Pepito, Alyssa Mae & Soleño, Keziah Antonette | Group 18 BM1 | CS173

M2 – SA (YOLOv7 + DeepSORT)

For this activity, the group was tasked to integrate a DeepSORT (Simple Online and Realtime Tracking with a Deep Association Metric) optimization on the YOLOv7 model created in M2-FA2. DeepSORT is a real-time object-tracking algorithm that combines the features of a deep neural network with a simple online and offline track association algorithm. It is built on top of the SORT (Simple Online and Realtime Tracking) algorithm, which uses the Hungarian algorithm to associate detections and tracks in a greedy manner. The deep neural network extracts features from the detections, which are then used to compute the similarity between detections and tracks.

I. Cloning of Repositories and Installing Requirements

In the previous activity, the group was asked to use the same dataset in M2-FA1 to create a YOLOv7 model that would detect motorcycle riders and the type of helmet being used. The dataset was divided into three-time scenarios, 5:00 - 6:00 pm, 6:00 - 7:00 pm, and 7:00 - 8:00 pm.

Following that activity, the group will integrate the DeepSORT optimization on the YOLOv7 model from that activity to compare those results with the optimized (DeepSORT) results later. To begin, the provided code from the Github link here was cloned to access the YOLOv7 and DeepSORT tracking code. To clone the repository and to download the yolov7-deepsort-tracking folder in the local project folder, the following code was run:

```
# Clone yolov7-deepsort-tracking github repository
!git clone https://github.com/deshwalmahesh/yolov7-deepsort-tracking
%cd yolov7-deepsort-tracking
```

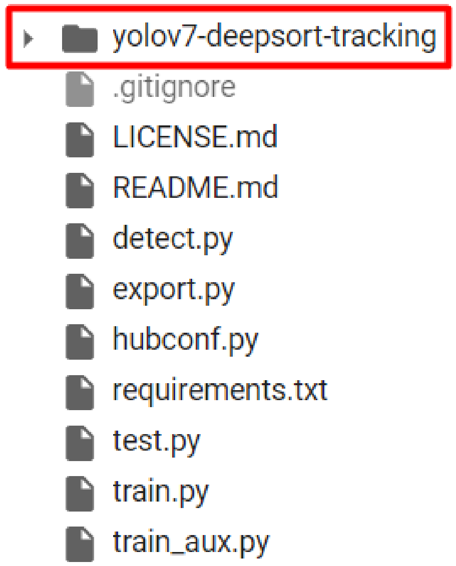
Additionally, the following code was run to download the weight file from the previous assignment into our current assignment so that this file will be used later to detect the classes from the dataset.

```
# Download the weight file from M2 - FA2
!wget https://github.com/MaePp2/yolov7-M2-SA/raw/main/yolov7z.pt
```

Lastly, to access the test videos that were created in the previous assignments, the code snippet below was run.

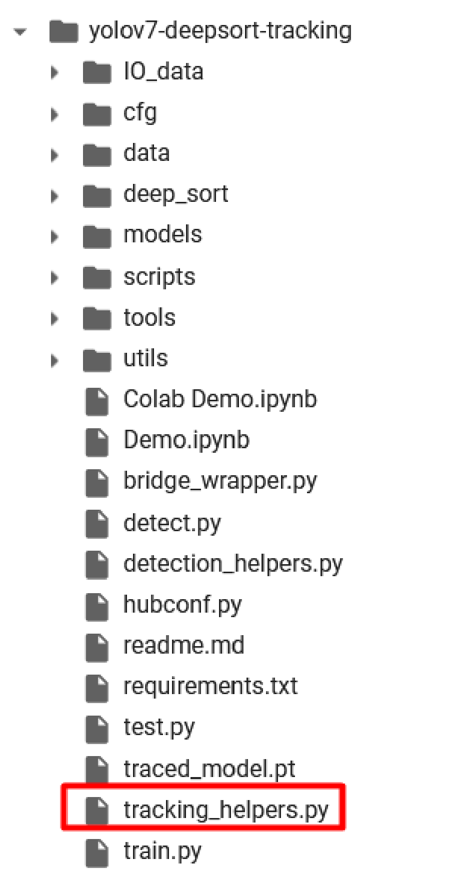
```
from google.colab import drive
drive.mount ("/content/drive")
```

After these codes finish running, the project folder should contain the new folder as highlighted in the figure below.



II. Importing Local Libraries from Cloned GitHub Repository

After cloning the repository and installing the necessary files, we must first modify the *tracking_helpers.py* file under the yolov7-deepsort-tracking folder we have just installed. Navigate to that folder and click on the .py file.



Scroll down to the .py file until you will reach the *read_class_names* function. On this function, you must change the *classes* list to the classes we have to identify in our model. After editing, make sure to save the changes to your colab notebook.

Note: Refer to the previously made Yolov7 colab notebook for the order for these classes. It should be revealed on the table (top to bottom) under the detect.py or test.py results.

```
def read_class_names():
    '''
    Raad COCO classes names
    '''
    classes = ['Full-Faced', 'Half-Faced', 'Invalid', 'Not Wearing Helmet', 'Rider']

    return dict(zip(range(len(classes)), classes))
```

Following this, local libraries detection_helpers, tracking_helpers, and bridge_wrapper from the project folder were imported by running the following code.

```
from detection_helpers import *
from tracking_helpers import *
from bridge_wrapper import *
from PIL import Image
```

III. Detection

The videos that were previously used for testing will be detected using the detector class as the weight file mentioned earlier will be referenced and set as the model path. To set the YOLOv7 model weight file as the model path, the code snippet below was run.

```
detector = Detector()
model_path = '/content/yolov7z.pt'
detector.load_model(model_path,)
```

Now that the detector class has been defined and the weight file set as the model path, the next step is to test the detector if it has been successfully loaded. To do so, we used a test image that was acquired by taking a screenshot from one of the test videos; we set that image as one of the parameters of the detect class, as seen in the code snippet below.

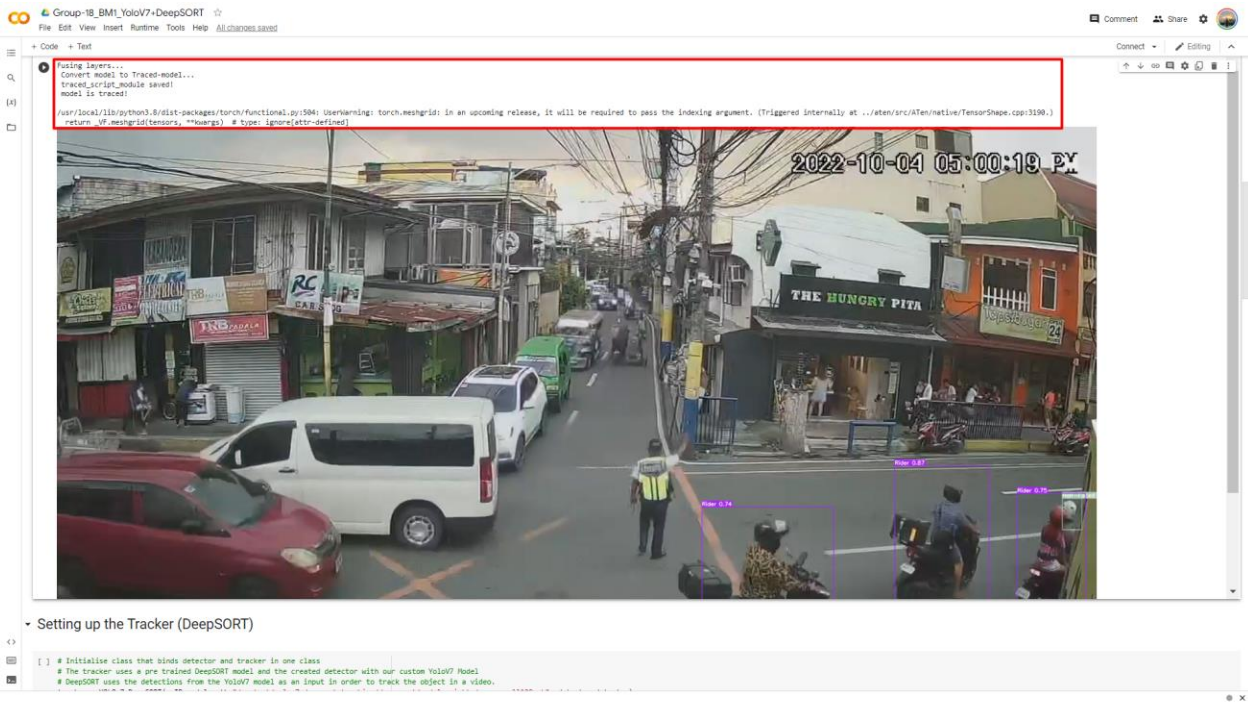
```
result = detector.detect('/content/drive/MyDrive/M2-
SA/testvideos/testpic.png', plot_bb = True)
```

Additionally, the following code snippets will be run to convert the image into a proper image, which is also known as the BGR image, and to display the results in the following cell.

```
if len(result.shape) == 3:
    result = Image.fromarray(cv2.cvtColor(result,cv2.COLOR_BGR2RGB))
```

```
result
```

The figure below is the output of the codes written above.



IV. Setting up the YOLOv7-DeepSORT Tracker

Once we have tested the detector, it was time to initialize a class that binds the detector and tracker into one. The tracker provided by the repository uses a pre-trained DeepSORT model and the created detector using our Yolov7 model. To track an object in a video, DeepSORT uses the detections from the YoloV7 model as an input.

```
tracker = YOLOv7_DeepSORT(reID_model_path="/content/yolov7-deepsort-tracking/deep_sort/model_weights/mars-small128.pb", detector=detector)
```

V. Tracking MP4 files via DeepSORT

The YOLOv7-DeepSORT tracker that had been set up in the previous section was then used to track the test videos found in [this GoogleDrive].

Note: The MP4 files in the GoogleDrive were the only videos tracked during this project.

Like in the previous assignments, each video was classified according to the classes *Full-Faced*, *Half-Faced*, *Invalid*, *Not Wearing Helmet*, and *Rider*. To track these videos according to the specified classes, a similar code snippet, as seen below, was run for each time scenario.

5:00PM – 6:00PM

```
tracker.track_video("/content/drive/MyDrive/M2-SA/testvideos/5-6PM.mp4", output="/content/drive/MyDrive/M2-SA/testvideos/5-6PM_Scenario.avi", show_live = False, skip_frames = 0, count_objects = True, verbose=2)
```

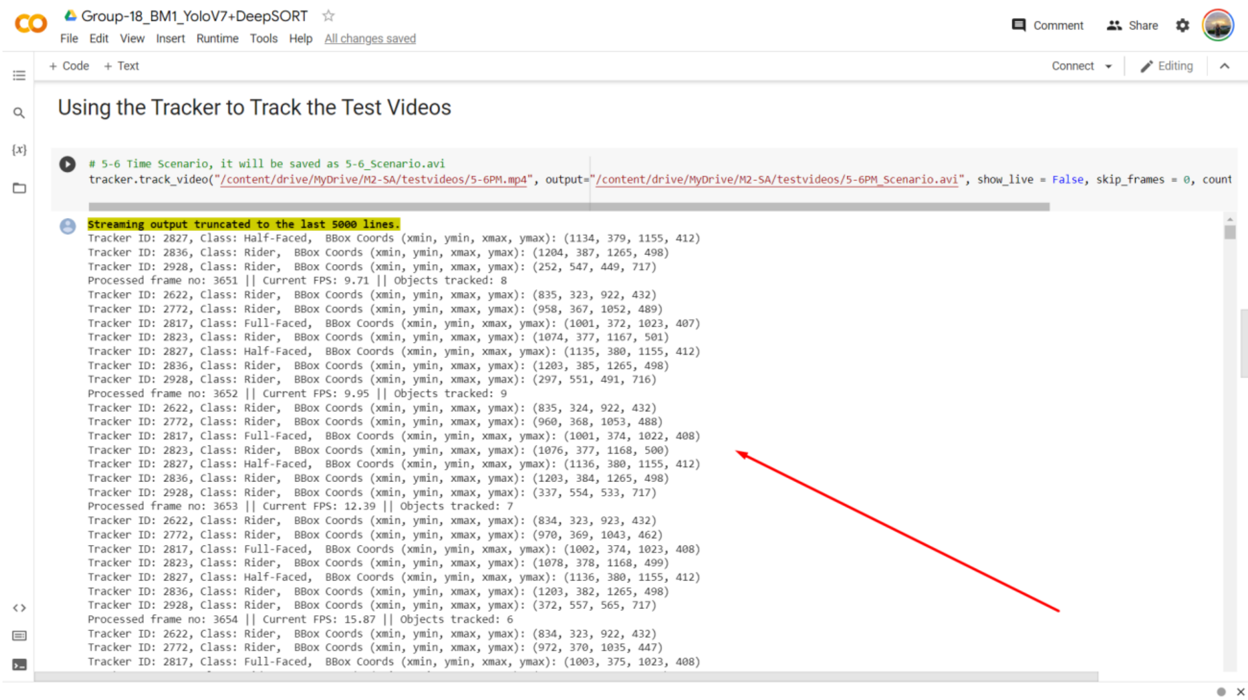
6:00PM – 7:00PM

```
tracker.track_video("/content/drive/MyDrive/M2-SA/testvideos/6-7PM.mp4", output="/content/drive/MyDrive/M2-SA/testvideos/6-7PM_Scenario.avi", show_live = False, skip_frames = 0, count_objects = True, verbose=2)
```

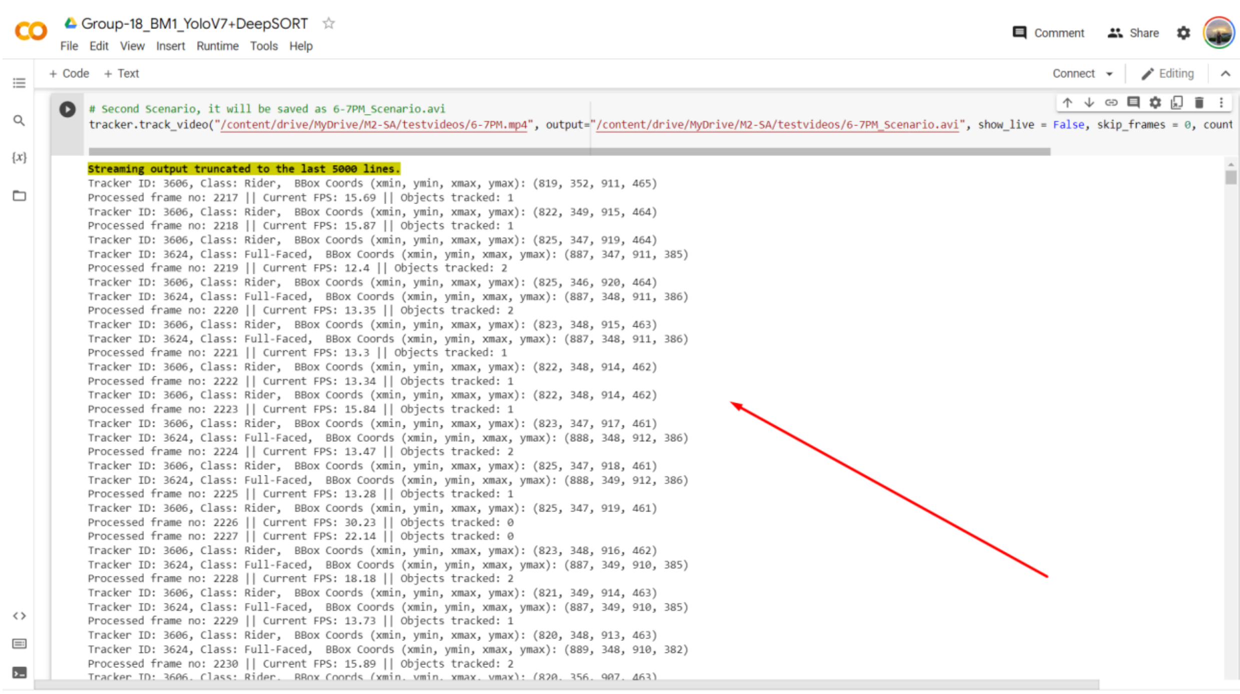
7:00PM – 8:00PM

```
tracker.track_video("/content/drive/MyDrive/M2-SA/testvideos/7-8PM.mp4", output="/content/drive/MyDrive/M2-SA/testvideos/7-8PM_Scenario.avi", show_live = False, skip_frames = 0, count_objects = True, verbose=2)
```

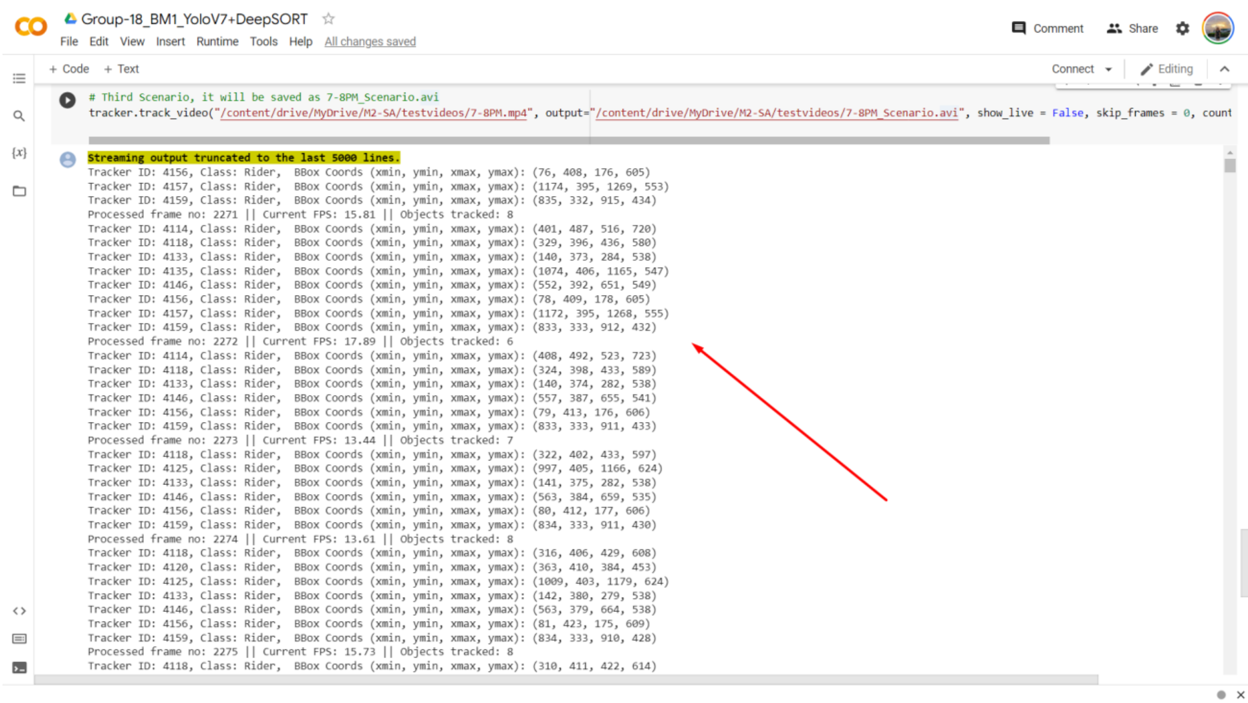
Output:



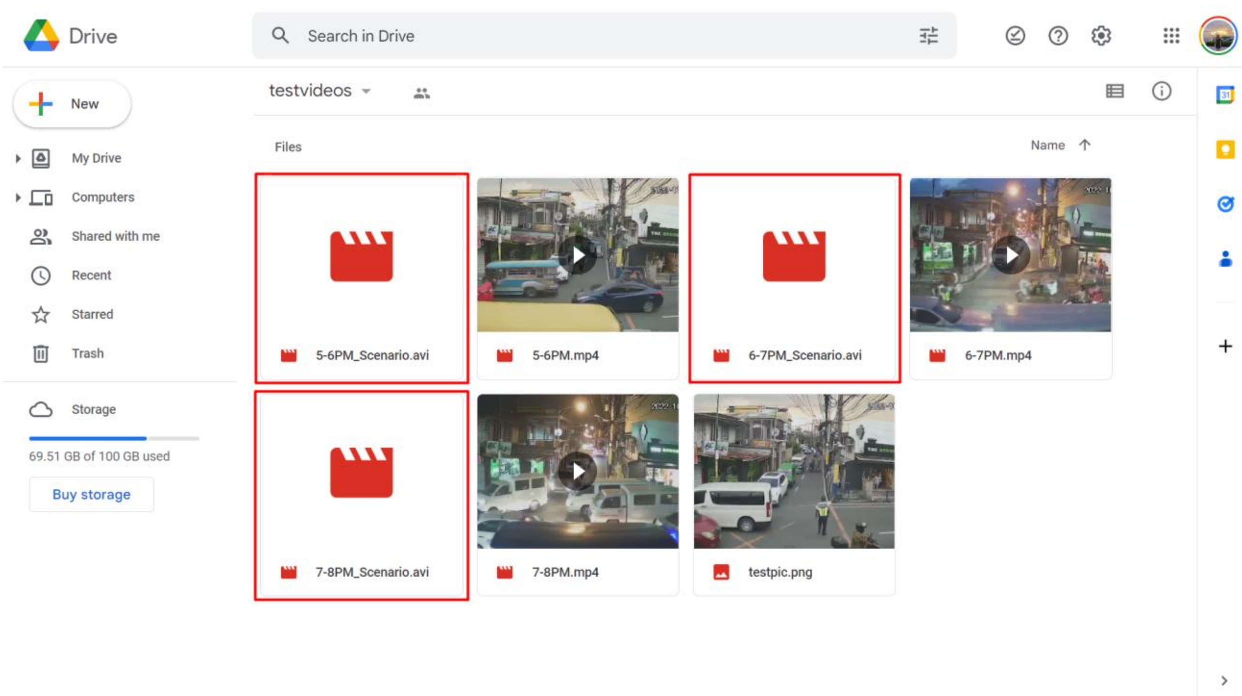
5:00PM – 6:00PM



6:00PM – 7:00PM



7:00PM – 8:00PM



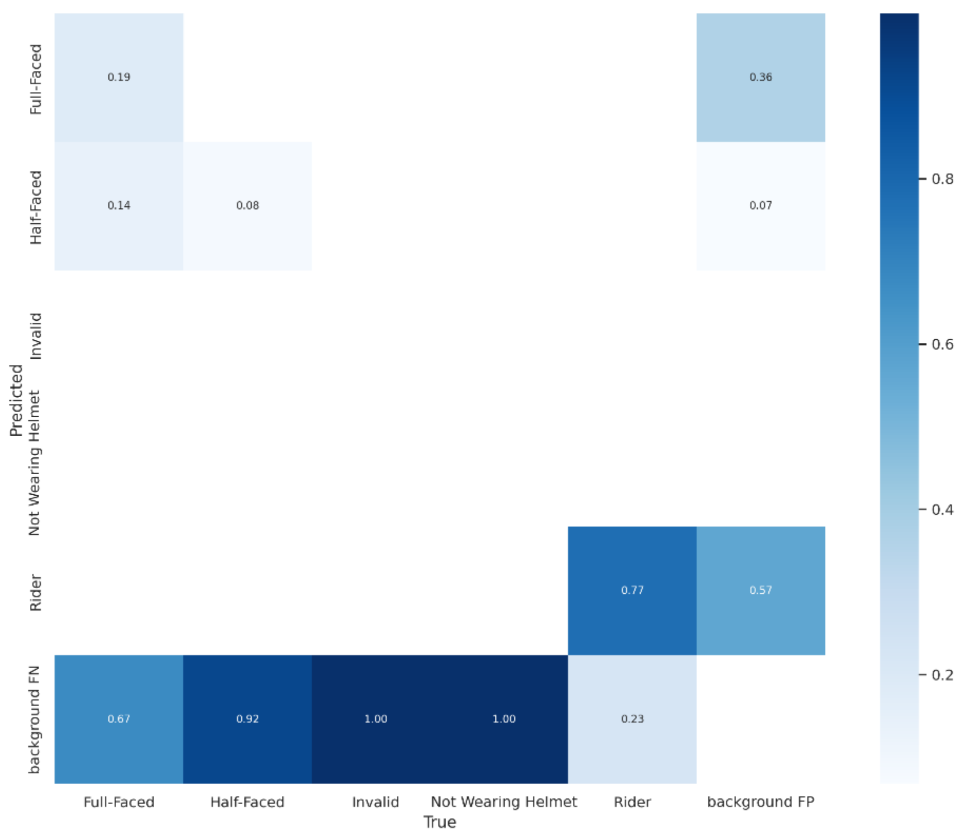
GoogleDrive Output

IV. Ground Truth of Dataset and Confusion Matrix

Using the same method as previously stated in our assessments M2-FA1 and M2-FA2, we manually counted all the classes within the test dataset. This time, however, the dataset we used was the combination of all the three-time scenarios. Using the excel method, we created the following table:

Total	
Rider	234
Full-Faced	112
Half-Faced	77
Invalid	16
Not Wearing Helmet	7

After testing the YOLOv7 model, we were able to produce this confusion matrix:



V. DeepSORT Optimization on Modeling Results (Analysis)

As we recall, on the first assessment that utilized YOLOv5, the model did poorly detecting the proper classes in the videos. The graphs revealed that the model was underfitted and overfitted because of its subpar performance on the training and evaluation data, respectively.

On the other hand, the YOLOv7 model of the previous assessment performed average in detecting if it was a proper helmet or not. But despite the average results, YOLOv7 is a much better model in terms of precision/accuracy.

Now with DeepSORT as optimization, the YOLOv7 model performed higher in tracking accuracy than the previous models. Furthermore, the FPS and run times on the three videos were quicker. With this knowledge, it does make sense because DeepSORT employs a better association measure that incorporates motion and appearance descriptors. However, DeepSORT only optimizes the model in terms of tracking objects and not increasing accuracy/precision.

To conclude, DeepSORT is an excellent algorithm for tracking objects as it utilizes deep learning into the SORT algorithm. It is a solid choice for multiple object detection and tracking.

References

Boesch, G. (2023, January 1). *YOLOv7: The Most Powerful Object Detection Algorithm (2023 Guide)*. viso.ai. https://viso.ai/deep-learning/yolov7-guide/

Dwyer, B. (2022, December 29). *How to Train YOLOv7 on a Custom Dataset*. Roboflow Blog. https://blog.roboflow.com/yolov7-custom-dataset-training-tutorial/

Kanjee, R. (2022, May 9). *DeepSORT — Deep Learning applied to Object Tracking*. Medium. https://medium.com/augmented-startups/deepsort-deep-learning-applied-to-object-tracking-924f59f99104

Sanyam. (2022, June 21). *Understanding Multiple Object Tracking using DeepSORT*. LearnOpenCV. https://learnopencv.com/understanding-multiple-object-tracking-using-deepsort/

Skelton, J. (2022, August 17). *How to train and use a custom YOLOv7 model*. Paperspace Blog. https://blog.paperspace.com/yolov7/

theAIGuysCode. (2021, August 21). *Simple Online and Realtime Tracking with a Deep Association Metric (Deep SORT)*. GitHub. https://github.com/theAIGuysCode/yolov4-deepsort