supports detection, recognition, and alignment of facial features.
- **ONNXRuntime**: Backend engine used to execute the ONNX (Open Neural Network Exchange) models efficiently.
- **Google Colab Files**: A utility from Colab to facilitate file upload and download operations.

Face Detection in Image - Code :

```
# Step 1: Install required packages
!pip install -q insightface onnxruntime

# Step 2: Import libraries
import cv2
import numpy as np
from matplotlib import pyplot as plt
from insightface.app import FaceAnalysis
from google.colab import files

# Step 3: Upload your image
uploaded = files.upload()
image_path = list(uploaded.keys())[0]  # Use the first uploaded file
```

```
# Step 4: Load image
img = cv2.imread(image_path)
if img is None:
    raise ValueError("Image loading failed!")

# Step 5: Initialize the face detector
app = FaceAnalysis(name='buffalo_l',
providers=['CPUExecutionProvider'])  # Use 'CUDAExecutionProvider' if you
have GPU
app.prepare(ctx_id=0, det_size=(640, 640))

# Step 6: Detect faces
faces = app.get(img)

# Step 7: TEMP FIX for np.int error
import numpy as np
if not hasattr(np, 'int'):
    np.int = int

# Now draw the detections
img_with_faces = app.draw_on(img, faces)


# Step 8: Save and display result
output_filename = "face_detected_output.jpg"
cv2.imwrite(output_filename, img_with_faces)

# Display
plt.figure(figsize=(10, 6))
plt.imshow(cv2.cvtColor(img_with_faces, cv2.COLOR_BGR2RGB))
plt.axis('off')
plt.title('Detected Faces using SCRFD')
plt.show()

Download the result
files.download(output_filename)
```

3. Step-by-Step Code Explanation

Step 1: Install Required Packages

!pip install -q insightface onnxruntime

Installs insightface and onnxruntime. These are essential for loading and running the face detection model.

Step 2: Import Libraries

```
import cv2

import numpy as np

from matplotlib import pyplot as plt

from insightface.app import FaceAnalysis

from google.colab import files
```

- Loads all required libraries to handle images, arrays, visualization, and model operations.

Step 3: Upload Your Image

```
uploaded = files.upload()

image_path = list(uploaded.keys())[0]
```

- Prompts the user to upload an image and stores the filename for further use.

Step 4: Load Image

```
img = cv2.imread(image_path)

if img is None:

    raise ValueError("Image loading failed!")
```

- Reads the image using OpenCV and checks if loading was successful.

Step 5: Initialize the Face Detector

```
app = FaceAnalysis(name='buffalo_l', providers=['CPUExecutionProvider'])

app.prepare(ctx_id=0, det_size=(640, 640))
```

- Initializes the face detection model using buffalo_l configuration. The detection size is set to 640x640.

Step 6: Detect Faces

```
faces = app.get(img)
```

- Runs the detection model and stores the detected face data in faces.

Step 7: Fix for Deprecated np.int

```
if not hasattr(np, 'int'):
    np.int = int
```

- A compatibility fix for deprecated np.int in NumPy versions 1.20 and above.

Step 8: Draw Detected Faces

```
img_with_faces = app.draw_on(img, faces)
```

- Annotates the original image with bounding boxes and facial landmarks.

Step 9: Save and Display Result

```
output_filename = "face_detected_output.jpg"
cv2.imwrite(output_filename, img_with_faces)
```

- Saves the annotated image to disk.

```
plt.figure(figsize=(10, 6))
plt.imshow(cv2.cvtColor(img_with_faces, cv2.COLOR_BGR2RGB))
plt.axis('off')
plt.title('Detected Faces using SCRFD')
plt.show()
```

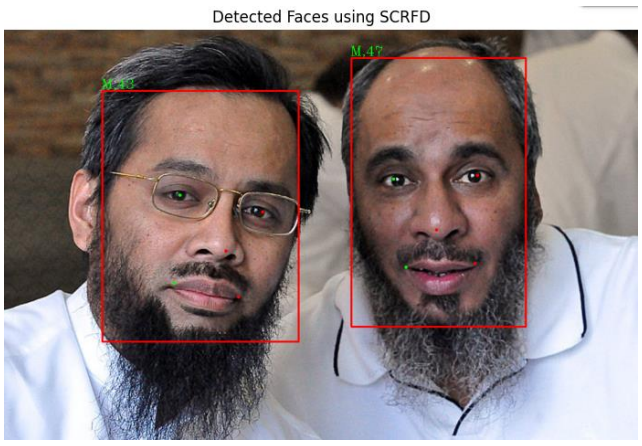
- Displays the image in Colab using matplotlib after converting BGR to RGB.

Step 10: Download the Output

```
files.download(output_filename)
```

- Allows the user to download the final image to their local machine.

Output:



Face Detection in Video - Code :

```
import cv2
import numpy as np
from insightface.app import FaceAnalysis
from google.colab.patches import cv2_imshow

# Fix for deprecated np.int (for compatibility with InsightFace)
if not hasattr(np, 'int'):
    np.int = int

# Initialize FaceAnalysis
app = FaceAnalysis(providers=['CPUExecutionProvider'])
app.prepare(ctx_id=0, det_size=(640, 640))

# Input Video
input_path = "/content/drive/MyDrive/Colab Notebooks/faceImages/Video1"  #
Replace with your file path
cap = cv2.VideoCapture(input_path)

# Output Video Writer Setup
output_path = "/content/drive/MyDrive/Colab
Notebooks/faceImages/output.mp4"
fourcc = cv2.VideoWriter_fourcc(*'mp4v')  # Codec
fps = int(cap.get(cv2.CAP_PROP_FPS))
width  = int(cap.get(cv2.CAP_PROP_FRAME_WIDTH))
height = int(cap.get(cv2.CAP_PROP_FRAME_HEIGHT))
out = cv2.VideoWriter(output_path, fourcc, fps, (width, height))

# Process Frame-by-Frame
frame_count = 0
while True:
    ret, frame = cap.read()
    if not ret:
        break

    faces = app.get(frame)
    frame = app.draw_on(frame, faces)

    out.write(frame)              # Save frame to output video
    if frame_count % 10 == 0:     # Show every 10th frame in Colab
(optional)
        print(f"Processed frame {frame_count}")
        cv2_imshow(frame)
```

```
    frame_count += 1

# Cleanup
cap.release()
out.release()
print(f"Face detection complete. Saved to: {output_path}")
```

Code Explanation:

Load the Input Video

input_path = "/content/drive/MyDrive/Colab Notebooks/faceImages/Video1"

cap = cv2.VideoCapture(input_path)

- Loads the input video using OpenCV.
- cap object is used to read frames sequentially.

Set Up the Output Video Writer

output_path = "/content/drive/MyDrive/Colab Notebooks/faceImages/output.mp4"

fourcc = cv2.VideoWriter_fourcc(*'mp4v')

fps = int(cap.get(cv2.CAP_PROP_FPS))

width = int(cap.get(cv2.CAP_PROP_FRAME_WIDTH))

height = int(cap.get(cv2.CAP_PROP_FRAME_HEIGHT))

out = cv2.VideoWriter(output_path, fourcc, fps, (width, height))

- Prepares a video writer to save the processed frames.
- fourcc: codec for MP4 format.
- fps, width, height: extracted from original video to match properties.

Process Each Frame

frame_count = 0

while True:

    ret, frame = cap.read()

    if not ret:

break

- Starts reading video frame-by-frame in a loop.
- ret is False when the video ends or a frame read fails.

Apply Face Detection & Save Frame

faces = app.get(frame)

frame = app.draw_on(frame, faces)

out.write(frame)

- app.get(frame): runs the SCRFD model to detect all faces in the frame.
- app.draw_on(frame, faces): overlays bounding boxes on the detected faces.
- out.write(frame): writes the modified frame to the output video.

Display (Every 10th Frame in Colab)

if frame_count % 10 == 0:

print(f"Processed frame {frame_count}")

cv2_imshow(frame)

- Displays every 10th frame to reduce I/O overhead.
- cv2_imshow() is used because cv2.imshow() crashes in Colab.

Cleanup

cap.release()

out.release()

print(f"Face detection complete. Saved to: {output_path}")

- Releases system resources and finalizes the video file.
- Prints success message.

Final Output

- The script will generate a new video file (output.mp4) where all faces are highlighted.
- It processes each frame independently and displays real-time progress in the console.

4. Conclusion

This project successfully demonstrates real-time face detection using the SCRFD model from InsightFace. The model is lightweight, fast, and accurate, making it ideal for a wide range of applications including security, user authentication, and human-computer interaction. The script is modular, easy to understand, and can be extended to include webcam and video processing functionalities.