Abnormal activity detection using deep learning

The backend functions that are used are Tensorflow, Pytorch and yolov5

Fully tested on the UCSD pedestrians dataset

Easy execution from anaconda prompt

Features

Detection of abnormal activities

Tracking all pedestrians with high accuracy

Generating foreground masks

Uses optical flow features

Uses convolutional saptial auto-encoder

Files organization

run .py : the pilot script that runs in order other scripts

UCSDped1 .py : contains the labeling and reorganization code for the dataset

utils .py : takes in the number of tests to be performed as an argument and generates a folder containing only the selected samples

test_script .py : loads the pretrained model results from features/ and processes the test sample

frames2video .py : a script that compiles a given sequence of images to a video

track .py : runs yolov5 scripts detect and track different classes of objects MOT (Multi-Object Tracker) the run .py restrains the process to class 0 only 'pedestrians'

bg .py : generates foreground masks as a preprocessing step for optical_flow .py

Download the UCSD dataset : UCSD_Anomaly_Dataset.v1p2

"run.py"

You can choose how many videos to process (from a special dataset) and if you want to track people in the final videos.

The script first prepares the videos using another script (like organizing and labeling).

Then, it uses a super good guesser model (another script) to find weird stuff happening in the videos.

It puts the video frames back together into videos showing these weird parts.

If you asked it to track people, it adds lines to these videos showing where people are walking.

In the end, you get new videos showing both the weird things and people walking around.

"track.py"

This code helps you track people in videos:

It gets ready to use special pre-trained models for object detection (finding people) and tracking (following them).

It takes video data (webcam, file, or folder) as input.

It goes through each frame of the video:

It uses a super good guesser model (YOLOv5) to find people in the frame.

It uses another model (DeepSort) to keep track of which person is which across frames.

It can show you the video with boxes around the people and tracks showing their movement (if you ask it to).

It can also save this video with boxes and tracks for later (if you ask it to).

"utils.py"

This code helps you pick a certain number of videos from a big dataset for anomaly detection.

You can tell it how many videos you want to use (from 1 to 36) using the -test option.

The script then copies that many videos from a specific folder in the dataset to a new folder.

"test_Script.py"

This code helps find unusual activity (anomalies) in videos. It does this by:

Using a pre-trained model to learn what normal activity looks like.

Comparing each frame of a new video to what it thinks is normal.

Highlighting anything that looks strange as an anomaly.

It outputs how well it did at finding anomalies and stores the videos with the highlighted anomalies.

The entire program:

1. Setup and Argument Parsing:

- The script accepts command-line inputs to activate tracking and specify the number of files to test.

2. Dataset Preparation:

- Organizes and labels the chosen dataset samples using a utility script.

3. Model Execution:

- Runs a pretrained model to process the dataset.

4. Output Directories:

- Prepares directories to store the results.

5. Video Conversion and Tracking:

- Converts processed frames into videos.

- Uses a script to track pedestrians in the videos.

6. Anomaly Detection:

- Defines a class to detect anomalies using a decision tree classifier.

- Loads training features and fits the classifier.

- Processes each frame of the video to detect anomalies.

- Processes the entire video and keeps track of processing time.

7. Running Anomaly Detection:

- Runs the anomaly detection on the dataset.

- Calculates precision, recall, and F1 score based on the results.

8. Object Tracking with YOLO and DeepSort:

- Integrates YOLO (You Only Look Once) for object detection and DeepSort for object tracking.

- Configures and runs the detection and tracking on the video source.

Key Steps

1. Prepare Dataset:

- Organizes dataset for testing.

2. Process with Pretrained Model:

- Runs a model to process the data.

3. Convert Frames to Video:

- Converts image frames into video files.

4. Track Objects in Video:

- Tracks pedestrians in the video using tracking algorithms.

5. Detect Anomalies:

- Uses a machine learning model to detect anomalies in the video frames.

6. Calculate Metrics:

- Computes precision, recall, and F1 score to evaluate anomaly detection performance.

7. Run YOLO and DeepSort:

- Uses YOLO for detecting objects and DeepSort for tracking these objects through video frames.

This entire process helps in identifying and tracking anomalies in videos by combining several machine learning and computer vision techniques.

"Pseudocode"

```
BEGIN

# Parse command-line arguments
arguments = parse_arguments()

# Reorganize and label dataset samples
run("python utils.py -test {arguments.test}")

# Run the test script to generate output frames
run("python test_script.py")

# Convert output frames to a video
convert_frames_to_video(input="output/frames", output="output/video.mp4", fps=25)

# If tracking is enabled, perform pedestrian tracking and anomaly detection
```

```
IF arguments.tracking THEN
    # Create results directory
    create_directory("results")

    # Run tracking script
    run("python track.py --source output/video.mp4 --yolo-weights weights/yolov5s.pt --classes 0")

    # Move tracked video to results directory
    move_file("output/video.mp4", "results/")

    # Initialize anomaly detection
    anomaly_detector = AnomalyDetection("results")

    # Process videos for anomalies
    anomaly_detector.process_videos()

    # Generate anomaly detection report
    anomaly_detector.generate_report()

    # Cleanup intermediate files
    cleanup("output/frames")
    cleanup("output/video.mp4")
    cleanup("results")
END IF

END

# Function to parse command-line arguments
FUNCTION parse_arguments()
```

```
    RETURN parse("--tracking", "--test")
END FUNCTION

# Function to run a system command
FUNCTION run(command)
    execute(command)
END FUNCTION

# Function to convert frames to a video
FUNCTION convert_frames_to_video(input, output, fps)
    convert(input, output, fps)
END FUNCTION

# Function to create a directory
FUNCTION create_directory(path)
    create(path)
END FUNCTION

# Function to move a file
FUNCTION move_file(source, destination)
    move(source, destination)
END FUNCTION

# Function to clean up files or directories
FUNCTION cleanup(path)
    delete(path)
END FUNCTION

# Class for anomaly detection
```

```
CLASS AnomalyDetection
    FUNCTION init(dataset_path)
        self.dataset_path = dataset_path
        self.classifier = load_classifier()
        self.load_features("anomaly_features.npy")
    END FUNCTION

    FUNCTION load_features(file)
        load(file)
    END FUNCTION

    FUNCTION process_videos()
        FOR each video IN self.dataset_path DO
            self.process_video(video)
        END FOR
    END FUNCTION

    FUNCTION process_video(video)
        FOR each frame IN video DO
            self.process_frame(frame)
        END FOR
    END FUNCTION

    FUNCTION process_frame(frame)
        features = extract_features(frame)
        anomaly = classify(features, self.classifier)
        save_result(anomaly)
    END FUNCTION
```

```
    FUNCTION generate_report()
        results = compute_results()
        print(results)
        save("report.txt", results)
    END FUNCTION
END CLASS
```

Algorithm

1. **Parse Command-Line Arguments**

- **Input**: None

- **Output**: Parsed arguments

- **Steps**:

  - Call `parse_arguments()` function to parse command-line arguments `--tracking` and `--test`.

2. **Reorganize and Label Dataset Samples**

- **Input**: `arguments.test`

- **Output**: Labeled dataset

- **Steps**:

  - Run `python utils.py -test {arguments.test}` using the `run` function.

3. **Run the Test Script**

- **Input**: None

- **Output**: Output frames

- **Steps**:

- Run `python test_script.py` using the `run` function.

4. **Convert Output Frames to a Video**

- **Input**: Frames from `output/frames`

- **Output**: Video `output/video.mp4`

- **Steps**:

  - Call `convert_frames_to_video` function with input `output/frames`, output `output/video.mp4`, and `fps=25`.

5. **Conditional Tracking and Anomaly Detection**

- **Input**: `arguments.tracking`

- **Output**: Anomaly detection report, cleaned intermediate files

- **Steps**:

  - If `arguments.tracking` is enabled:

    - Create `results` directory using `create_directory` function.

    - Run `python track.py --source output/video.mp4 --yolo-weights weights/yolov5s.pt --classes 0` using `run` function.

    - Move tracked video from `output/video.mp4` to `results/` using `move_file` function.

    - Initialize `AnomalyDetection` class with `results` path.

    - Call `process_videos()` method of `AnomalyDetection` class to process videos for anomalies.

    - Call `generate_report()` method of `AnomalyDetection` class to generate anomaly detection report.

    - Clean up intermediate files:

      - Call `cleanup` function to delete `output/frames`.

      - Call `cleanup` function to delete `output/video.mp4`.

      - Call `cleanup` function to delete `results`.

6. **End**

Functions

parse_arguments()

- **Input**: None
- **Output**: Parsed command-line arguments
- **Steps**:
  - Parse `--tracking` and `--test` arguments.
  - Return parsed arguments.

run(command)

- **Input**: Command string
- **Output**: None
- **Steps**:
  - Execute the given system command.

convert_frames_to_video(input, output, fps)

- **Input**: Input directory, output file path, frames per second
- **Output**: Video file
- **Steps**:
  - Convert frames in the input directory to a video file.

create_directory(path)

- **Input**: Directory path
- **Output**: None
- **Steps**:
  - Create the specified directory.

move_file(source, destination)

- **Input**: Source file path, destination directory
- **Output**: None

- **Steps**:
  - Move the file from source to destination.

cleanup(path)
- **Input**: Path to file or directory
- **Output**: None
- **Steps**:
  - Delete the specified file or directory.

Class: AnomalyDetection

init(dataset_path)
- **Input**: Path to dataset
- **Output**: Initialized AnomalyDetection object
- **Steps**:
  - Set `self.dataset_path` to the provided dataset path.
  - Load classifier using `load_classifier()`.
  - Load features from `anomaly_features.npy` using `load_features` method.

load_features(file)
- **Input**: File path to features
- **Output**: Loaded features
- **Steps**:
  - Load features from the specified file.

process_videos()
- **Input**: None
- **Output**: Processed videos for anomalies
- **Steps**:

  - For each video in `self.dataset_path`, call `process_video` method.

process_video(video)

- **Input**: Video file

- **Output**: Processed video frames

- **Steps**:

  - For each frame in the video, call `process_frame` method.

process_frame(frame)

- **Input**: Frame data

- **Output**: Anomaly result

- **Steps**:

  - Extract features from the frame.

  - Classify features using the classifier.

  - Save the anomaly result.

generate_report()

- **Input**: None

- **Output**: Anomaly detection report

- **Steps**:

  - Compute results.

  - Print results.

  - Save results to `report.txt`.

This algorithm captures the entire process, including dataset preparation, model execution, frame-to-video conversion, pedestrian tracking, and anomaly detection.