Human Fall Detection using YOLO

ABSTRACT:

The "Fall Detection using YOLO (You Only Look Once)" project presents an innovative approach to real-time fall detection leveraging state-of-the-art object detection techniques. YOLO, a powerful object detection algorithm, is employed to accurately identify and locate individuals within video streams.

The project focuses on the specific application of detecting falls, a critical scenario, and combines YOLO's efficiency with fall-specific training data. The system's architecture involves preprocessing video feeds, applying YOLO for object detection, and implementing fall classification algorithms.

YOLO :

You Only Look Once (YOLO) proposes using an end-to-end neural network that makes predictions of bounding boxes and class probabilities all at once. It differs from the approach taken by previous object detection algorithms, which repurposed classifiers to perform detection.YOLO performs all of its predictions with the help of a single fully connected layer.

Methods that use Region Proposal Networks perform multiple iterations for the same image, while YOLO gets away with a single iteration. YOLO is a real-time object detection algorithm that is faster and more accurate than many other CNN-based object detection algorithms. YOLO is also able to detect multiple objects in a single image, while many other CNN-based algorithms can only detect one object at a time.

Speed: YOLO is a single-shot object detection algorithm, which means that it can detect objects in a single pass through the image. This makes YOLO much faster than other CNN-based algorithms, which typically require multiple passes through the image to detect objects.

Accuracy: YOLO is also very accurate, even on challenging datasets. In fact, YOLO has outperformed many other CNN-based object detection algorithms on standard benchmarks.

Multi-object detection: YOLO is able to detect multiple objects in a single image, while many other CNN-based algorithms can only detect one object at a time. This makes YOLO ideal for real-world applications such as self-driving cars and video surveillance.

Code:

```
import matplotlib.pyplot as plt
import torch
import cv2
import math
from torchvision import transforms
import numpy as np
import os
from tqdm import tqdm
from utils.datasets import letterbox
from utils.general import non_max_suppression_kpt
from utils.plots import output_to_keypoint, plot_skeleton_kpts
def fall_detection(poses):
    for pose in poses:
        xmin, ymin = (pose[2] - pose[4] / 2), (pose[3] - pose[5] / 2)
        xmax, ymax = (pose[2] + pose[4] / 2), (pose[3] + pose[5] / 2)
        left_shoulder_y = pose[23]
        left_shoulder_x = pose[22]
        right_shoulder_y = pose[26]
        left_body_y = pose[41]
        left_body_x = pose[40]
        right_body_y = pose[44]
        len_factor = math.sqrt(((left_shoulder_y - left_body_y) ** 2 +
(left_shoulder_x - left_body_x) ** 2))
        left_foot_y = pose[53]
        right_foot_y = pose[56]
        dx = int(xmax) - int(xmin)
        dy = int(ymax) - int(ymin)
        difference = dy - dx
        if left_shoulder_y > left_foot_y - len_factor and left_body_y >
left_foot_y - (
                len_factor / 2) and left_shoulder_y > left_body_y -
(len_factor / 2) or (
                right_shoulder_y > right_foot_y - len_factor and
right_body_y > right_foot_y - (
                len_factor / 2) and right_shoulder_y > right_body_y -
(len_factor / 2)) \
                or difference < 0:
            return True, (xmin, ymin, xmax, ymax)
    return False, None
def falling_alarm(image, bbox):
    x_min, y_min, x_max, y_max = bbox
    cv2.rectangle(image, (int(x_min), int(y_min)), (int(x_max),
int(y_max)), color=(0, 0, 255),
                  thickness=5, lineType=cv2.LINE_AA)
    cv2.putText(image, 'Fall Detected', (11, 100), 0, 1, [0, 0, 2550],
thickness=3, lineType=cv2.LINE_AA)
def get_pose_model():
    device = torch.device("cuda:0" if torch.cuda.is_available() else "cpu")
    print("device: ", device)
    weigths = torch.load('yolov7-w6-pose.pt', map_location=device)
    model = weigths['model']
```

```
    _ = model.float().eval()
    if torch.cuda.is_available():
        model = model.half().to(device)
    return model, device
def get_pose(image, model, device):
    image = letterbox(image, 960, stride=64, auto=True)[0]
    image = transforms.ToTensor()(image)
    image = torch.tensor(np.array([image.numpy()]))
    if torch.cuda.is_available():
        image = image.half().to(device)
    with torch.no_grad():
        output, _ = model(image)
    output = non_max_suppression_kpt(output, 0.25, 0.65,
nc=model.yaml['nc'], nkpt=model.yaml['nkpt'],
                                     kpt_label=True)
    with torch.no_grad():
        output = output_to_keypoint(output)
    return image, output
def prepare_image(image):
    _image = image[0].permute(1, 2, 0) * 255
    _image = _image.cpu().numpy().astype(np.uint8)
    _image = cv2.cvtColor(_image, cv2.COLOR_RGB2BGR)
    return _image
def prepare_vid_out(video_path, vid_cap):
    vid_write_image = letterbox(vid_cap.read()[1], 960, stride=64,
auto=True)[0]
    resize_height, resize_width = vid_write_image.shape[:2]
    out_video_name = f"{video_path.split('/')[-
1].split('.')[0]}_keypoint.mp4"
    out = cv2.VideoWriter(out_video_name, cv2.VideoWriter_fourcc(*'mp4v'),
30, (resize_width, resize_height))
    return out
def process_video(video_path):
    vid_cap = cv2.VideoCapture(video_path)
    if not vid_cap.isOpened():
        print('File not Found')
        return
    model, device = get_pose_model()
    vid_out = prepare_vid_out(video_path, vid_cap)
    success, frame = vid_cap.read()
    _frames = []
    while success:
        _frames.append(frame)
        success, frame = vid_cap.read()
    for image in tqdm(_frames):
        image, output = get_pose(image, model, device)
        _image = prepare_image(image)
        is_fall, bbox = fall_detection(output)
        if is_fall:
            falling_alarm(_image, bbox)
        vid_out.write(_image)
    vid_out.release()
    vid_cap.release()
```

```
if __name__ == '__main__':
    videos_path = 'fall_dataset/videos'
    for video in os.listdir(videos_path):
        video_path = os.path.join(videos_path, video)
        process_video(video_path)
```

fall_detection(poses)

Purpose: Detects falls based on pose information.

Parameters: poses - List of pose information.

Logic: Computes bounding box coordinates and keypoint information, then applies conditions to determine if a fall has occurred.

Returns: True if a fall is detected, along with the bounding box coordinates; otherwise, False, None.

falling_alarm(image, bbox)

Purpose: Adds annotations to an image for a detected fall.

Parameters: image - Input image, bbox - Bounding box coordinates.

Logic: Draws a red rectangle around the fall area and adds a text message.

Output: Modifies the input image.

get_pose_model()

Purpose: Initializes and loads the pose estimation model.

Returns: The loaded model and the device (CPU or GPU).

get_pose(image, model, device)

Purpose: Performs pose estimation on an input image using a given model.

Parameters: image - Input image, model - Pose estimation model, device - Device for computation.

Logic: Applies transformations, performs model inference, and post-processes the output.

Returns: The processed image and pose estimation output.

prepare_image(image)

Purpose: Converts a processed image back to the BGR color space and adjusts pixel values.
Parameters: image - Processed image.
Returns: The image in BGR format.

prepare_vid_out(video_path, vid_cap)

Purpose: Prepares settings for video output.
Parameters: video_path - Path to the input video, vid_cap - Video capture object.
Logic: Reads a frame to get dimensions, sets up the output video file name, and initializes a video writer.
Returns: Video writer object.

process_video(video_path)

Purpose: Processes a video by performing pose estimation, detecting falls, and creating an annotated output video.
Parameters: video_path - Path to the input video.
Logic: Iterates through video frames, performs pose estimation, detects falls, adds annotations, and writes annotated frames to an output video.

if __name__ == '__main__':

Purpose: Main execution block for processing multiple videos in a specified directory.

References :

1. https://ieeexplore.ieee.org/document/9854070
2. https://thesai.org/Downloads/Volume14No4/Paper_9-Human_Fall_Detection_for_Smart_Home_Caring.pdf
3. https://www.sciencedirect.com/science/article/pii/S1077314222000595
4. https://www.v7labs.com/blog/yolo-object-detection