Table of Contents

I. Introduction

YOLOV7 and DeepSort are both cutting-edge technologies that have revolutionized the field of object detection and object tracking. YOLOV7, short for You Only Look Once Version 7, is an object detection algorithm that uses deep learning that can identify and locate objects in real-time through videos. It is widely regarded as one of the fastest and most accurate object detection algorithms available today and it is one of the popular choices for a wide range of applications, including surveillance, autonomous driving, and sports analysis. In terms of helmet detection on motorcycle riders, YOLOV7 can accurately identify helmets quickly in real-time video streams, becoming a necessary tool to keep motorcyclists safe on the road.

Whilst DeepSort is an object tracking algorithm that can track objects over long periods of time, even in complex and crowded environments. It uses a combination of deep learning and classical tracking techniques to track objects based on their appearance and motion patterns, making it an ideal tool for tracking motorcycle riders and their helmets in real-world scenarios. When combined with YOLOV7, DeepSort can provide a comprehensive solution for helmet detection and tracking, allowing authorities to quickly identify and monitor riders who are not wearing helmets and take appropriate action to ensure their safety.

With that said, YOLOV7 and DeepSort are both open-source technologies that are constantly evolving and improving as our era progresses with the world reliant on technology as gateways to communication and helping people. Because they implemented this, both have contributed to the research community and public. Its capabilities are endless, people continue to add more features. The advantages of YOLOv7 is the ability to detect multiple objects in a single image or video frame with high accuracy and speed.

Which means, it serves beneficial for observing surveillance and sports analysis. Other than that, it can detect pedestrians, vehicles,and animals. Making YOLOv7 a versatile tool for a wide range of applications including the following: Traffic Monitoring, WIldlife conservation, and industrial automation. DeepSort, on the other hand, excels at tracking objects over a certain long period of time, its capabilities to detect objects even if it has blurs, clutter, and occlusions or a point where the rider is unpredictable/unseen. It is the ideal tool for tracking riders that not even a person may be able to do.

Combining these two, we can further enhance integrating other technologies that will be practicable and beneficial to the progression of people utilizing technology in this growing era such as: Computer Vision, Machine Learning, and Natural Language processing. They have the best solution for solving complicated problems that are significant to aiding us. The technology we have. It may not be perfect and there will certainly be limitations to it. Challenges associated with YOLOV7 and Deepsort. For example, they may struggle with detecting objects in low-light conditions, or they may be vulnerable to adversarial attacks that manipulate and confuse the

algorithm. It is important to be aware of the limitations. And as Data Scientist Students, it is our role and responsibility to continue evaluating and improving the technology to ensure effectiveness and safety. For the future of revolutionizing object detection to greater heights!

II. Related Literatures

City-Scale Multi-Camera Vehicle Tracking of Vehicles based on YOLOv7

[1] aims to create a model that can accurately track the trajectory of a car using data from multiple cameras. The issue at hand is the problem of tracking one car with multiple cameras as if they were multiple cars. To do this, the researchers created thresholds between cameras so that a single camera will stop tracking cars that are within the vicinity of another camera. In a way, despite being a multiple camera setup, the cameras were tracking cars individually in a multiple single camera setup. The researchers saw performance metrics above 0.9 across the board.

Tree species identification method based on improved YOLOv7

In [2], the researchers improved upon YOLOv7 by adding a new small target detection layer in the backbone structure as well as an attention mechanism. The researchers then used this to detect tree species. The researchers used CIoU or Complete Intersection over Union as the loss function. Their dataset consisted of 4000 training images, and another 789 images for training and testing. Their final model achieved the highest correctness of 99.8.

Safety Helmet Detection Using Deep Learning: Implementation and Comparative Study Using YOLOv5, YOLOv6, and YOLOv7

In [3], the researchers wanted to use YOLOv7 to detect persons wearing safety helmets. Ther performed no alteration of the YOLOv7 algorithm and used datasets retrieved from Kaggle as well as the Mendeley dataset that has the requisite classes for helmet, person without helmet, and other classes in between. Their model was shown to perform ebay in the Mendeley dataset when using YOLOv7 and trained for 100 epochs and a batch size of 16, resulting in a mean average precision of 89.6%

Multi-target Pedestrian Tracking Based on YOLOv5 and DeepSORT

Researchers in [4] aim to monitor and track pedestrians in real-time videos. Thus, they propose a pedestrian tracking system that uses the YOLOv5 model and applied DeepSORT to improve accuracy and robustness of the model. The dataset used is called a PANDA is pixel-level video dataset for large-scale with annotations that is appropriate for long-term multi-object tracking of pedestrians in a video. The videos are split into image frames in jpg format that will be used for training and videos for testing. The YOLOv5 algorithm is used to detect multi-target locations and display tracking frames while DeepSORT is a tracker that combines prediction and detection in order to track multi-targets and generate unique IDs per object. After testing, the

proposed model was able to achieve a precision of 0.84, a recall score of 0.72, a F1-score of 0.78 and an mAP of 0.80. The experimental findings validate that the detection model created using the aforementioned techniques was able to accurately detect for various moving pedestrian targets in different traffic scenarios. [4]

People Tracking System Using DeepSORT

In study [5], its objective is to use the framework of DeepSORT to track people in a crowd surveillance system that uses learned information to track the person's real-time trajectory until it exits the frame of the camera. The dataset consists of videos of people passing by a street that were converted to an image format frame by frame. Then YOLO models such YOLOv3, YOLOv3 tiny and YOLOv3 custom are trained using this dataset that only differs in weight size. Then DeepSORT algorithm is used to process the detected person, assign a unique ID to it and then track the target object's movement even during occlusion. As for the experimental results, YOLOv3 has the highest accuracy of 91.03%. Thus, it was able to successfully detect and track the person's movement path with an average 2.59 frames per second (FPS). Meanwhile, YOLOv3 tiny performed way below the YOLOv3 model with an accuracy of 62.12% and YOLOv3 custom performed the worst with an accuracy of 45.80%. The different size of the dataset and object classes greatly affected the accuracy of the model. Therefore, an accurate dataset to be fed to the tracker will be useful for the performance of the tracking process. [5]

Pedestrian Target Tracking Based On DeepSORT With YOLOv5

In order to address tracking errors and low tracking accuracy, [6] proposed a system that is based on the YOLOv5 model with DeepSORT algorithm for tracking pedestrian targets. The dataset used in the study is a real street surveillance taken by the researchers containing scenes of moving pedestrian targets. A labeling software is utilized to label the video and to obtain the real label per pedestrian target for verification accuracy. Using YOLOv5 as the target detection algorithm, it extracts feature information from the dataset and detects the pedestrian target per frame. This serves as the basis for tracking the pedestrian target using the DeepSORT algorithm. The YOLOv5+DeepSORT model is compared to the YOLOv3+DeepSORT model to verify the performance of the proposed system. As for the performance comparison, the tracking accuracy MOTA of the YOLOv5+DeepSORT model is 60.1% while the YOLOv3+DeepSORT model has only an accuracy of 50.2%. Both models have successfully detected pedestrian targets. However, YOLOv3 failed to detect targets near the edges of the frame while YOLOv5+DeepSORT was still able to successfully detect objects positioned at the edges. Hence, YOLOv5+DeepSORT performs better in detecting pedestrian target objects even with few details located at the frame edges and was able to overcome false detection and tracking errors due to occlusion. Unfortunately, the level of accuracy is not that high as the model still resulted in detection errors. [6]

Improve High Speed Flame Detection Method Based on YOLOv7

The paper [7] proposes the use of a modified YOLOv7 model in detecting high speed flame. The paper uses YOLOv7-CN-B method, or the CBS module consists of Batch Normalization of Conv, BN (BN), and Silu activation functions, which improves the base model. The dataset contains 2059 images, which were collected from an open flame data set which was annotated by a box bound around the object or flame. The performances of the models were measured through mAP, accuracy, and frames per second or FPS. The images are then resized to 640x640, which improves the accuracy of the model. The proposed model is compared with YOLOv3, YOLOv4, YOLOv7, and Faster R-CNN Based on the results. The YOLOv7-CN-B scored 10% higher and is observed to be faster than the models compared. The proposed model has an improved accuracy of 5% and an improved mAP of 21%, as well as a FPS that is 54 frames higher than the compared models. Because of its score in accuracy, mAP value, and detection speed, it can be observed that YOLOv7-CN-B is more efficient in high-speed flame detection.

Research on volleyball players tracking based on improved DeepSORT

The study [8] proposes a Vol-DeepSORT model for tracking volleyball players using DeepSORT, which uses a Kalman Filter to predict the motion of the target player based on the Convolution Neural Network. The main objective of the study is to determine the outcome of the game and avoid rematches. The dataset used in the study consists of a video of a Guandong University Volleyball League match in 2021, containing 12 players. A total of 600 images are obtained from the video, and all the players are annotated. The proposed model matches the appearance features and targets of the players using a Kalman filter. The Mahalanobis Distance is used to accurately predict the location of the players during the match. The same video is used for testing the model, and the proposed model obtains the highest score of 90.7% in comparison to other models.

Vehicle Analysis System Based on DeepSORT and YOLOv5

The paper [9] focuses on the detection of a vehicle through the use of DeepSORT and YOLOv5. The paper utilizes Kalman filter for the prediction of the vehicle and the Hungarian algorithm for matching the vehicle for its identification, it also uses Inverse Perspective Mapping to map the position of the vehicle in the video. The dataset used in the study contains 11,000 vehicles from a video of high-speed road traffic flow, which is divided into “car” and “truck” class. The model is measured based on the mAP, precision, and recall. Based from the results, the car class was able to obtain a precision score of 94.8%, a recall score of 92.9%, and mAP of 96.0%, which is observed to be higher than the truck class. The truck class obtain a precision score of 93.9%, recall score of 90.7%, and mAP 91.7%. The proposed model was able to achieve high performance in high-speed roads based on the results.

A Comparative Study Of Deep-Learning Object Detectors For Semiconductor Defect Detection

[10] Compares the different Deep-Learning Object Detectors against each other in terms of their performance. Different models were used for the paper, for example, Faster R-CNN, DINO, RetinaNet, and YOLOv7. The dataset used in this study is a collection of 1324 SEM images with 3265 defect instances of semiconductor line space patterns with defects. These defects are classified as belonging to one of five possible classes: line collapse, gap, probable gap, bridge, or micro-bridge. From the results of the paper, the YOLOv7 is the best one step detector while only losing out to the best multi-step detector by 2% in mean Average Precision.

Empirical Study Of The Performance Of Object Detection Methods On Road Marking Dataset

[11] Compares the different performance of object detection methods on the Road Marking Dataset also known as the CeyMo Road Marking dataset. Numerous methods are implemented within this paper, some methods include, DDOD, TOOD, Guided Anchoring Cascade R-CNN, Sparse R-CNN, YOLOv7, YOLOX, and YOLOF. Introduced as a new upgrade in the YOLO family, YOLOv7 provides superior performance compared to the previous methods, i.e., YOLOv5. The models are then trained and tested using the dataset as mentioned before, the different results show that the YOLOv7 model has the highest mean average precision with 69.5%, beating other models such even in the two-stage category. Some classes that were detected in the dataset are: Diamond, Pedestrian Crossing, Bus Lane, and others.

Yolov7: Trainable Bag-Of-Freebies Sets New State-Of-The-Art For Real-Time Object Detectors

This paper introduces the new YOLOv7 model and compares its performance of the COCO dataset to past YOLO models. The paper claims that YOLOv7 surpasses all known object detectors and has the highest accuracy 56.8% AP among all known real-time object detectors. The paper uses the Microsoft COCO Dataset to train the YOLOv7 model from scratch in object detection. This paper proposes a new architecture of real-time object detector and the corresponding model scaling method. Based on the results of the developed YOLOv7 model the results achieved state-of-theart results [12].

Collision-Line Counting Method Using Deepsort To Count Pedestrian Flow Density And Hungary Algorithm

The aim of the paper [13] is to utilize YOLO algorithm together with DeepSORT in creating model that detects and measure multiple detected objects in real-time. Particularly, this measures the density of people coming in and out of an area through line collision counting method. To train the model, the researchers used the Market1501 dataset which contains 751 people in the training set and 750 people in the test set. Researchers proposed the combination of YOLOv4 and DeepSORT algorithms to measure the density of people coming in and out of a place. In doing so, 5 Kalman Filter and Hungarian Algorithm is applied to predict the motion and the data of the detected object. Furthermore, density statistics is also done which uses ID and trajectory data of the detected person to calculate the amount of people and motion in which they will proceed. Results show that there is a 5.8% difference in using the proposed method in comparison to the manual method. In validating the data 5 times, it is observed that 1st, 2nd, and 5th trials showed same results, thus resulting in a high performance on the proposed method. However, to improve future works datasets needs to be improved and trained on the model to improve its accuracy. Overall, it can be stated that this proposed method could be applied not just on measuring and tracking pedestrians but also be utilized in traffic flow such that through DeepSORT the model is able to predict the motion of multiple objects detected in real time.

Increasing Efficiency In Counting Number Of Motorcycle With Object Relation Matching

The paper [14] focuses on counting motorcycle riders using object detection and object tracking methods through YOLOv4 and DeepSORT with object relation matching to improve the performance of the model wherein some errors encountered by the researchers when measuring the number of riders are alleviated. The researchers utilized video inputs of motorcycle riders as its dataset wherein each frame is converted into RGB, and image size changed. The process consists of object detection using YOLOv4 and tracked by DeepSORT using object ID. Further, object relation matching determines the relation among the rider and the motorcycle wherein same motorcycles, but different riders are not identified as one. Results show that through object relation matching the accuracy of the model increased from 68.29% to 80.99 %, thus improving the errors encountered when counting the objects. Despite good results, improvement such as limited dataset, changing camera angles, and quality of the CCTV data needs to be further assessed to increase efficiency of the proposed method. Overall, the research identified the limitation that DeepSORT algorithm has in counting objects that may look similar and increased its efficiency. With this, not only can the proposed method be used in counting motorcycles but also in other areas such as measuring traffic, crowd densities, and products.

III. Documentation of Code

The activity at hand involves optimizing the hyperparameter settings for the YOLOv7 + DeepSORT model used in Motorcycle Rider Helmet detection. These settings are based on previous activities and are crucial for improving the model's performance. Key hyperparameters such as learning rate, batch size, epochs, and image size are adjusted to achieve better results. In this activity, all of these configurations are different in regards to their sizes. However, all of it has the same image size.

For a quick refresher, batch size impacts the number of photos processed per iteration, whereas learning rate controls the step size of the optimizer during training. The quantity of epochs controls the number of times the full training dataset is processed, while image size affects the dimensions of the input images. Moreover, all of the hyperparameter settings will have the same commands or line of codes but will only be different with the values inputted.

Import Yolov7 Github

```
import torch
import os
from IPython.display import Image, clear_output  # to display images

print(f"Setup complete. Using torch {torch.__version__} ({torch.cuda.get_device_properties(0).name if torch.cuda.is_available() else 'CPU'})")
```

Importing the required libraries is the first step in the procedure. Torch for deep learning operations, os for operating system interface, and IPython are also included in this section of the code. Within the Ipython environment, a display for showing images. The version of torch being used, together with whether a GPU or CPU is being used for computing, are all listed in the message that follows the print statement that confirms setup completion.

```
# Download YOLOv7 repository and install requirements
!git clone https://github.com/WongKinYiu/yolov7
%cd yolov7
!pip install -r requirements.txt
```

```
Requirement already satisfied: psutil in /usr/local/lib/python3.10/dist-packages (from -r requirements.txt (line 35)) (5.9.5)
Collecting thop
  Downloading thop-0.1.1.post2209072238-py3-none-any.whl (15 kB)
Requirement already satisfied: python-dateutil>=2.7 in /usr/local/lib/python3.10/dist-packages (from matplotlib>=3.2.2->-r requirements.txt (line 4)) (2.8.2)
Requirement already satisfied: pyparsing>=2.3.1 in /usr/local/lib/python3.10/dist-packages (from matplotlib>=3.2.2->-r requirements.txt (line 4)) (3.0.9)
```

Next is the downloading and installation of the YOLOv7 repository along with its requirements. The !git clone command clones the repository from the specified URL into the current working directory. The %cd command changes the working directory to the newly cloned repository. Lastly, the !pip install -r requirements.txt command installs the necessary Python packages listed in the requirements.txt file to ensure all dependencies are met for running the YOLOv7 model.

```
# download COCO starting checkpoint
!wget https://github.com/WongKinYiu/yolov7/releases/download/v0.1/yolov7_training.pt
```

For this section, !wget command is used for the downloading of the COCO starting checkpoint. Yolov7_training.pt, the checkpoint file, is downloaded from the given URL. The YOLOv7 model was trained using the COCO dataset, and this checkpoint file acts as the model's initial weights. It is crucial for initiating the model training or inference process.

Import Roboflow Dataset

```
!pip install roboflow

from roboflow import Roboflow
rf = Roboflow(api_key="j633EJ6Z8NqqzQDL8xRa")
project = rf.workspace("data-science-61hvy").project("helmet-detection-data-science-4")
dataset = project.version(3).download("yolov7")
```

This step includes the installation of the roboflow package using pip. After installation, it is then imported to the Roboflow class from roboflow and initializes an instance of it, passing the API key as a parameter. The next line specified the project within the Roboflow workspace, and then selected a specific dataset version.

Train YoloV7 model with Hyperparameter Tuning

```
# run this cell to begin training dont forget to change hyperparameters in content/yolov7/data/hyp.scratch.p5.yaml
!python train.py --batch 32 --epochs 75 --img-size 640 --data /content/yolov7/Helmet-Detection-DATA-SCIENCE-4-3/data.yaml --weights 'yolov7_training.pt' --cache
```

In this part of the code, we are to input the hyperparameter given in the instructions. As stated there are 3 hyperparameter settings given. In all configurations of the different hyperparameters, all of them will have the same commands and will have the same image size. However, they will be distinct to each other particular to the values of LR, batch, and epoch sizes

hyp.scratch.p5.yaml

```
lr0: 0.02  # initial learning rate (SGD=1E-2, Adam=1E-3)
lrf: 0.1  # final OneCycleLR learning rate (lr0 * lrf)
momentum: 0.937  # SGD momentum/Adam beta1
weight_decay: 0.0005  # optimizer weight decay 5e-4
warmup_epochs: 3.0  # warmup epochs (fractions ok)
warmup_momentum: 0.8  # warmup initial momentum
```

```
Epoch   gpu_mem       box       obj       cls     total    labels  img_size
 0/49     3.77G   0.08098   0.01388     0.022    0.1169       180       544: 100% 58/58 [01:34<00:00,  1.62s/it]
           Class      Images      Labels          P          R     mAP@.5  mAP@.5:.95: 100% 8/8 [00:21<00:00,  2.66s/it]
             all         490        2527      0.616     0.0279    0.00659     0.00132

Epoch   gpu_mem       box       obj       cls     total    labels  img_size
 1/49     15.6G   0.07603   0.01169   0.01528     0.103       227       544: 100% 58/58 [01:16<00:00,  1.32s/it]
           Class      Images      Labels          P          R     mAP@.5  mAP@.5:.95: 100% 8/8 [00:08<00:00,  1.02s/it]
             all         490        2527    1.5e-05    0.00129   1.96e-07    1.96e-08

Epoch   gpu_mem       box       obj       cls     total    labels  img_size
 2/49     15.6G   0.07862   0.01133   0.01605     0.106       198       544: 100% 58/58 [01:14<00:00,  1.28s/it]
           Class      Images      Labels          P          R     mAP@.5  mAP@.5:.95: 100% 8/8 [00:14<00:00,  1.81s/it]
             all         490        2527    6.84e-05    0.00284   1.33e-06    1.87e-07

Epoch   gpu_mem       box       obj       cls     total    labels  img_size
 3/49     15.6G   0.06956    0.0122   0.01385    0.0956       245       544: 100% 58/58 [01:14<00:00,  1.29s/it]
           Class      Images      Labels          P          R     mAP@.5  mAP@.5:.95: 100% 8/8 [00:05<00:00,  1.37it/s]
             all         490           0          0          0          0           0

Epoch   gpu_mem       box       obj       cls     total    labels  img_size
 4/49     15.6G   0.07475   0.01121   0.01272   0.09868       313       544:  53% 31/58 [00:39<00:34,  1.26s/it]
```

Class	Images	Labels	P	R	mAP@.5	mAP@.5:.95:
all	490	2527	0.203	0.187	0.0906	0.0306
Full_Face	490	382	0.168	0.0916	0.0603	0.0149
Half_Face	490	735	0.208	0.224	0.0954	0.0246
Invalid	490	155	0.187	0.0774	0.0554	0.0169
No_Helmet	490	124	0.229	0.0161	0.0146	0.0057
Rider	490	1131	0.224	0.525	0.227	0.0907

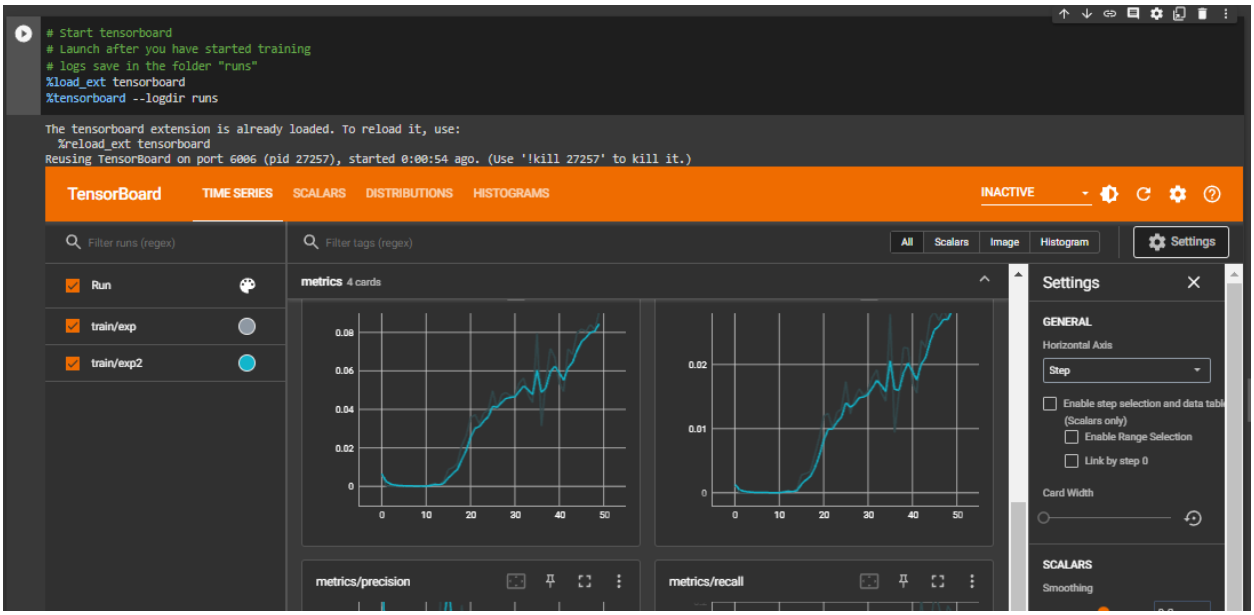
Seen in the images above shows the training process and its results particular to the hyperparameter setting that is configured. In this case, this is the hyperparameter 2 with DeepSORT screenshots. Furthermore, all of the hyperparameters will have the same process that they will go through.

```
# Run evaluation
!python detect.py --weights runs/train/exp2/weights/best.pt --conf 0.1 --source {dataset.location}/test/images
 The image with the result is saved in: runs/detect/exp2/3-videos_mp4-46_jpg.rf.303dfd3317cec576d6cbc44374c7a348.jpg
3 Half_Faces, 7 Riders, Done. (14.2ms) Inference, (1.0ms) NMS
 The image with the result is saved in: runs/detect/exp2/3-videos_mp4-490_jpg.rf.86870399477b38ff465ee78e5a764c0f.jpg
1 Full_Face, 4 Half_Faces, 11 Riders, Done. (13.8ms) Inference, (1.0ms) NMS
```

By executing the detect.py script, the evaluation procedure is started in this code. The best weights file from the designated location and a confidence level of 0.1 are supplied as command-line parameters to the script. The dataset's test photos are used for the evaluation, and the findings are obtained as a result.

This set of code is for importing the glob module for finding the file paths, and the display function from IPython.display for showing the images. The maximum number of photos that can be printed is determined by limit, with the variable i serving as a counter. The display() method and a newline are used to display each image after the glob.glob() function has located all JPG files in the specified directory. The cycle repeats till the number of images is reached.

TensorBoard



This section includes the launching of tensorboard. This is a visualization tool provided by TensorFlow, to monitor and analyze the training processes. The %load_ext tensorboard line loads the TensorBoard extension, allowing it to be used in the colab.The %tensorboard – logdir runs line starts TensorBoard and specifies the directory where the log files are saved. Once it is launched successfully, it provides the user with visualizations such as loss curves, metric, and other useful information that is helpful in further analyses and understanding the training process. Seen in the images above are the different graphs produced in particular to the model's confusion matrix, precision, recall, mean average precisions, MSE, RMSE, and the effects of the hyperparameter adjustments.

DeepSORT

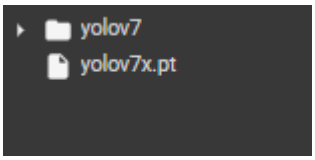
```
#Clone the github repository for YoloV7 + DeepSORT
!git clone https://github.com/deshwalmahesh/yolov7-deepsort-tracking
%cd yolov7-deepsort-tracking

Cloning into 'yolov7-deepsort-tracking'...
remote: Enumerating objects: 186, done.
remote: Counting objects: 100% (28/28), done.
remote: Compressing objects: 100% (22/22), done.
remote: Total 186 (delta 12), reused 9 (delta 6), pack-reused 158
Receiving objects: 100% (186/186), 68.18 MiB | 36.98 MiB/s, done.
Resolving deltas: 100% (48/48), done.
/content/yolov7/yolov7-deepsort-tracking
```

This section is for cloning the github repository for YOLOv7 with DeepSORT. The repository that is to be cloned is indicated by the provided URL. The line %cd yolov7-deepsort-tracking switches to the newly cloned repository as the current working directory. This enables simple navigation and access to the project's code and files.

```
#copy pt file to content folder and rename
!cp /content/yolov7/runs/train/exp2/weights/best.pt /content/yolov7x.pt
```

Now, the pt file is copied and is set in the specified directory or folder. The code snippet effectively duplicates the best.pt file and renames it as yolo7x.pt in the specified directory.



As seen in the image above, this is the result of the process that is previously discussed. Under the folder yolov7, there is a yolov7x.pt.

```
#Make sure to change the different python files before running this cell
#change the yolov7-deepsort/tracking_helpers.py line 247 to classes = ['Full_Face','Half_Face','Invalid','No_Helmet','Rider']
#change the yolov7-deepsort/data/coco.yml line 15 to classes = ['Full_Face','Half_Face','Invalid','No_Helmet','Rider'] and line 12 to nc: 5
from detection_helpers import *
from tracking_helpers import *
from  bridge_wrapper import *
from PIL import Image
#Not required
from google.colab import drive
drive.mount ("/content/drive")
```

```
Mounted at /content/drive
```

The image above provides the snippet in regards to the commands and imports set that are related to setting up the environment and files before running a specific cell. This also includes the modifying the list of classes in the yolov7-deepsort/tracking_helpers.py at line 247 as well as the classes and the number of classes in the yolov7-deepsort/data/coco.yml file. Further, the program import the PIL the required modules, including image, bridge_wrapper, tracking_helpers, and detection_helpers. This also contains a separate import line to mount google drive.

coco.yaml

```
# COCO 2017 dataset http://cocodataset.org

# download command/URL (optional)
download: bash ./scripts/get_coco.sh

# train and val data as 1) directory: path/images/, 2) file: path/
train: ./coco/train2017.txt  # 118287 images
val: ./coco/val2017.txt  # 5000 images
test: ./coco/test-dev2017.txt  # 20288 of 40670 images, submit to

# number of classes
nc: 5

# class names
names: ['Full_Face','Half_Face','Invalid','No_Helmet','Rider']

```

tracking_helpers.py

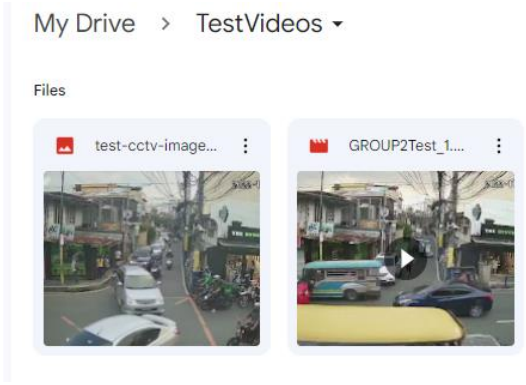
```
    parser.add_argument(
        "--model",
        default="resources/networks/mars-small128.pb",
        help="Path to freezed inference graph protobuf.")
    parser.add_argument(
        "--mot_dir", help="Path to MOTChallenge directory (train or te
        required=True)
    parser.add_argument(
        "--detection_dir", help="Path to custom detections. Defaults t
        "standard MOT detections Directory structure should be the def
        "MOTChallenge structure: [sequence]/det/det.txt", default=None
    parser.add_argument(
        "--output_dir", help="Output directory. Will be created if it
        " exist.", default="detections")
    return parser.parse_args()


def main():
    args = parse_args()
    encoder = create_box_encoder(args.model, batch_size=32)
    generate_detections(encoder, args.mot_dir, args.output_dir,
                        args.detection_dir)


def read_class_names():
    '''
    Raad COCO classes names
    '''
    classes = ['Full_Face','Half_Face','Invalid','No_Helmet','Rider']

```

Model Testing




```
detector = Detector()
model_path = '/content/yolov7x.pt'
detector.load_model(model_path,)

#Make sure that the detector is successfuly loaded try to detect with a test image
#you can use any test image or not run the below code at all
result = detector.detect('/content/drive/MyDrive/TestVideos/test-cctv-image.png', plot_bb = True) # plot_bb = False output the predictions as [x,y,w,h, confidence, class]


if len(result.shape) == 3:# If it is image, convert it to proper image. detector will give "BGR" image
    result = Image.fromarray(cv2.cvtColor(result,cv2.COLOR_BGR2RGB))

#Print the detected test image
result
```

The code creates an instance of the Detector class, which will be utilized for object detection, in this phase. The load_model() method loads the pre-trained model from the provided yolov7x.pt file. This procedure checks that the detector has been successfully loaded and is prepared for inference. A test image is sent to the detect() method of the detector object to confirm this. The code displays the detected objects in the test image with bounding boxes by setting plot_bb to True. If the output is an image in BGR format, it is transformed to the correct RGB format using PIL's Image.fromarray() method and OpenCV's cv2.cvtColor() function. Lastly, the test image that was detected in then printed, giving the results of the detection a visual representation.

```
Fusing layers...
RepConv.fuse_repvgg_block
RepConv.fuse_repvgg_block
RepConv.fuse_repvgg_block
torch.meshgrid: in an upcoming release, it will be required to pass the indexing argument. (Triggered internally at ../aten/src/ATen/native/TensorShape.cpp:3483.)
 Convert model to Traced-model...
 traced_script_module saved!
 model is traced!
```

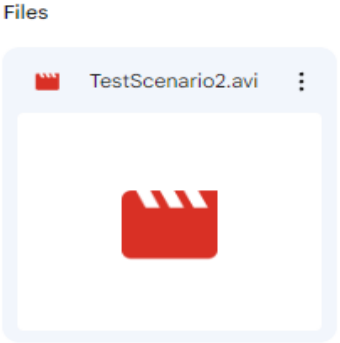

```
[26] # Initialise  class that binds detector and tracker in one class
     # The tracker uses a pre trained DeepSORT model and the created detector with our custom YoloV7 Model
     # DeepSORT uses the detections from the YoloV7 model as an input in order to track the object in a video.
     tracker = YOLOv7_DeepSORT(reID_model_path="/content/yolov7/yolov7-deepsort-tracking/deep_sort/model_weights/mars-small128.pb", detector=detector)
```

Model Video Output

After running all of the codes above, this is the last part of the code wherein it will create a video with the implementation of YOLOv7 and DeepSORT on the video output. After it has finalized the run, it will automatically download the video file.

```
# Change the input and output to be the location of the test video.
tracker.track_video("/content/drive/MyDrive/TestVideos/GROUP2Test_1.mp4", output="/content/drive/MyDrive/TestVideos/TestScenario.avi", show_live = False, skip_frames = 0, count_obj
```

```
Processed frame no: 1417 || Current FPS: 86.36 || Objects tracked: 0
Processed frame no: 1418 || Current FPS: 85.87 || Objects tracked: 0
Processed frame no: 1419 || Current FPS: 68.04 || Objects tracked: 0
Processed frame no: 1420 || Current FPS: 83.05 || Objects tracked: 0
Processed frame no: 1421 || Current FPS: 86.29 || Objects tracked: 0
Tracker ID: 40, Class: Rider,  BBox Coords (xmin, ymin, xmax, ymax): (155, 327, 258, 472)
Processed frame no: 1422 || Current FPS: 40.72 || Objects tracked: 1
Tracker ID: 40, Class: Rider,  BBox Coords (xmin, ymin, xmax, ymax): (150, 328, 253, 472)
Processed frame no: 1423 || Current FPS: 62.66 || Objects tracked: 1
Tracker ID: 40, Class: Rider,  BBox Coords (xmin, ymin, xmax, ymax): (148, 328, 251, 473)
Processed frame no: 1424 || Current FPS: 63.3 || Objects tracked: 1
Tracker ID: 40, Class: Rider,  BBox Coords (xmin, ymin, xmax, ymax): (136, 328, 239, 473)
Processed frame no: 1425 || Current FPS: 86.07 || Objects tracked: 0
```



IV. Hyperparameter Tuning

Yv7 Hyperparameter Settings	LR	Batch	Epochs	Image Size
Hyperparameter 1 + DeepSORT	0.01	64	50	640X640
Hyperparameter 2 + DeepSORT	0.02	32	75	640X640
Hyperparameter 3 + DeepSORT	0.03	16	100	640X640

A. Hyperparameter 1

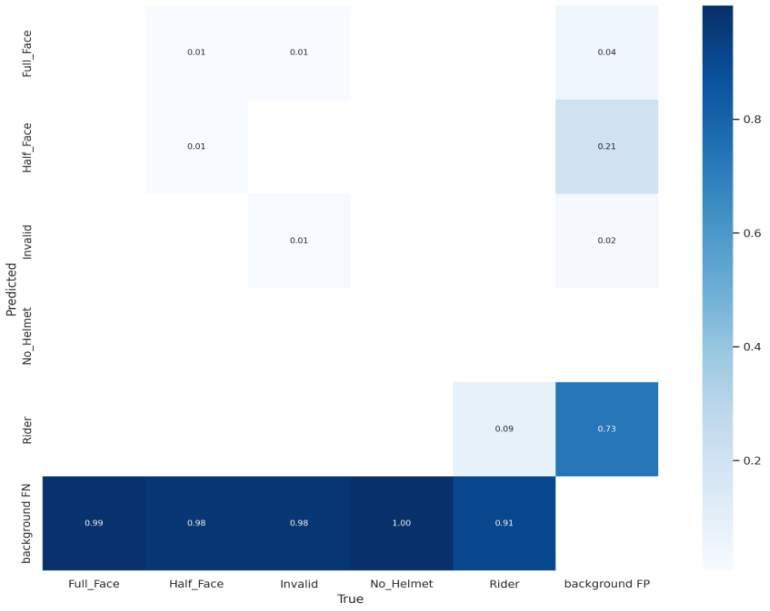
- **Model Execution**

To begin with the model training, we have to adjust the hyperparameters based on the given settings. In this case, the hyperparameter 1 parameters shall be set to 50 epochs, a learning rate of 0.01, image size of 640, and a batch size of 64. As seen on the code below, a "data.yaml" file is called and this yaml file contains the code's learning rate.

```
# run this cell to begin training dont forget to change hyperparameters in content/yolov7/data/hyp.scratch.p5.yaml
!python train.py --batch 64 --epochs 50 --img-size 640 --data /content/yolov7/Helmet-Detection-DATA-SCIENCE-4-3/data.yaml --weights 'yolov7_training.pt' --cache
```

- **Confusion Matrix Results**

The model accurately classified instances in the negative class with true negative values ranging from the lowest value 0.91 from the rider class to the highest value of 1.00 from the No_Helmet class. Despite the No_Helmet class having the highest number of true negative values, this class was underrepresented in the dataset, hence, the model might not be able to detect instances of riders with no helmets.



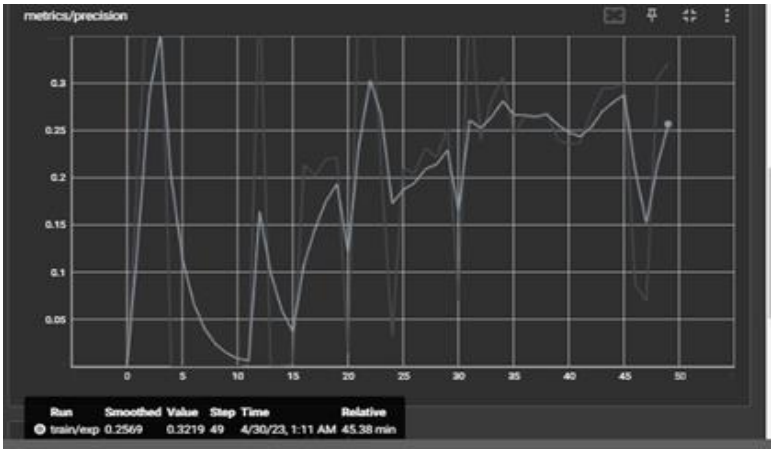
- **Training Results**

Class	Images	Labels	P	R	mAP@.5	mAP@.5:.95:
all	490	2527	0.322	0.115	0.0474	0.0162
Full_Face	490	382	0.104	0.0497	0.0252	0.00721
Half_Face	490	735	0.149	0.0912	0.0373	0.0086
Invalid	490	155	0.184	0.0839	0.0404	0.0131
No_Helmet	490	124	1	0	0.00171	0.000341
Rider	490	1131	0.172	0.348	0.132	0.0517

- **Tensorflow Results**

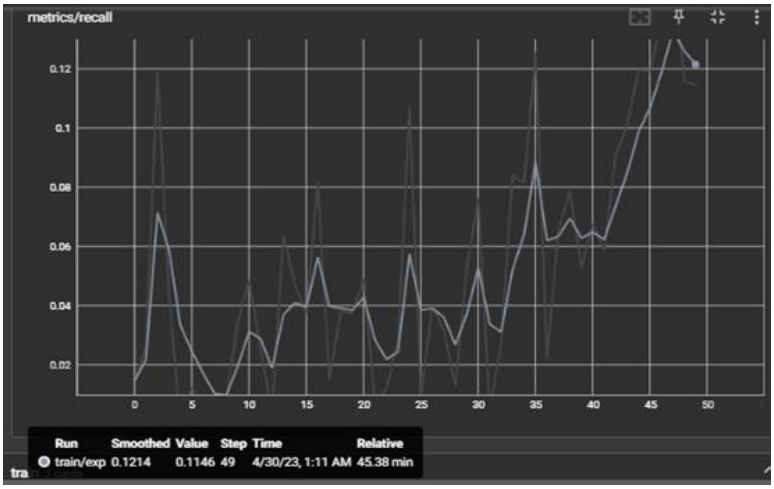
1. Precision

This portion shows the result of how precise or correct the model detects helmets. As seen in the graph below, the final value obtained by the model is at around 0.27.



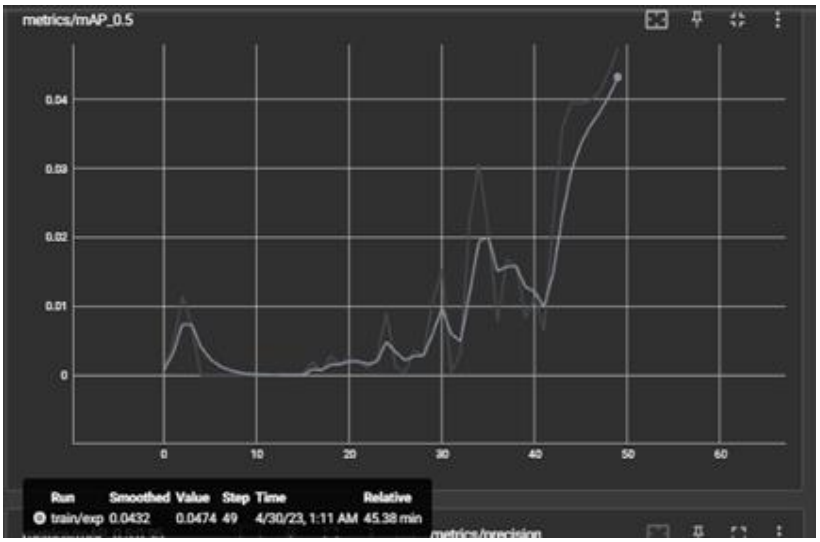
2. Recall

Similar to the precision graph, the lines in the recall graph of hyperparameter 1 are rising and declining. However, the movement gradually kept increasing at the end which shows that the recall value kept improving until it reached the final value at around 0.12. This shows that the model could identify 12% instances of helmets accurately



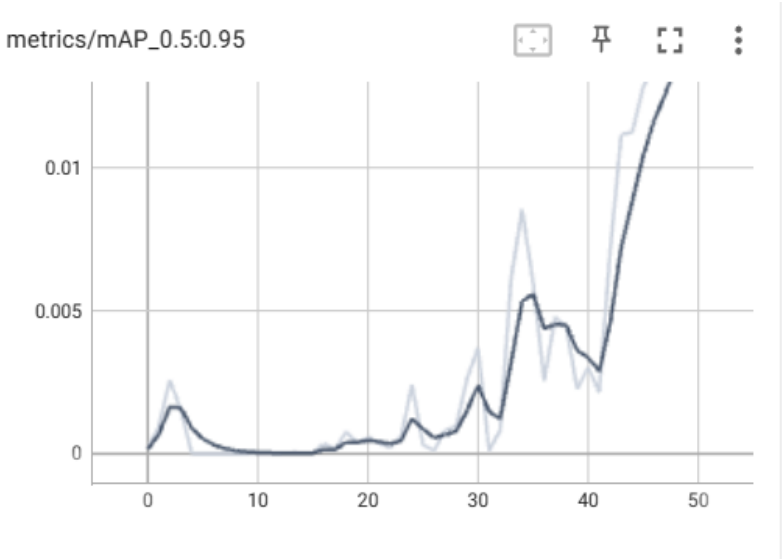
3. mAP 0.5

The value of the mAP obtained at the 0.5 threshold is at 0.04. As seen in the graph, the mAP value first increased when model training was executed and then it declined before gradually rising again. This shows that as the model was being trained, the mAP score kept increasing before it reached the final value.



4. mAP 0.5:0.95

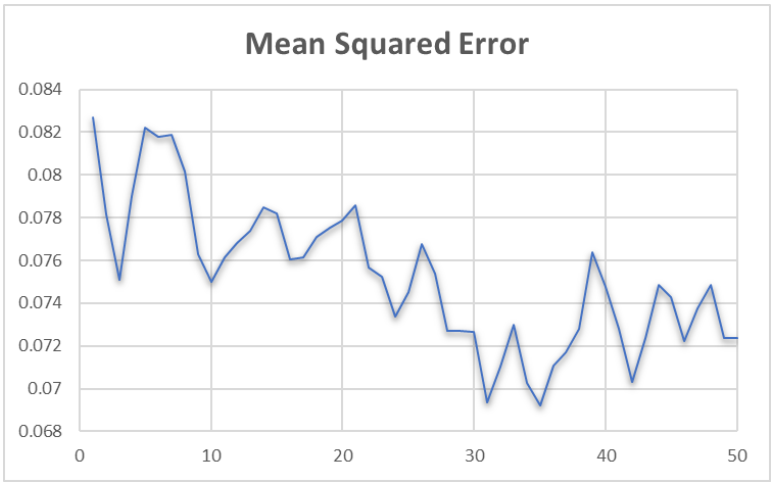
Compared to the mAP score for the 0.5 threshold, the performance of this model was lower. Despite the line having a similar movement - and the value kept rising during training - the final value obtained (at around 0.015) was still significantly lower than the previous mAP score.



- **Additional Model Evaluation**

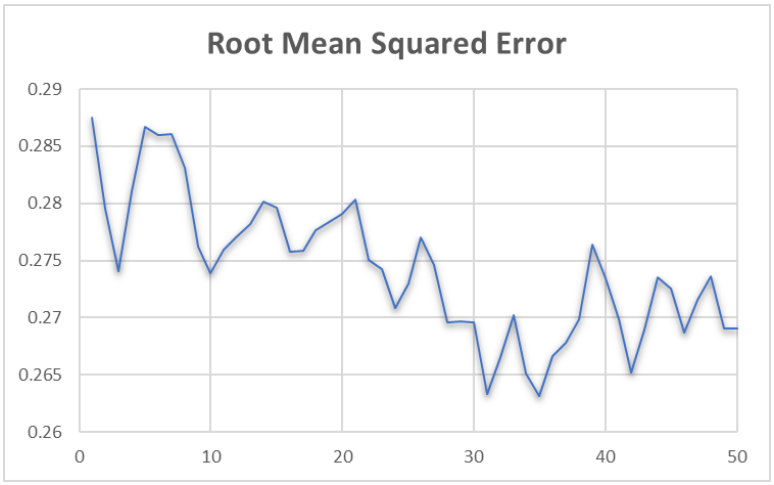
1. MSE

The MSE value decreased from 0.0821 to 0.072. The graph shows that the value of the MSE is fluctuating per epoch. It first starts with the highest value of 0.0821 and has a low prediction error, however, the value decreases over time, reaching to 0.072. Therefore, we could deduce that the performance of the model improved since a lower MSE value indicates a higher prediction accuracy.



2. RMSE

The RMSE graph movement is somewhat similar to the MSE graph wherein the lines are fluctuating and the value of the graphs slowly decreases overtime. The decrease indicates that the model is able to create accurate predictions and errors are also decreasing. The RMSE value starts at around 0.286 and decreases down at around 0.27.



- **Hyperparameter Conclusion**

  Based on our findings, the performance of the model may be enhanced further. The model produced good true negative values for the negative class, but struggled with underrepresented classes such as instances of riders without helmets. Precision and mAP scores may still be improved but the recall value gradually improved over time, indicating that the model has the ability to accurately identify instances of helmets. Lastly, the MSE and RMSE values decreased, portraying that the model's prediction accuracy improved.

B. Hyperparameter 2

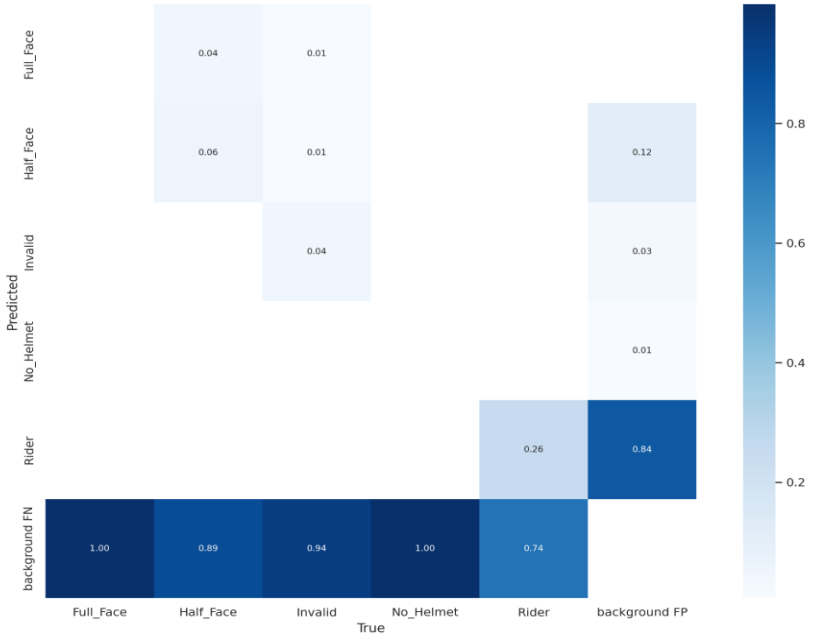
- **Model Execution**

```
# run this cell to begin training dont forget to change hyperparameters in content/yolov7/data/hyp.scratch.p5.yaml
!python train.py --batch 32 --epochs 75 --img-size 640 --data /content/yolov7/Helmet-Detection-DATA-SCIENCE-4-3/data.yaml --weights 'yolov7_training.pt' --cache
```

For Hyperparameter 2, the required parameters are, 75 epocs and a learning rate of 0.02. Similar to hyper parameter 1, the images will have a size of 640x640. The data.yaml will still be utilized here since the learning rate of the code differs per hyperparameter

- **Confusion Matrix Results**

The model was able to classify the different instances with the highest value being 1.00 from the No_Helmet and Full_Face. The lowest value would be the rider which had a value of 0.74. The model was unable to detect the no helmet class and incorrectly identified full face and half face helmets.



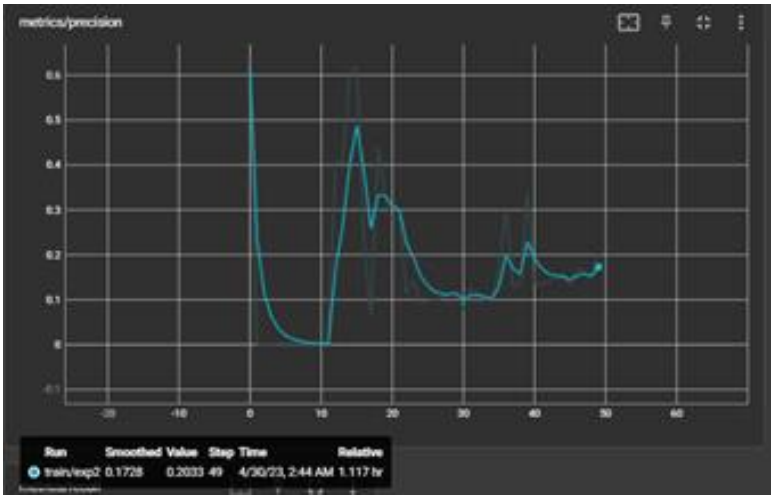
- **Training Results**

Class	Images	Labels	P	R	mAP@.5	mAP@.5:.95:
all	490	2527	0.203	0.187	0.0906	0.0306
Full_Face	490	382	0.168	0.0916	0.0603	0.0149
Half_Face	490	735	0.208	0.224	0.0954	0.0246
Invalid	490	155	0.187	0.0774	0.0554	0.0169
No_Helmet	490	124	0.229	0.0161	0.0146	0.0057
Rider	490	1131	0.224	0.525	0.227	0.0907

- **Tensorflow Results**

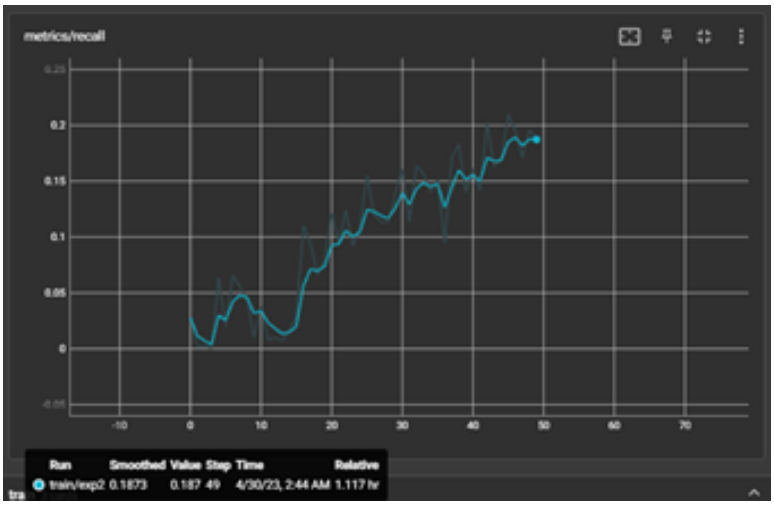
1. Precision

This portion shows the result of how precisely the model detects helmets. The values are fluctuating as the epochs increase. At the start of execution, it showed a massive drop in value followed by a drastic increase as the epoch increased. After reaching the highest value, the value decreased to reach its lowest point. It increased after reaching its lowest point and showed a stable value towards the end of the epoch. It achieved a precision of 17%



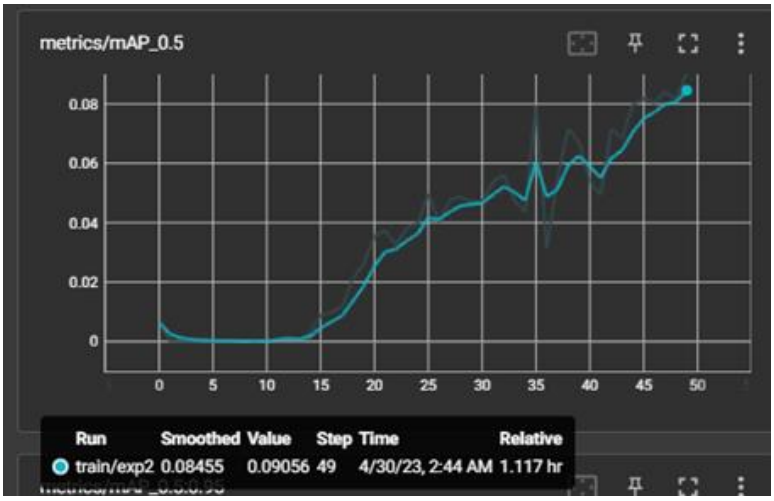
2. Recall

The graph shows a gradual increase in recall value as the epoch increased. It achieved a final value of 0.19. This value indicates that the model can identify 19% of helmets properly



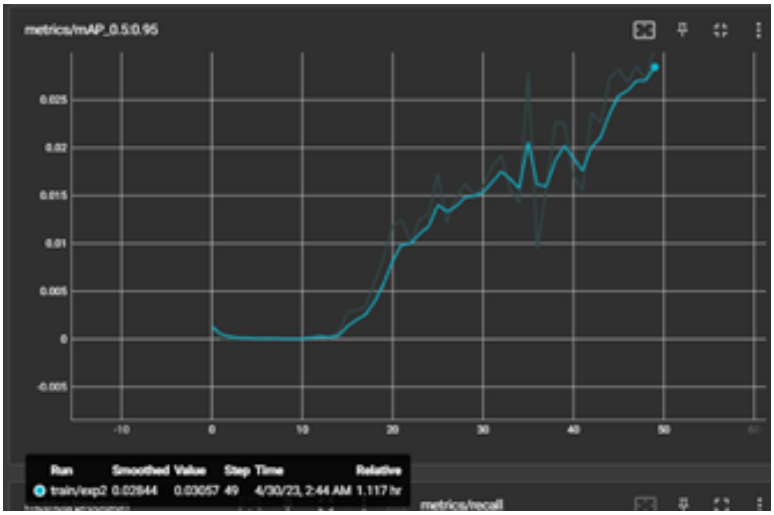
3. mAP 0.5

The map value obtained at a 0.5 threshold is around 0.08. At the start of the model execution, the map value showed no signs of increase or decrease. As the epoch value increased it showed a steady increase in value before reaching its peak. This shoes that as the number of epocs increase the map score will also increase.



4. mAP 0.5:0.95

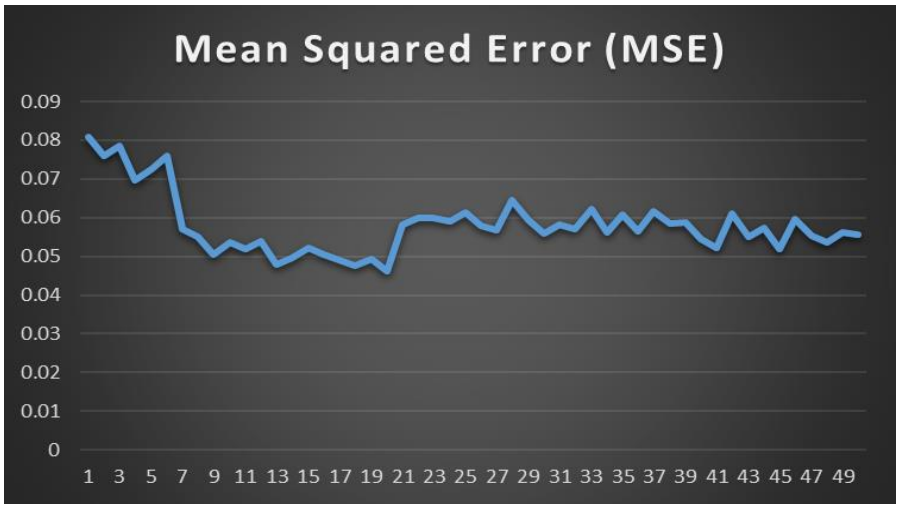
Similar to the graph of mAP 0.5, the graph showed consistent stability at the start but increased as the epochs increased. between epoch 30 to 40 the values began to fluctuate before reaching a stable increase. The mAP 0.5:0.95 achieved a low final score of 0.03 signifying a low accuracy when compared to results of mAP 0.5.



- **Additional Model Evaluation**

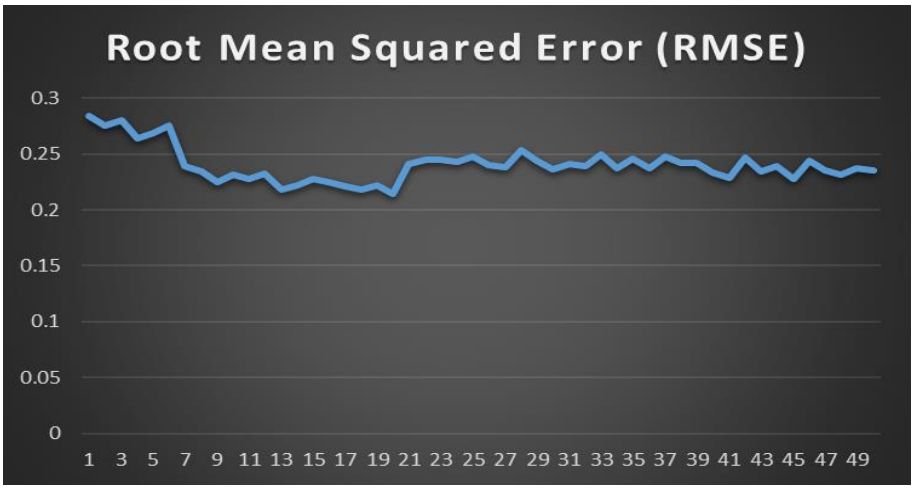
1. MSE

The MSE decreased from 0.08 to around 0.056. The decrease in MSE reached a lowest value of below 0.05 but began to increase before reaching its final value. from epoch 1-20, it showed a decrease but as the number of epochs increased, so did the MSE which could signify the prediction accuracy is most accurate at a specific range of epoch.



2. **RMSE**

Similar to the MSE graph, it showed a decrease in value with the lowest value being at epoch 20. After epoch 20, it increased again and showed stability until the last epoch. Compared the the MSE, the graph showed more stability since the graph in the MSE showed more fluctuations. After epoch 20, the error reduction is less effective. The final value of the RSME is around 0.24



- **Hyperparameter Conclusion**

Hyperparameter 2 had a low precision of 17% and recall of 19%. When compared to hyperparameter 1, the precision of hyperparameter 2 was lower but its recall score was higher. The MSE ended with a value of around 0.056 while the RSME ended with a value of around 0.24.The MSE and RSME also showed a decrease in value which signifies the increase in prediction accuracy. Although the MSE and RSME showed to be more accurate at a specific epoch range. The confusion matrix results show that the model was unable to detect helmets in the most efficient way.

C. Hyperparameter 3

- **Model Execution**

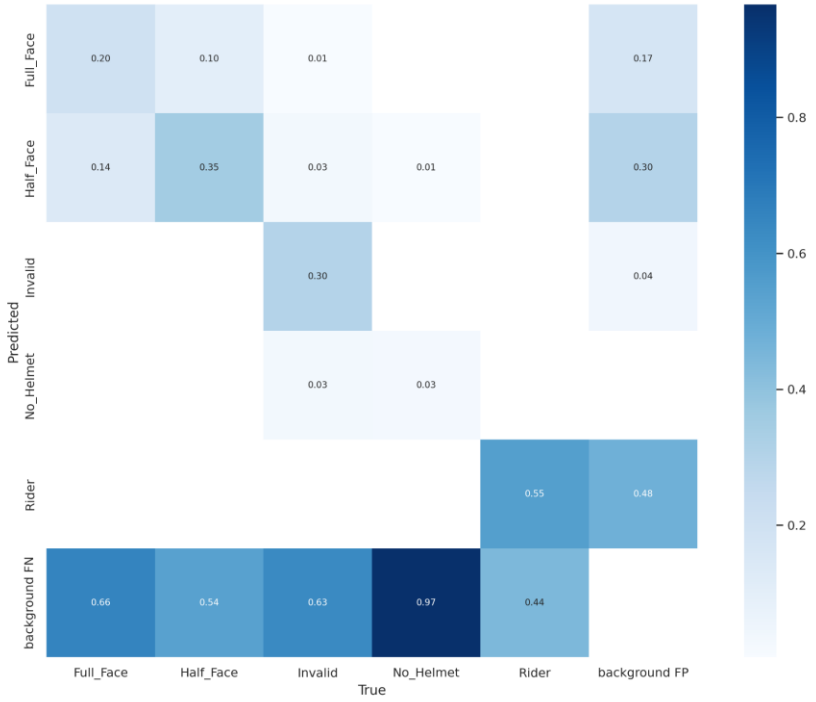
The set of parameters in hyperparameter 3 are 100 epochs, a learning rate of 0.03, image size of 640, and a batch size of 16. As with the other parameters the "data.yaml" file is called and this yaml file contains the code's learning rate of 0.03.

```
# run this cell to begin training dont forget to change hyperparameters in content/yolov7/data/hyp.scratch.p5.yaml
!python train.py --batch 16 --epochs 100 --img-size 640 --data /content/yolov7/Helmet-Detection-DATA-SCIENCE-4-3/data.yaml --weights 'yolov7_training.pt' --cache
```

- **Confusion Matrix Results**

Confusion Matrix shows better detection results but consistent errors with the previous hyperparameters, mostly incorrectly detecting the background as the given classes. Correctly identified metrics have Half_Face at 0.35, invalid at 0.3 and Full_Face at 0.2. No_Helmet is the lowest at 0.03. The Rider class is the highest correctly predicted class at 0.55 with 0.44 incorrectly

predicted. Correctly identified helmet classes are higher than their false positive and true negative misidentifications, indicating that the model can discern differences between helmet classes. The background mispredictions show that the model is still struggling with misidentifying the classes with the background with the highest misprediction being No_Helmet at 0.97 ranging to the lowest 0.44 of the Rider.



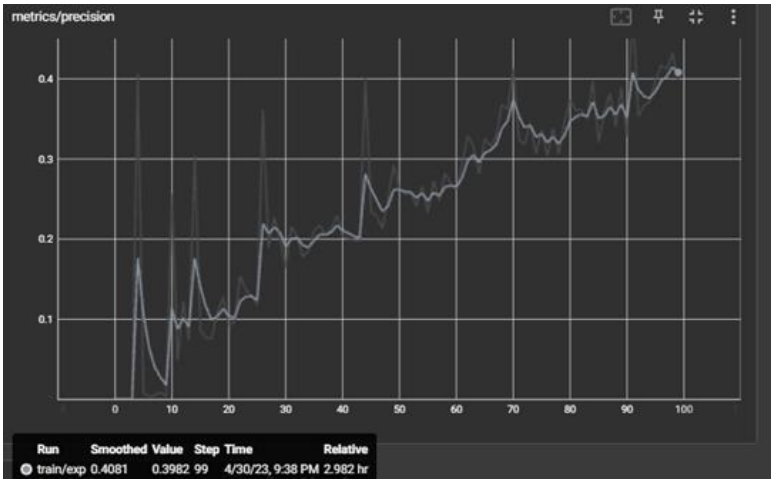
- **Training Results**

Class	Images	Labels	P	R	mAP@.5	mAP@.5:.95:
all	490	2527	0.398	0.382	0.331	0.152
Full_Face	490	382	0.286	0.343	0.237	0.08
Half_Face	490	735	0.411	0.486	0.412	0.16
Invalid	490	155	0.38	0.368	0.34	0.13
No_Helmet	490	124	0.444	0.0565	0.0771	0.0297
Rider	490	1131	0.47	0.656	0.589	0.36

- **Tensorflow Results**

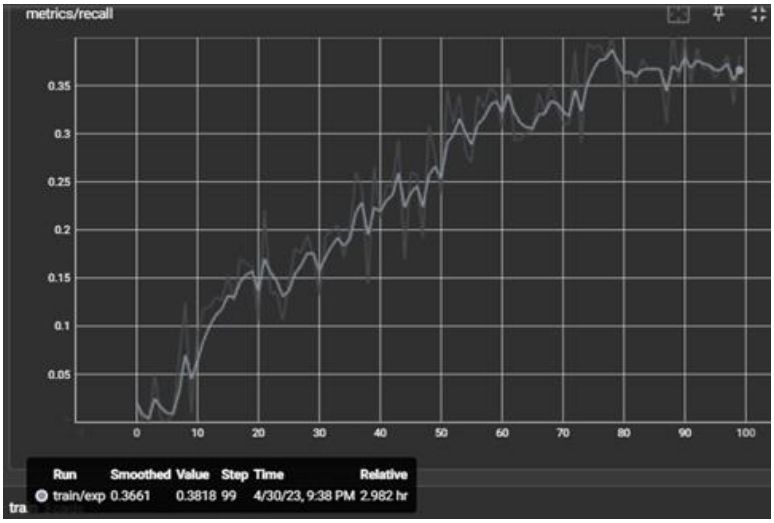
1. Precision

This portion shows the result of how precisely the model detects helmets with a final precision value obtained by the model at around 0.41. The precision of the model rises at an increasingly stable rate as the epoch increases. 41% of the helmets that the model identified as correct/positive were actually correct/positive.



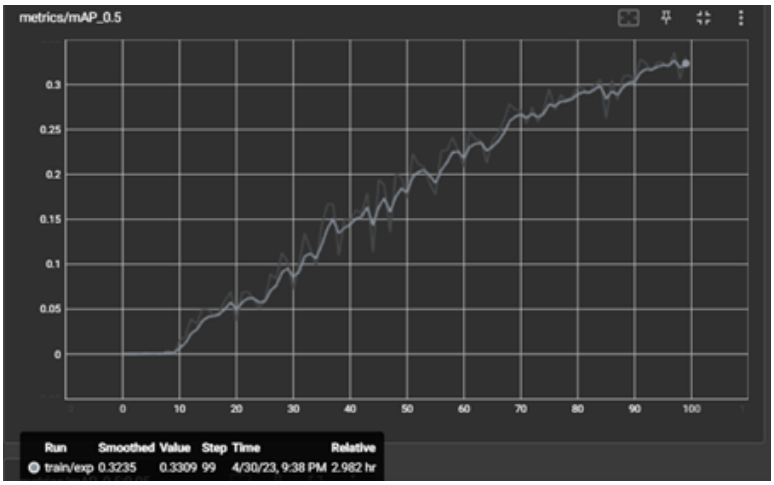
2. Recall

The recall increases as the epoch increases as well. The graph below shows the model peaking at 0.38 recall and finalizing at around 0.36. The volatility of the recall change is higher at the lower stages, further stabilizing as the epoch progresses.



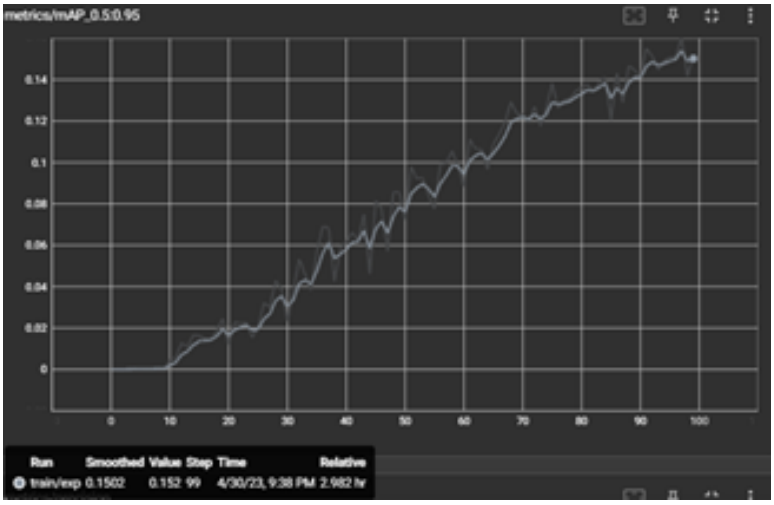
3. mAP 0.5

The average mAP obtained at the 0.5 threshold is at 0.331. mAP growth is largely linear throughout the entire training session as shown in the graph. This could indicate that the model performance in terms of mAP may increase steadily if trained beyond 100 epochs. It also can indicate the stability of hyperparameter 3. The final value is low indicating low object detection capability.



4. mAP 0.5:0.95

As expected the mAP 0.5:0.95 of this model is lower in comparison to the mAP score for the 0.5 threshold. A similar linear growth can be observed between the two graphs, indicating stable growth and potential further growth. The average mAP 0.5:0.95 finalizes at 0.152, which is relatively very low indicating low accuracy.

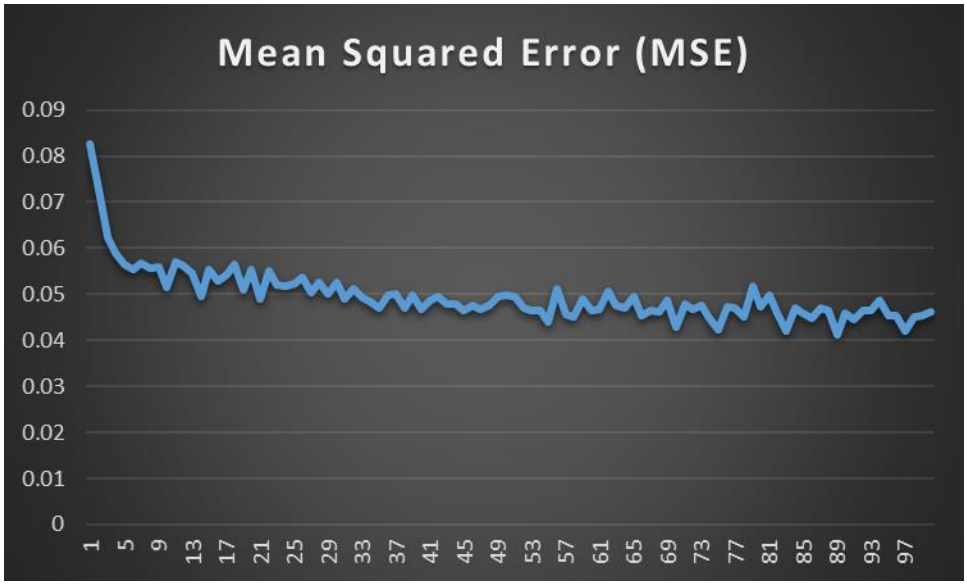


- **Additional Model Evaluation**

1. MSE

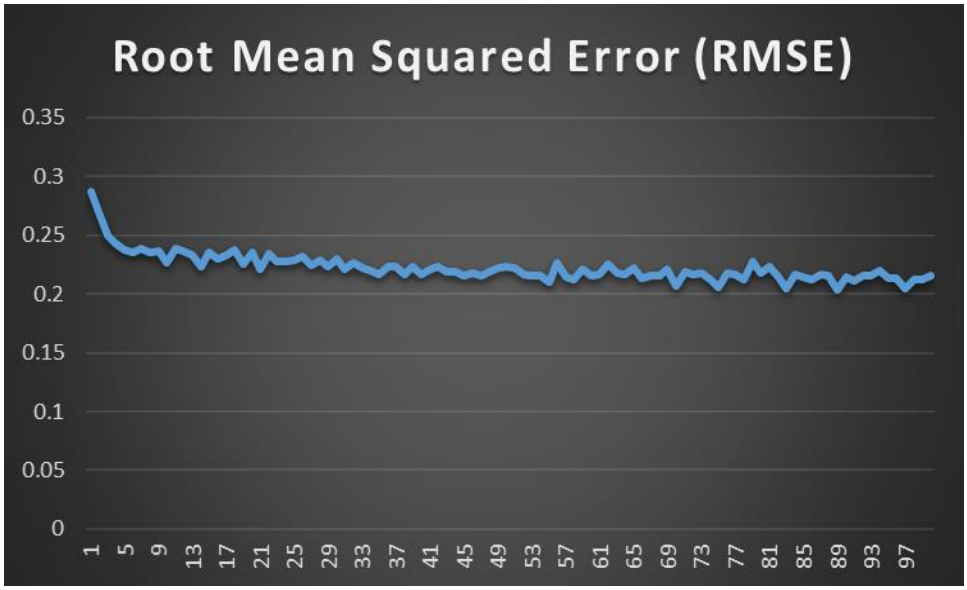
Below is a line graph demonstrating the MSE of the model as the number of epochs are increased according to hyperparameter settings 3. It can be observed that MSE starts at a high measurement at 0.084 meaning that the initial epoch has a substantial amount of errors and low prediction accuracy. A decrease in growth is further observed where the amount of measurement decreases between each epoch also decreases over time. After a sharp decrease from epochs 1 - 8

there is stability from epochs 9 (0.056) and onwards with a constant but slow decrease in MSE measurement over time. This signifies that the initial epochs of 1 - 8 eliminates a substantial amount of detection errors and performance issues, but from epoch 9 has greatly diminished efficacy in error reduction. The final measurement for MSE is 0.045.



2. RMSE

The RMSE graph below is similar to the MSE, starting at a measurement of 0.29. The decrease in growth is also similar as the amount of measurement decreases between each epoch further decreases over time. After a sharp decrease from epochs 1 - 8 there is stability from epochs 9 (0.24) and onwards with a constant but slow decrease in MSE measurement over time. While fluctuations can still be seen, they are not at the magnitude at the first 8 epochs. This further supports the interpretation that the initial epochs of 1 - 8 eliminates a substantial amount of detection errors and performance issues, but epochs greater than 9 have greatly diminished efficacy in error reduction. The final measurement for RMSE is around 0.22.



- **Hyperparameter Conclusion**

Evaluation metrics showed that the hyperparameter set 3 performed relatively well but inaccurately, with precision scoring at 41% and recall around 36%. The mean squared error was decreased to 0.045 while the root-mean-square error ended at 0.22 indicating improved accuracy and reduction in errors. The confusion matrix showed low true positive rates for all classes with high mispredictions on background elements. The rider class had the highest rate of 0.55 and the no helmet class had the lowest true positive rate of 0.03. Results indicate that this hyperparameter 3 setting does not accurately detect helmets effectively.

V. Comparison Analysis of Hyperparameters

Based on the provided performance results from Hyperparameter 1 to Hyperparameter 3, it is observed that the best performance was shown in the last hyperparameter tuning with precision of 41% and recall at 38%. However, based on the performance results from Hyperparameter 1 to Hyperparameter 2 the precision and recall increased due to an increase in batch size and an increase in learning rate. Evidently, it is stated that higher batch sizes leads to lower asymptotic test accuracy while learning rate recovers the lost test accuracy from the higher batch sizes. Despite an increase in precision and recall is shown in hyperparameter 2, it is evident that there is a large gap among the precision and recall values, with precision being 0.6 and recall being 0.187, meaning that the model is returning very few results but correct in labels. Overall, it shows that from the first hyperparameter to the second hyperparameter it showed that the model turned into a highly specialized model.

Further, from the hyperparameter 3 tuning the precision decreased to 41% while the recall increased at 38% thus showing that the values are closer together and thus less overfitting. Evidently, it can be observed that not much change from hyperparameter 2 to 3 as MSE or RMSE from hyperparameter 2 greatly decreases but at Hyperparameter 3 a slow decrease in MSE is shown, thus stating that the model is slowly reducing its performance in identifying errors. Other than this, true positive rates from hyperparameter 2 to 3 increased with half-face being the helmet class that is best classified by the models. Overall, from hyperparameter 1 it is observed that the model greatly increased its performance from its precision and recall increasing as the model is being adjusted.

Overall, although the model did not result in very high precision and recall values, the model is able to develop by decreasing its MSE and RMSE all throughout the tuning. It is also seen that true positive values increase and the model is able to find more correct predictions all throughout the training. However, in comparison to the previous models done by the users, it is evident that the performance in using YOLOv7 and DeepSORT decreased in comparison to the YOLOv5 model. This is the case such that YOLOv7 is sensitive to changes in lighting and environmental conditions, thus making it difficult to detect objects at different scales as well as small objects. Although YOLOv7 has high computational performance in comparison to its previous models, applying it to real-word may be inconvenient as it is a sensitive model, thus why a decrease in performance resulted in comparison to the other models.

VI. Conclusion

In conclusion, the comparison of hyperparameters revealed that changing several hyperparameters, such as batch size, learning rate, and MSE, can significantly enhance the model's performance. It was noticed that the precision and recall values increased when the model was changed, suggesting that the model was becoming more specialized. Even though the recent hyperparameter tuning resulted in a drop in precision, the model was becoming less overfit and getting better at spotting errors as seen by an improvement in recall and a fall in MSE and RMSE. It should be mentioned, nevertheless, that the YOLOv7 model employed in this study was discovered to be sensitive to changes in lighting and ambient conditions, making it challenging to detect items of various sizes and minute objects. It may not be the best model for real-world applications despite its excellent computational performance. Overall, the comparison of hyperparameters demonstrated how crucial it is to change different hyperparameters to enhance a model's performance and how crucial it is to take into account the constraints of various models in real-world scenarios.

VII. Recommendation

Due to the relatively low performance of the models across all hyperparameters, we recommend reviewing and improving the dataset. The researchers recommend two related points to improve the dataset: Provide images of better visual fidelity and quantity, and ensure uniformity among annotations. Due to the low resolution, graininess, and overall low visual fidelity of the CCTV images from the dataset, it is unlikely for the models to extract meaningful patterns that would help in object detection. This also makes it hard for annotators to correctly annotate bounding boxes as some images may appear distorted. Uniformity in annotations also plays a crucial part in creating a better dataset. Since large datasets are expected to have multiple annotators, guidelines should be followed as to how the dataset is to be annotated. For example in this project, the annotators should annotate the entire object (the rider and the motorcycle) rather than just the object of interest (helmet). The researchers posit that these points would allow future researchers to achieve better results using the same technologies used in this project.

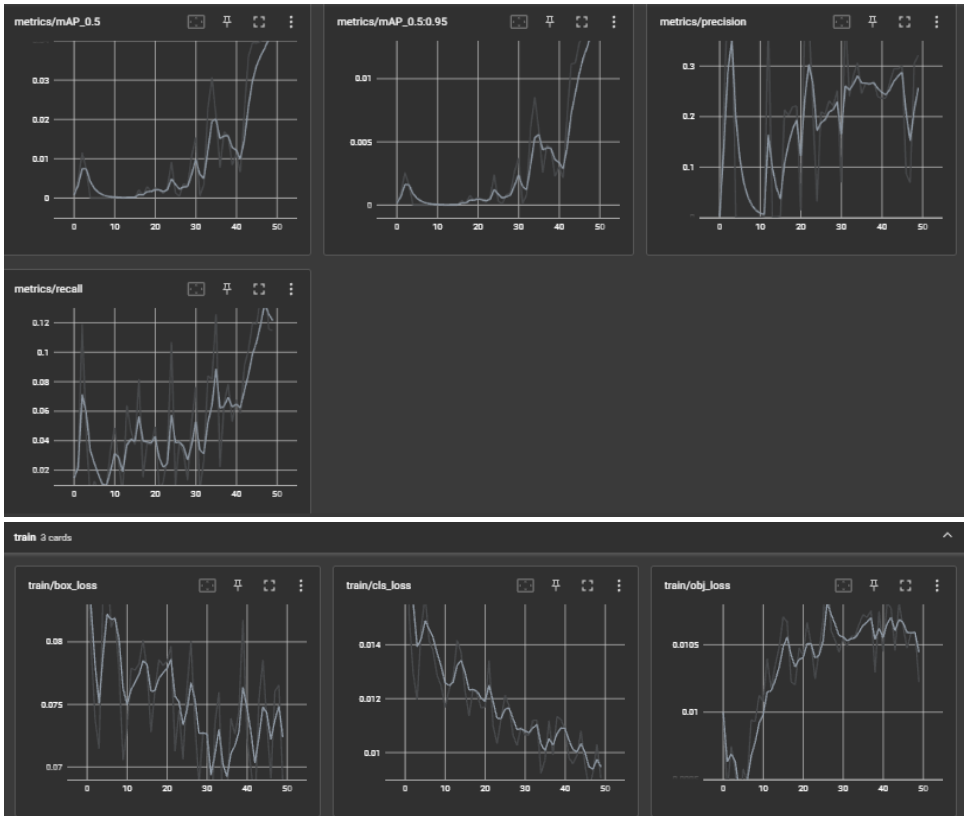
VIII. Other Requirements

A. Roboflow Dataset Link

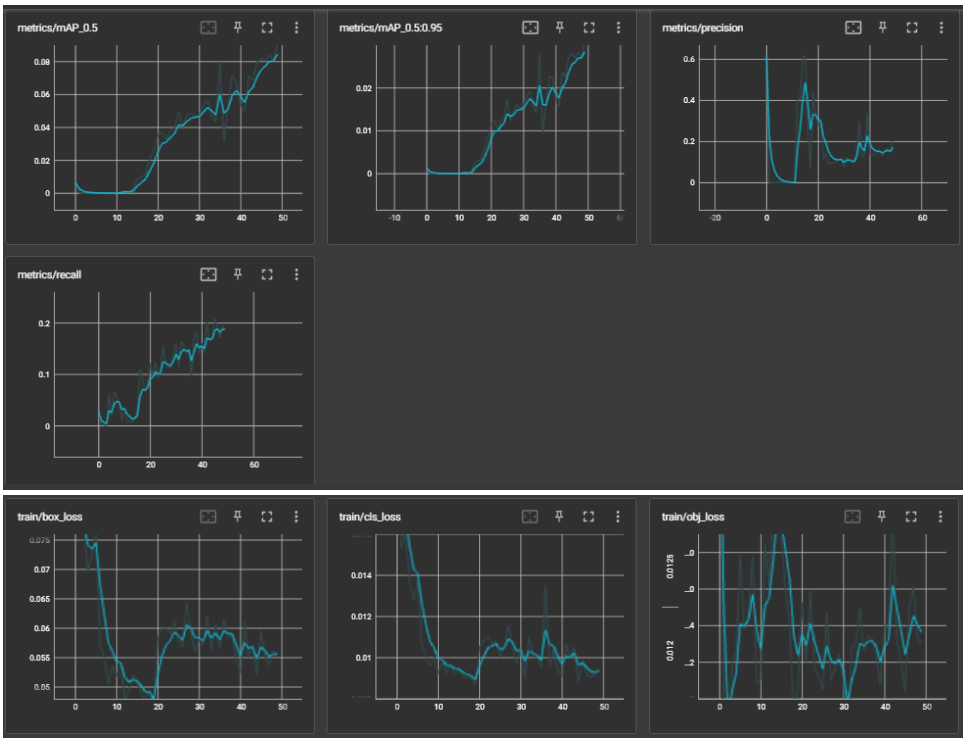
https://app.roboflow.com/data-science-61hvy/helmet-detection-data-science-4/1

B. Screenshot of Tensorboard Results

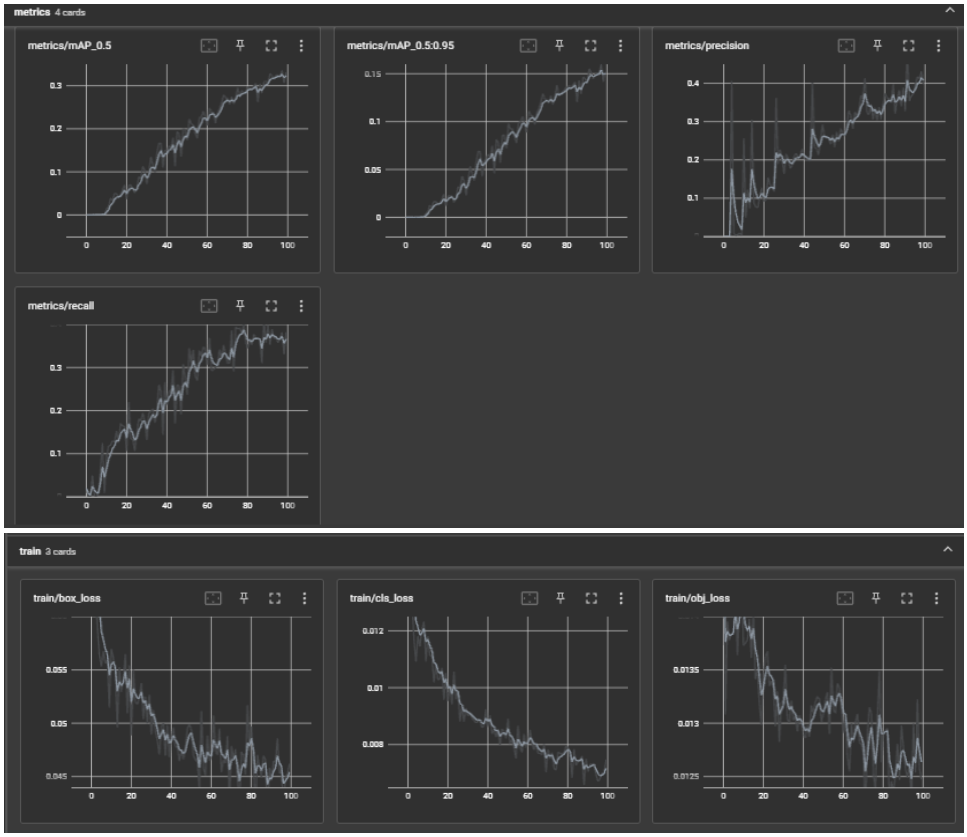
- Hyperparameter 1



- Hyperparameter 2



- Hyperparameter 3



C. Images of different classes/labels based on time (Class image only)

- Hyperparameter 1
  - Train

  - Test Labels

  - Test Prediction

- Hyperparameter 2
  - Train

  - Test Labels

  - Test Prediction

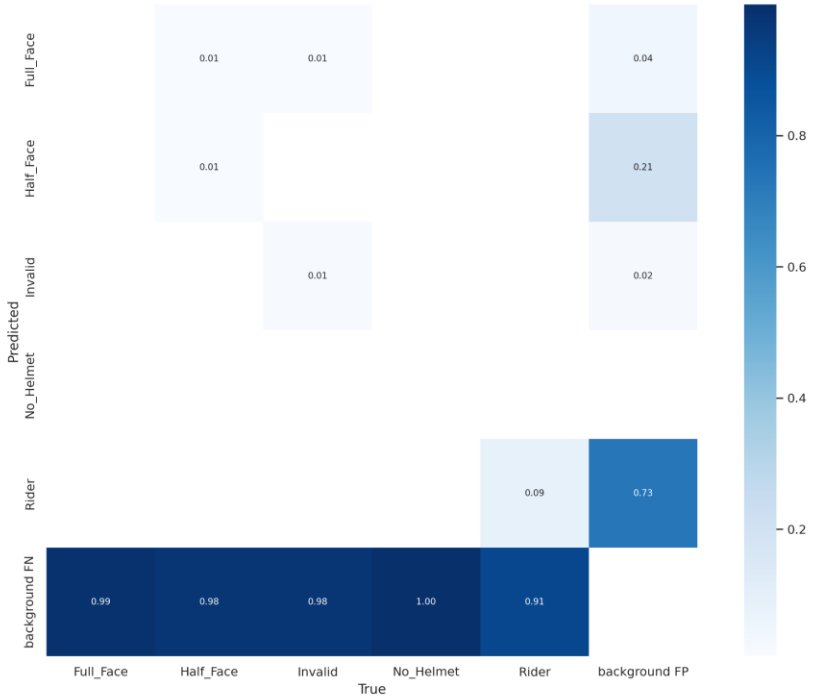
- Hyperparameter 3
  - Train

  - Test Labels

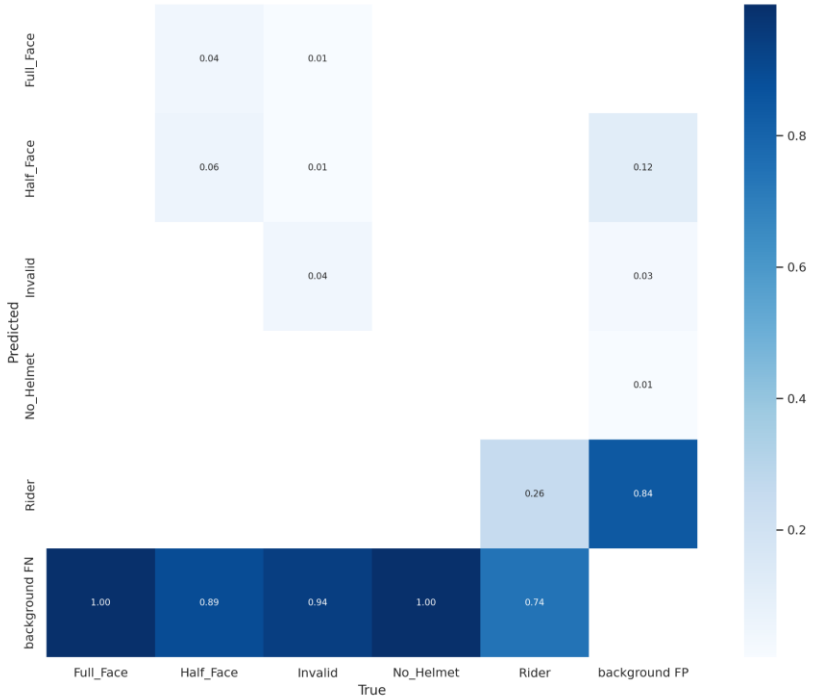
  - Test Prediction

D. Screenshots of confusion matrix vs actual and predicted

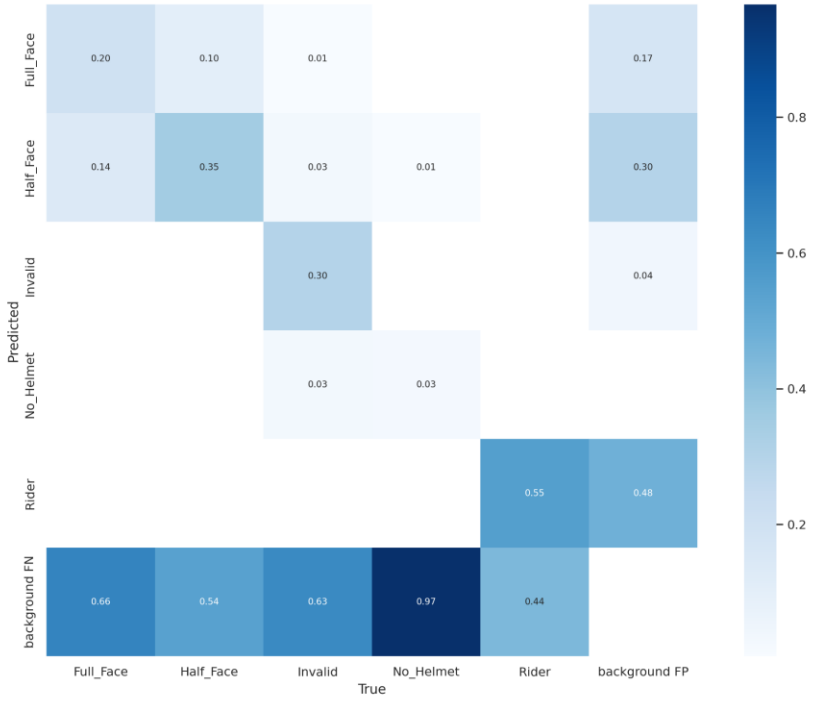
- Hyperparameter 1



- Hyperparameter 2



- Hyperparameter 3



IX. Google Links

A. Sample video output of your model

- YOLOv7 + Deep SORT_Hyperparameter 1

https://drive.google.com/file/d/1TqiESyQnM7iLpuMH2BlqA5zfn3pHnbkx/view?usp=share_link

- YOLOv7 + Deep SORT_Hyperparameter 2

https://drive.google.com/drive/folders/1_xwmjA4mBu_8X4qoCTNwDJXPEqC608qW?usp=share_link

- YOLOv7 + Deep SORT_Hyperparameter 3

https://drive.google.com/file/d/1w3v_cShrWIKVO-Rob0P-uLrzYHo9e3ZX/view?usp=share_link

B. Model link (Google Collab)

- YOLOv7 + Deep SORT_Hyperparameter 1

https://colab.research.google.com/drive/1WvnC00c6jDQnSfqFZNaX0qFzM50gDFZn?usp=sharing

- YOLOv7 + Deep SORT_Hyperparameter 2

https://drive.google.com/file/d/1rns1-OyJgIbNSuTEcQ-SCeuDnEiKGPzq/view?usp=share_link

- YOLOv7 + Deep SORT_Hyperparameter 3

https://drive.google.com/file/d/1lhsRhnrdaUqnNIHPOqAEob7Wh7af9kw8/view?usp=share_link

C. Compilation of related literature at least 10 (Compress all PDFs, include in submission)

- Reference List

https://docs.google.com/document/d/1bpr0f_cmBgPIx7Pyqg2d1i6pL6TQ_uH0Vu5zrXXJANQ/edit#heading=h.xmsms6qnaivv

- Compilation of all PDFS

https://drive.google.com/drive/u/2/folders/1oRTFhSzfehDkSO_aegYwbX_nCIEx037Z

X. References

[1] D. N. -N. Tran, L. H. Pham, H. -H. Nguyen and J. W. Jeon, "City-Scale Multi-Camera Vehicle Tracking of Vehicles based on YOLOv7," 2022 IEEE International Conference on Consumer Electronics-Asia (ICCE-Asia), Yeosu, Korea, Republic of, 2022, pp. 1-4, doi: 10.1109/ICCE-Asia57006.2022.9954809.

[2] B. Hu, M. Zhu, L. Chen, L. Huang, P. Chen and M. He, "Tree species identification method based on improved YOLOv7," 2022 IEEE 8th International Conference on Cloud Computing and Intelligent Systems (CCIS), Chengdu, China, 2022, pp. 622-627, doi: 10.1109/CCIS57298.2022.10016392.

[3 N. D. T. Yung, W. K. Wong, F. H. Juwono and Z. A. Sim, "Safety Helmet Detection Using Deep Learning: Implementation and Comparative Study Using YOLOv5, YOLOv6, and YOLOv7," 2022 International Conference on Green Energy, Computing and Sustainable Technology (GECOST), Miri Sarawak, Malaysia, 2022, pp. 164-170, doi: 10.1109/GECOST55694.2022.10010490.

[4] Y. Wang and H. Yang, "Multi-target Pedestrian Tracking Based on YOLOv5 and DeepSORT," 2022 IEEE Asia-Pacific Conference on Image Processing, Electronics and Computers (IPEC), Dalian, China, 2022, pp. 508-514, doi: 10.1109/IPEC54454.2022.9777554.

[5] M. I. H. Azhar, F. H. K. Zaman, N. M. Tahir and H. Hashim, "People Tracking System Using DeepSORT," 2020 10th IEEE International Conference on Control System, Computing and Engineering (ICCSCE), Penang, Malaysia, 2020, pp. 137-141, doi: 10.1109/ICCSCE50387.2020.9204956.

[6] Y. Gai, W. He and Z. Zhou, "Pedestrian Target Tracking Based On DeepSORT With YOLOv5," 2021 2nd International Conference on Computer Engineering and Intelligent Control (ICCEIC), Chongqing, China, 2021, pp. 1-5, doi: 10.1109/ICCEIC54227.2021.00008.

[7] Hongwen Du, Wenzhong Zhu, Ke Peng, and Weifu Li. 2022. Improved high speed flame detection method based on Yolov7. Open Journal of Applied Sciences 12, 12 (2022), 2004– 2018.

[8] Zipei Pan, Jing Zhu, Xizhe Bao, Jingyi Lin, and Jiahui Ming. 2022. Research on volleyball players tracking based on improved DeepSORT. 2022 4th International Conference on Communications, Information System and Computer Engineering (CISCE) (2022). DOI:http://dx.doi.org/10.1109/cisce55963.2022.9851084

[9] Fuheng Guo zand Yi Xu. 2022. Vehicle Analysis System based on DeepSORT and yolov5. 2022 3rd International Conference on Computer Vision, Image and Deep Learning & International Conference on Computer Engineering and Applications (CVIDL & ICCEA) (2022). DOI:http://dx.doi.org/10.1109/cvidliccea56201.2022.9824363

[10] Enrique Dehaerne, Bappaditya Dey, and Sandip Halder. 2022. A comparative study of deep-learning object detectors for semiconductor defect detection. 2022 29th IEEE International Conference on Electronics, Circuits and Systems (ICECS) (2022). DOI:http://dx.doi.org/10.1109/icecs202256217.2022.9971022

[11] Nguyen D. Vo, Vy P. Le, Thien C. Lai, Khanh B. Duong, and Yen Tran. 2022. Empirical study of the performance of object detection methods on road marking dataset. 2022 RIVF International Conference on Computing and Communication Technologies (RIVF) (2022). DOI:http://dx.doi.org/10.1109/rivf55975.2022.10013909

[12] Chien-Yao Wang, Alexey Bochkovskiy, and Hong-Yuan Mark Liao. 2022. Yolov7: Trainable bag-of-freebies sets new state-of-the-art for real-time object detectors. (July 2022). Retrieved January 22, 2023 from https://arxiv.org/abs/2207.02696

[13] Yuncheng Pei, Haiying Liu, and Qiancheng Bei. 2021. Collision-line counting method using DeepSORT to count pedestrian flow density and Hungary algorithm. 2021 IEEE 3rd International Conference on Civil Aviation Safety and Information Technology (ICCASIT) (2021). DOI:http://dx.doi.org/10.1109/iccasit53235.2021.9633356

[14] Boonyakon Tongmonwit and Kreangsak Tamee. 2022. Increasing efficiency in counting number of motorcycle with object relation matching. 2022 Joint International Conference on Digital Arts, Media and Technology with ECTI Northern Section Conference on Electrical, Electronics, Computer and Telecommunications Engineering (ECTI DAMT & NCON) (2022). DOI:http://dx.doi.org/10.1109/ectidamtncon53731.2022.9720371