443 Final Report

Lyla, Andrew, Trina

6/8/23

1 Introduction

Multi-Object Tracking (MOT) involves identifying specific objects across multiple frames of a video. In our case, we are implementing single-camera tracking, which only requires classification to be consistent within one video sequence, rather than between multiple. In any case, the goal is for objects to be consistently classified as they move around the frame. Additionally, they should be correctly re-identified if they exit the frame and re-enter.

2 Applications

2.1 Security

MOT is a useful tool for identifying people, objects, or vehicles that might be of interest to security personnel when reviewing footage. If, for example, a business was robbed, running MOT on security camera footage would allow for faster identification of how many individuals were on the scene and what their pathways were.

2.2 Social Media

Social media sites and other video communication platforms need to be able to track objects across multiple frames in order to apply any kind of filter or animation that attaches to certain features, such as a face.

2.3 Research

MOT is very effective for researchers who need to quickly classify a large number of moving objects. Examples include a scientist trying to tally sightings of a certain animal or a doctor tracking certain cells in a blood sample.

3 Provided Materials

3.1 Dataset

The teaching team provided the dataset we used. It includes multiple computer-animated videos of people walking around a room, some of whom exit and re-enter multiple times.

3.2 Baseline Code

We chose to utilize the starter code written by the teaching team, which also included detection and embedding files from a pre-trained YOLO model. With the provided framework, our contribution was in improving the tracker.

4 Challenges and Obstacles

One major challenge when implementing accurate MOT is striking a proper balance between tracking objects consistently frame-to-frame and eliminating false positive results. With a higher confidence level threshold, one can reduce the amount of "objects" that the MOT falsely classifies (e.g. "recognizing" a chair as a person), but it also makes the tracking of genuine objects choppier. This is especially true if objects pass behind each other or are otherwise obscured. Another tricky aspect of MOT is properly grouping like objects. For example, in our case, we were attempting to identify individual people in a video. People who looked very different were easy to classify as individuals, but people who looked similar often got mistaken for each other.

5 Code Implementation

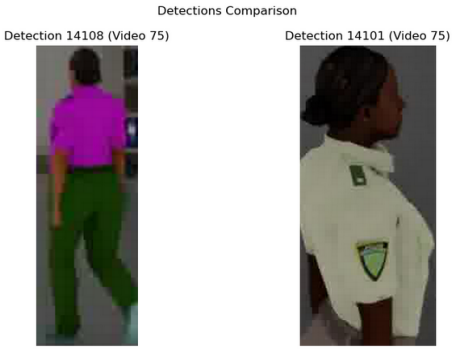
5.1 Preprocessing

In the preprocessing section, we load the detection and embedding data and organize it for further processing. The detection data is loaded from the `detection.txt` and the embedding is loaded `embedding.npy` file. We then sort the detection and embedding data based on the frame ID to ensure they are in the correct order for processing.

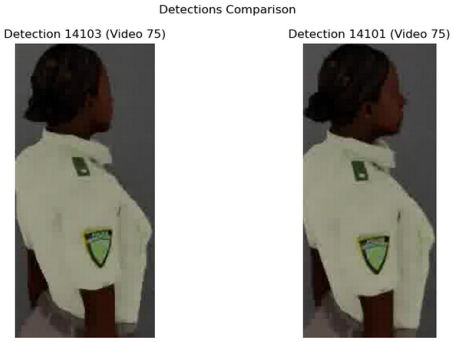
5.2 Helper Functions

We used the helper functions section to plot various functions that assist in the tracking process. One crucial function is the `Object_Detection` class, representing a detected object in a frame. This class contains attributes such as the object's ID, bounding box coordinates, and frame number.

Additionally, we have implemented a distance calculation function that computes the Euclidean distance between two object embeddings. This function is



(a) Cosine Difference between the Two detections: 0.0010015368461608887



(b) Cosine Difference between the two detections: 0.765363335609436

Figure 1: Comparison of Detections

used to determine the similarity between objects and assists in matching objects across frames as shown in Fig.1.

We also have a visualization function that draws bounding boxes and labels on the frames to visualize the tracking results.

5.3 Confidence Thresholding

In object detection tasks, the confidence score associated with each detection confidence in accurately identifying an object. Not all detections may have high confidence scores, and some may correspond to false positives or ambiguous detections.

Applying a confidence threshold filters out detections with low confidence scores. This helps to remove unreliable or less confident detections from further processing, ensuring that we focus on more confident and potentially more accurate detections.

Setting an appropriate confidence threshold involves finding a balance between including enough relevant detections while reducing false positives or noisy detections.

The confidence thresholding code snippet below filters out low-confidence detections below 0.5 threshold.

```
# Filter out low-confidence detections below a specified threshold
confidence_threshold = 0.5
confidence_mask = test_detection[:, 7] >= confidence_threshold
test_detection = test_detection[confidence_mask]
test_embedding = test_embedding[confidence_mask]
```

5.4 Intersection over Union (IoU)

The IoU code snippet calculates the overlap ratio of two bounding boxes using Intersection over Union (IoU). It measures the extent of overlap between two bounding boxes, which is a commonly used metric in object detection tasks.

The Intersection over Union (IoU) calculates the ratio of the intersection area to the union area of two bounding boxes.

Calculating the IoU between the bounding boxes of tracklets helps determine their similarity or overlap. By comparing the IoU values, we can assess the spatial correspondence between tracklets across frames. Higher IoU values indicate a higher likelihood of representing the same object or person, while lower IoU values indicate potential fragmentation or different objects.

The IoU calculation is used in the process of matching tracklets across frames and deciding whether to merge them or consider them as separate entities. Tmplementatio of the IoU code is shown below.

```
# Calculate the overlap ratio of two bounding boxes using IoU
def calculate_iou(bbox1, bbox2):
    x1_1, y1_1, x2_1, y2_1 = bbox1
```

```
    x1_2, y1_2, x2_2, y2_2 = bbox2
    % Calculate intersection area
    ...
    % Calculate union area
    ...
    % Calculate IoU
    iou = intersection_area / union_area
    return iou

% Example usage:
iou = calculate_iou(bbox1, bbox2)
```

5.5 K-means Clustering

K-means clustering is an algorithm used for grouping similar data points into clusters based on their feature similarity. K-means clustering is employed as a post-processing step to merge fragmented tracklets based on their appearance similarity.

By clustering the appearance features of tracklets using K-means, we aim to group together tracklets that exhibit similar visual characteristics. This helps in consolidating fragmented tracklets caused by occlusion or appearance variations. The algorithm assigns a cluster label (representing a specific person or object) to each tracklet, allowing us to identify and track individuals consistently across frames.

The number of clusters (in this case, the number of people) needs to be specified in advance. The choice of the number of clusters depends on the number of distinct individuals or objects present in the dataset.

The K-means clustering code snippet demonstrates the use of the K-means algorithm for post-processing the tracklets.

```
from sklearn.cluster import KMeans

% Class for post-processing using K-means clustering
class postprocess:
    def __init__(self, number_of_people):
        self.n = number_of_people
        self.cluster_method = KMeans(n_clusters=self.n, random_state=0)

    def run(self, features):
        self.cluster_method.fit(features)
        labels = self.cluster_method.labels_
        return labels
```

5.6 Re-identification (Re-ID)

The Re-identification code snippet demonstrates the process of assigning IDs to tracklets to maintain their identity across frames. This is achieved by comparing the appearance features of tracklets and assigning the same ID to those with similar appearances.

```
class Tracklet:
    def __init__(self, tracking_ID, box, feature, time, confidence):
        self.ID = tracking_ID
        self.boxes = [box]
        self.features = [feature]
        self.times = [time]
        self.confidences = [confidence]
        ...
```

The 'Tracklet' class represents a single tracklet and keeps track of its ID, bounding boxes, appearance features, timestamps, and confidences. By comparing the appearance features of tracklets, they can be matched and assigned the same ID if they are similar.

These code snippets demonstrate how to implement the confidence thresholding, IoU calculation, K-means clustering, and Re-identification in the given context. The explanations provided give an overview of the

```
# Code snippet example
import numpy as np

# Load detection data
detection_data = np.loadtxt('detection.txt')

# Load embedding data
embedding_data = np.load('embedding.npy')

# Sort data based on frame ID
detection_data = detection_data[np.argsort(detection_data[:, 0])]
embedding_data = embedding_data[np.argsort(embedding_data[:, 0])]
```

6 Results and Observations

The variable we modified the most was our confidence level threshold. We started it at 0.3 and found that removing anything with a confidence less than 0.3 was helpful in eliminating the small false-positive identifications that often appeared as two people walked past each other. However, it did not eliminate, for example, the identification of the chair in the left half of camera 75 as a person. We then tried a threshold of 0.6, which worked, but made the tracking of each person across the frame slightly choppy.

In testing several different thresholds, including 0.1, 0.3, 0.6, and 0.8, we found that certain metrics improved with a higher threshold (such as recall) but some improved with a lower threshold (such as IDF1). Some, such as MOTA, were best somewhere in the middle. Overall, taking into account the amount of false-positives, a threshold between 0.3 and 0.6 seemed like a good medium. 0.8 was certainly too high, and 0.1 certainly too low.

While we were not able to compare the test results (camera 75) with the ground truth to see those evaluation metrics, we were able to visualize the tracking in animation form to see how our model did. The results we settled on were for camera 75 with a 0.5 confidence threshold. We still had some false positives, and a couple people were conflated with each other, but the tracking was pretty smooth.