Social Distancing

Introduction

In this tutorial, we will walk through a simple use case that checks if someone is violating social distancing rules.

Please be aware that the goal of this tutorial is to help you get familiar with the usage of BrainFrame's inference capabilities. A real social distancing use case is much more complicated than this script.

In this script, we only have two rules:

- Two `person` detection bounding boxes cannot overlap
- The distance between the center of two people detections' bounding boxes must be greater than 500 pixels (by default; this will be configurable).

You can find the complete script on our GitHub repository.

Setup The Environment

The environment setup is similar to the environment we have in the WeChat Notification tutorial. You can refer to it to set up the environment.

In this tutorial, the API functions that we are going to use are:

- `api.set_stream_configuration(...)`
- `api.set_zone(...)`
- `api.get_zone_status_stream()`
- `api.set_plugin_option_vals(...)`

Help Function

First, import the dependencies:

```
import math
from argparse import ArgumentParser
from pathlib import Path

from brainframe.api import BrainFrameAPI, bf_codecs
```

To make the script more readable, we'll define two helper functions in advance.

The first will check if two bounding boxes are overlapped or not:

```
# Help function to check if two detections are overlapped
def is_overlapped(det1: bf_codecs.Detection,
                  det2: bf_codecs.Detection) -> bool:
    """
    :param det1: First Detection
    :param det2: Second Detection
    :return: If the two Detections' bboxes are overlapped
    """

    # Sort the x, y in ascending order
    coords1_sorted_x = sorted([c[0] for c in det1.coords])
    coords2_sorted_x = sorted([c[0] for c in det2.coords])
    coords1_sorted_y = sorted([c[1] for c in det1.coords])
    coords2_sorted_y = sorted([c[1] for c in det2.coords])

    # Return False if the rects do not overlap horizontally
    if coords1_sorted_x[0] > coords2_sorted_x[-1] \
            or coords2_sorted_x[0] > coords1_sorted_x[-1]:
        return False

    # Return False if the rects do not overlap vertically
    if coords1_sorted_y[0] > coords2_sorted_y[-1] \
            or coords2_sorted_y[0] > coords1_sorted_y[-1]:
        return False

    # Otherwise, the two rects must overlap
    return True
```

The second helper function one will calculate the distance between the center of two bounding boxes:

```
# Helper function to calculate the distance between the center points of two
# detections
def get_distance(det1: bf_codecs.Detection,
                 det2: bf_codecs.Detection) -> float:
    """
    :param det1: First Detection
    :param det2: First Detection
    :return: Distance between the center of the two Detections
    """
    return math.hypot(det1.center[0] - det2.center[0],
                      det1.center[1] - det2.center[1])
```

Create a New Stream from Local File

First, initialize the API instance and connect to the server.

```
# Initialize the API
api = BrainFrameAPI("http://localhost")
```

Then, we want to start a video stream, you can find the sample video on our tutorial repository.

```
# Upload the local file to the database and get its storage ID
storage_id = api.new_storage(
    data=Path("../videos/social_distancing.mp4").read_bytes(),
    mime_type="application/octet-stream"
)

# Create a Stream Configuration referencing the new storage ID
new_stream_config = bf_codecs.StreamConfiguration(
    # The display name on the client side
    name="Demo",
    # This stream will be from a file
    connection_type=bf_codecs.ConnType.FILE,
    # The storage ID of the file
    connection_options={
        "storage_id": storage_id,
    },
    runtime_options={},
    premises_id=None,
)

# Tell the server to connect to that stream configuration
new_stream_config = api.set_stream_configuration(new_stream_config)
```

Next, we will configure some capsule options instead of using the default ones we defined earlier to filter out some bad detections.

```
# Filter out duplicate detections
api.set_plugin_option_vals(
    plugin_name="detector_people_and_vehicles_fast",
    stream_id=new_stream_config.id,
    option_vals={
        # If one bounding box is overlapped >80% with another bounding box, we
        # assume that they are really the same detection and ignore them.
        "max_detection_overlap": 0.8,
        "threshold": 0.9
    }
)
```

Finally, don't forget to tell BrainFrame to start analyzing/preforming inference on the stream.

```
# Start analysis on the stream
api.start_analyzing(new_stream_config.id)
```

Check Social Distancing Rules

Next, similar to the WeChat Notification tutorial, we will get the zone status iterator. Will will iterate through all of its zone statuses, checking against the social distancing rules we defined above.

In the WeChat Notification tutorial, the operations on zone statuses were somewhat complicated, being a nested data structure. In this tutorial, we will reorganize in order to make our calculations here easier.

```
# Verify that there is at least one connected stream
assert len(api.get_stream_configurations()), \
    "There should be at least one stream already configured!"

# Get the inference stream.
for zone_status_packet in api.get_zone_status_stream():
    # Organize detections results as a dictionary of
    # {stream_id: [Detections]}.
    detections_per_stream = {
        stream_id: zone_status.within
        for stream_id, zone_statuses in zone_status_packet.items()
        for zone_name, zone_status in zone_statuses.items()
        if zone_name == "Screen"
    }

    # Iterate over each stream_id/detections combination
    for stream_id, detections in detections_per_stream.items():
        # Filter out Detections that are not people
        detections = [detection for detection in detections
                      if detection.class_name == "person"]

        # Skip stream frame if there are no person detections
        if len(detections) == 0:
            continue
        # Compare the distance between each detections.
        for i, current_detection in enumerate(detections):
            violating = False
            for j in range(i + 1, len(detections)):
                target_detection = detections[j]
                current_detection: bf_codecs.Detection
                target_detection: bf_codecs.Detection
                # If the bbox representing two people are overlapped, the
                # distance is 0, otherwise it's the distance between the
                # center of these two bbox.
                if is_overlapped(current_detection, target_detection):
                    distance = 0
                else:
                    distance = get_distance(current_detection, target_detection)

                if distance < min_distance:
                    print(f"People are violating the social distancing rules, "
                          f"current distance: {distance}, location: "
                          f"{current_detection.coords}, "
                          f"{target_detection.coords}")
                    violating = True
                    break
            if violating:
                break
```

Now, whenever people violate our social distancing rules, our script will print the message out, including where are they located in the frame.

```
People are violating the social distancing rules, current distance: 499.8899878973373,
location: [[30, 340], [411, 340], [411, 926], [30, 926]],
[[571, 238], [828, 238], [828, 742], [571, 742]]
```