Mapúa University

School Of Information Technology

M3-SA YOLOv7 + DeepSORT

Chapter Project (BM2)

by:

Group 7	**Group 8**	**Group 9**
GENETA, Daniel	SUBAAN, Val Rindel S.	LACHICA, Izaac Manuelle T.
LAGUNA, Hans		VILLARAMA, Red Stephen E.
Group 10	**Group 11**	**Group 12**
RIEGO, Gerome	CORTEZ, Mark	PASIA, Justin Carlos
MASACAYAN, Alosius		TOLENTINO, Ezekiel
Group 22	**Group 23**	**Group 24**
GAVINO, Leandro B.	BELTRAN, Angelo James V.	APDIAN, Jose Angelo
PEREZ, Hobie	ARAÑEZ, Yñigo Erick G.	LABRADO, Joshua Reynald

Submitted to:

Prof. John Paul Q. Tomas

Date:

May 2, 2023

658 Muralla St., Intramuros, Manila, 1002, Philippines
Tel. No. : +63(2) 247-5000 local 5804
Telefax : +63(2) 891-0800
Web : www.mapua.edu.ph

I. Introduction 2
II. Merging and Augmenting Dataset Annotations 4
III. Applying Hyperparameters 8
IV. Hyperparameter 1 12
A. **Training** 12
B. **Testing** 15
C. **Screenshot and Evaluation of Tensorboard results** 16
D. **Confusion Matrix** 25
E. **Model Output** 26
V. Hyperparameter 2 27
A. **Training** 27
B. **Testing** 29
C. **Tensorboard results** 30
D. **Confusion Matrix** 39
E. **Model Output** 40
VI. Hyperparameter 3 41
A. **Training** 41
B. **Testing** 43
C. **Tensorboard results** 45
D. **Confusion Matrix** 53
E. **Model Output** 54
VII. Conclusion and Comparison of Hyperparameters 55
VIII. Recommendation 55
IX. Related Literature 56
X. Dataset Link 65
XI. Google Collab Links 65
XII. References 66

658 Muralla St., Intramuros, Manila, 1002, Philippines
Tel. No. : +63(2) 247-5000 local 5804
Telefax : +63(2) 891-0800
Web : www.mapua.edu.ph

Introduction

This chapter project involved all 9 groups performing model training for YOLOv7 motorcycle detection and tracking with DeepSORT for the 6-7pm time scenario. The groups were provided with CCTV footage from 6-7pm to use as our training data. This footage contained numerous instances of motorcycles moving through the frame, providing us with a dataset to train our models on. However, the models could be trained, the dataset had to be manually annotated. This involved going through the footage frame by frame and labeling each instance of a motorcycle. The goal of this chapter project was to train three different YOLOv7 models, each with different hyperparameters, in order to evaluate which model performed the best. Through this project, we aimed to gain a deeper understanding of the capabilities and limitations of YOLOv7 and DEEPSORT for motorcycle detection and tracking.

Hyperparameters

Each of the three models have a different set of hyperparameters. These hyperparameters include the following: learning rate, batch size, epoch, and image size. Hyperparameters are adjustable parameters that control the behavior of a model during training. While each hyperparameter serves a different function, they all play a crucial role in determining the performance of the model. By carefully selecting and tuning these hyperparameters, it is possible to optimize the model's performance and achieve better results.

- **Learning rate**: The learning rate is a hyperparameter that controls the size of the step the model takes when updating its weights during training. A high learning rate can result in faster convergence but may also cause the model to overshoot the optimal solution. On the other hand, a low learning rate can result in more precise convergence but may also cause the model to get stuck in a suboptimal solution.

- **Batch size**: The batch size is a hyperparameter that determines the number of training examples used in one iteration of model training. A larger batch size can result in faster training but may also require more memory and may result in less stable convergence. A smaller batch size can result in slower training but may also provide more stable convergence and better generalization.

- **Epoch**: An epoch is a hyperparameter that represents one complete pass through the entire training dataset. The number of epochs determines how many times the model will be exposed to the entire training dataset during training. Increasing the number of epochs can result in better model performance but may also increase the risk of overfitting.

- **Image size**: The image size is a hyperparameter that determines the dimensions of the input images used for training the model. A larger image size can result in more detailed feature extraction and better model performance but may also require more memory and computational resources. A smaller image size can result in faster training but may also result in less detailed feature extraction and reduced model performance.

Import YOLOv7 in Google Colab

Once the dataset has been annotated, augmented and finalized, the next step is to clone the YOLOv7 github repository to the Google Colab file of the project. From the script, the command !git clone https://github.com/WongKinYiu/yolov7 would download the Github repository of YOLOv7 to the Google Colab storage.

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

Figure 1.1: Import YOLOv7 in Google Colab

Importing the Roboflow Dataset

Once the YOLOv7 github repository has been cloned in the Google Colab, the next step is to import the custom Roboflow dataset into the script. Here, unique API keys allows the script to access the manually annotated and augmented Roboflow dataset formatted for the YOLOv7 model.

```
[ ]  # REPLACE with your custom code snippet generated above

     !pip install roboflow

     from roboflow import Roboflow
     rf = Roboflow(api_key="09hsUdev5ZIM3g3Na6ca")
     project = rf.workspace("new-dataset-67").project("new-dataset-6-7")
     dataset = project.version(1).download("yolov7")

     Collecting requests-toolbelt
       Downloading requests_toolbelt-0.10.1-py2.py3-none-any.whl (54 kB)
          ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 54.5/54.5 kB 8.3 MB/s eta 0:00:00
     Requirement already satisfied: tqdm>=4.41.0 in /usr/local/lib/python3.10/dist-packages (from roboflow) (4.65.0)
     Collecting python-dotenv
       Downloading python_dotenv-1.0.0-py3-none-any.whl (19 kB)
     Requirement already satisfied: python-dateutil in /usr/local/lib/python3.10/dist-packages (from roboflow) (2.8.2)
     Requirement already satisfied: PyYAML>=5.3.1 in /usr/local/lib/python3.10/dist-packages (from roboflow) (6.0)
     Collecting idna==2.10
```

Figure 1.2: Import the Roboflow Dataset

DeepSORT

DeepSORT was also implemented in the YOLOv7 model in order to apply object tracking along with the detection. For the video outputs of the project, the DeepSORT tracker using YOLOv7 weights was used in order to visualize the tracking in bounding boxes. DeepSORT is an algorithm used for Multiple Object Tracking (MOT) which is an important technology in the field of computer vision. YOLOv7-DeepSORT is a proposal that applies YOLOv7 as the object detection part to the DeepSORT. In order to

apply DeepSORT for the project, the script included a !git clone line that would clone the DeepSORT repository into the project.

```
[ ]  !git clone https://github.com/deshwalmahesh/yolov7-deepsort-tracking
     %cd yolov7-deepsort-tracking
     !rm tracking_helpers.py
```

Figure 1.3: Import DeepSORT

Merging and Augmenting Dataset Annotations

Prior to implementing the new hyperparameters and start the model training process, it was necessary to enhance the quality of the dataset. To accomplish this objective, a compiled dataset from the CCTV 6-7 PM time scenario was created. In order to ensure consistency and accuracy in the dataset, a standardized annotation schema was established. Our group determined that the Rider class should be composed of the entire motorcycle, while the helmet classes should solely include the head area of the rider. This step was crucial for achieving consistent annotations and better results.

Figure 2.1: Annotation schema

After finalizing the annotation schema, the groups were able to combine their datasets into one main dataset with at least 1,000 images for each class. There were five classes: Rider, Full-Face, Half-Face, Invalid, and No-Helmet. However, some groups used different names for their helmet classes, so it was necessary to standardize the class names before performing data augmentation.

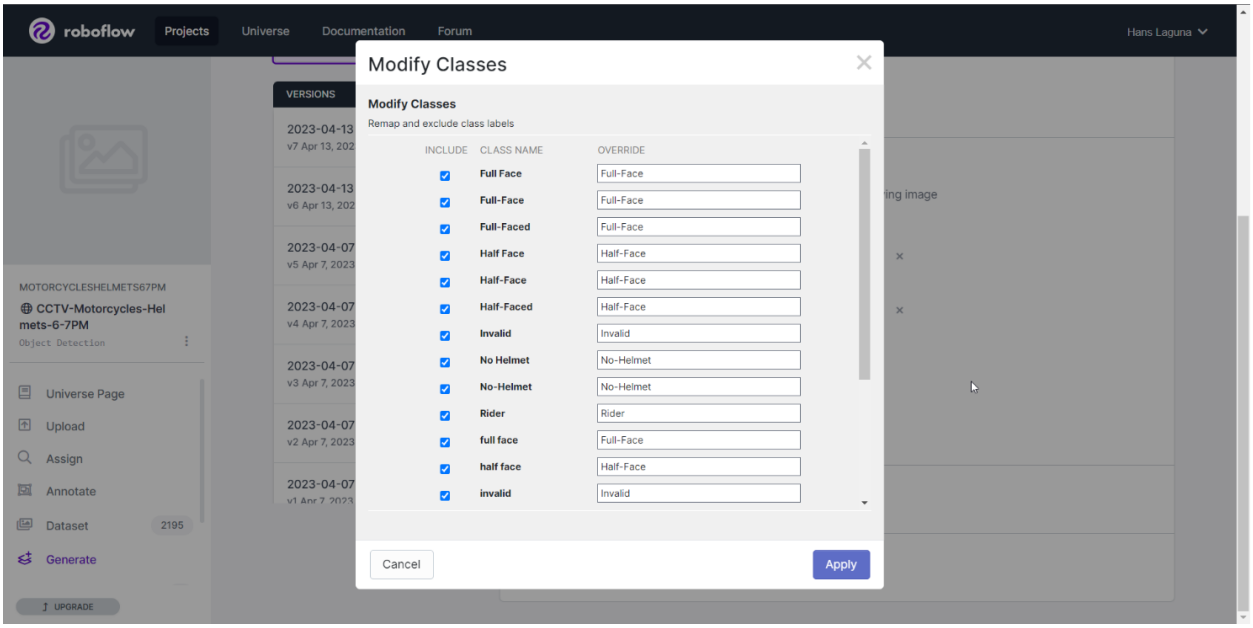


Figure 2.2: Class modifications

After standardizing the class names, the data augmentation process was proceeded. To artificially increase the size of the dataset, several augmentation techniques were employed. These were the following included data augmentation techniques:

- Flip: Creates a mirror image of the input image either horizontally or vertically; 90°.
- Rotate: rotates the input image by 90 degrees in either direction.
- Rotation, which randomly rotates the input image between -15° and +15°.
- Saturation, which randomly adjusts the saturation of the input image between -10% and +10%.
- Bounding Box Flip, which mirrors the bounding box coordinates.
- Bounding Box 90° Rotate, which rotates the bounding box coordinates by 90 degrees.
- Bounding Box Brightness, which randomly adjusts the brightness of the bounding box by up to +15%.

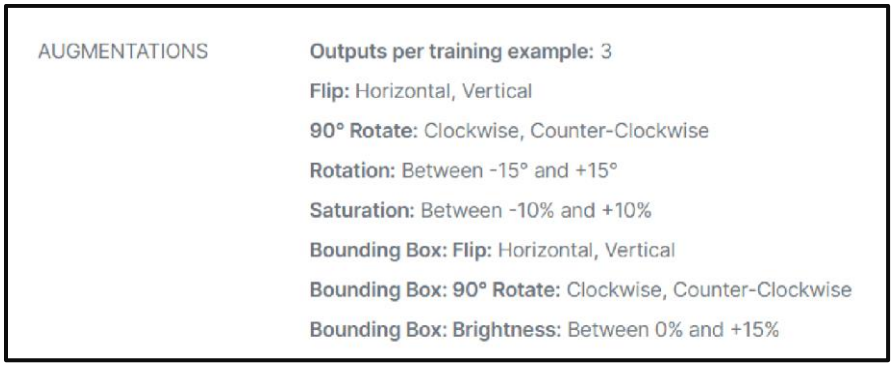


Figure 2.3: Dataset Augmentations

The final dataset after the data augmentation process reached a final count of 1000 images per class. This included the Rider, Full-Face, Half-Face, Invalid, and No-Helmet classes. Here are clear images of Clear images of different classes/labels based on time (Class image only):

Figure 2.4: Half face and Full Face Annotations

Figure 2.5: Invalid and No Helmet Annotations

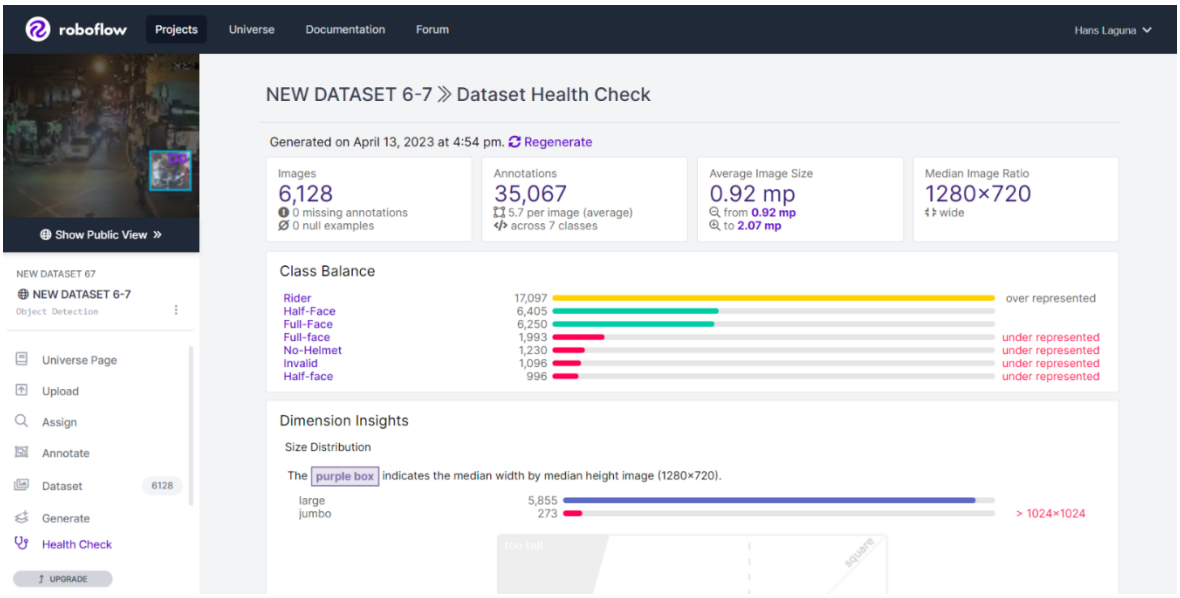


Figure 2.6: 1K Images Per Class

658 Muralla St., Intramuros, Manila, 1002, Philippines
Tel. No. : +63(2) 247-5000 local 5804
Telefax : +63(2) 891-0800
Web : www.mapua.edu.ph

Applying Hyperparameters

As for applying the hyperparameters, three files were created to adjust the learning rate for each hyperparameter. These files include: hyp.scratch.p5-h1.yaml, hyp.scratch.p5-h2.yaml, and hyp.scratch.p5-h3.yaml. The learning rate for hyperparameter 1 was set to 0.01.

E: > Library > Downloads > ! hyp.scratch.p5-h1.yaml

```yaml
lr0: 0.01  # 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
warmup_bias_lr: 0.1  # warmup initial bias lr
box: 0.05  # box loss gain
cls: 0.3  # cls loss gain
cls_pw: 1.0  # cls BCELoss positive_weight
obj: 0.7  # obj loss gain (scale with pixels)
obj_pw: 1.0  # obj BCELoss positive_weight
iou_t: 0.20  # IoU training threshold
anchor_t: 4.0  # anchor-multiple threshold
# anchors: 3  # anchors per output layer (0 to ignore)
fl_gamma: 0.0  # focal loss gamma (efficientDet default gamma=1.5)
hsv_h: 0.015  # image HSV-Hue augmentation (fraction)
hsv_s: 0.7  # image HSV-Saturation augmentation (fraction)
hsv_v: 0.4  # image HSV-Value augmentation (fraction)
degrees: 0.0  # image rotation (+/- deg)
translate: 0.2  # image translation (+/- fraction)
scale: 0.9  # image scale (+/- gain)
shear: 0.0  # image shear (+/- deg)
perspective: 0.0  # image perspective (+/- fraction), range 0-0.001
flipud: 0.0  # image flip up-down (probability)
fliplr: 0.5  # image flip left-right (probability)
mosaic: 1.0  # image mosaic (probability)
mixup: 0.15  # image mixup (probability)
copy_paste: 0.0  # image copy paste (probability)
paste_in: 0.15  # image copy paste (probability), use 0 for faster training
loss_ota: 1 # use ComputeLossOTA, use 0 for faster training
```

Figure 3.1: hyp.scratch.p5-h1.yaml

As for hyperparameter 2, the learning rate for hyp.scratch.p5-h1.yaml was changed to 0.01

E: > Library > Downloads > ! hyp.scratch.p5-h2.yaml

```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
warmup_bias_lr: 0.1  # warmup initial bias lr
box: 0.05  # box loss gain
cls: 0.3  # cls loss gain
cls_pw: 1.0  # cls BCELoss positive_weight
obj: 0.7  # obj loss gain (scale with pixels)
obj_pw: 1.0  # obj BCELoss positive_weight
iou_t: 0.20  # IoU training threshold
anchor_t: 4.0  # anchor-multiple threshold
# anchors: 3  # anchors per output layer (0 to ignore)
fl_gamma: 0.0  # focal loss gamma (efficientDet default gamma=1.5)
hsv_h: 0.015  # image HSV-Hue augmentation (fraction)
hsv_s: 0.7  # image HSV-Saturation augmentation (fraction)
hsv_v: 0.4  # image HSV-Value augmentation (fraction)
degrees: 0.0  # image rotation (+/- deg)
translate: 0.2  # image translation (+/- fraction)
scale: 0.9  # image scale (+/- gain)
shear: 0.0  # image shear (+/- deg)
perspective: 0.0  # image perspective (+/- fraction), range 0-0.001
flipud: 0.0  # image flip up-down (probability)
fliplr: 0.5  # image flip left-right (probability)
mosaic: 1.0  # image mosaic (probability)
mixup: 0.15  # image mixup (probability)
copy_paste: 0.0  # image copy paste (probability)
paste_in: 0.15  # image copy paste (probability), use 0 for faster training
loss_ota: 1 # use ComputeLossOTA, use 0 for faster training

```

Figure 3.2: hyp.scratch.p5-h2.yaml

658 Muralla St., Intramuros, Manila, 1002, Philippines
Tel. No. : +63(2) 247-5000 local 5804
Telefax : +63(2) 891-0800
Web : www.mapua.edu.ph

Lastly, the learning rate for hyperparameter 3 was changed from the file: hyp.scratch.p5-h3.yaml. The learning rate was changed to 0.03.

E: > Library > Downloads > ! hyp.scratch.p5-h3.yaml

```yaml
lr0: 0.03  # 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
warmup_bias_lr: 0.1  # warmup initial bias lr
box: 0.05  # box loss gain
cls: 0.3  # cls loss gain
cls_pw: 1.0  # cls BCELoss positive_weight
obj: 0.7  # obj loss gain (scale with pixels)
obj_pw: 1.0  # obj BCELoss positive_weight
iou_t: 0.20  # IoU training threshold
anchor_t: 4.0  # anchor-multiple threshold
# anchors: 3  # anchors per output layer (0 to ignore)
fl_gamma: 0.0  # focal loss gamma (efficientDet default gamma=1.5)
hsv_h: 0.015  # image HSV-Hue augmentation (fraction)
hsv_s: 0.7  # image HSV-Saturation augmentation (fraction)
hsv_v: 0.4  # image HSV-Value augmentation (fraction)
degrees: 0.0  # image rotation (+/- deg)
translate: 0.2  # image translation (+/- fraction)
scale: 0.9  # image scale (+/- gain)
shear: 0.0  # image shear (+/- deg)
perspective: 0.0  # image perspective (+/- fraction), range 0-0.001
flipud: 0.0  # image flip up-down (probability)
fliplr: 0.5  # image flip left-right (probability)
mosaic: 1.0  # image mosaic (probability)
mixup: 0.15  # image mixup (probability)
copy_paste: 0.0  # image copy paste (probability)
paste_in: 0.15  # image copy paste (probability), use 0 for faster training
loss_ota: 1 # use ComputeLossOTA, use 0 for faster training
```

Figure 3.3: hyp.scratch.p5-h3.yaml

Once the learning rate was adjusted, the yaml files were uploaded to the data folder inside YOLOv7 using !down.

```
[ ] %cd /content/yolov7/data/
    !gdown --id "1_0OPUndGthg8VfZqFZW5vKZQR1SXD6Qz&confirm=t"
    !gdown --id "169CEuWXoIbuu0-0fSkU0kOHf85W34dPy&confirm=t"
    !gdown --id "180G9fpOwZCERW4r_yTnyxK_PIq61jVKH&confirm=t"

    /content/yolov7/data
    /usr/local/lib/python3.10/dist-packages/gdown/cli.py:121: FutureWarning: Option `--id` was deprecated in version 4.3.1 and will be removed in 5.0. You
      warnings.warn(
    Downloading...
    From: https://drive.google.com/uc?id=1_0OPUndGthg8VfZqFZW5vKZQR1SXD6Qz&confirm=t
    To: /content/yolov7/data/hyp.scratch.p5-h1.yaml
    100% 1.51k/1.51k [00:00<00:00, 11.8MB/s]
    /usr/local/lib/python3.10/dist-packages/gdown/cli.py:121: FutureWarning: Option `--id` was deprecated in version 4.3.1 and will be removed in 5.0. You
      warnings.warn(
    Downloading...
    From: https://drive.google.com/uc?id=169CEuWXoIbuu0-0fSkU0kOHf85W34dPy&confirm=t
    To: /content/yolov7/data/hyp.scratch.p5-h2.yaml
    100% 1.51k/1.51k [00:00<00:00, 11.3MB/s]
    /usr/local/lib/python3.10/dist-packages/gdown/cli.py:121: FutureWarning: Option `--id` was deprecated in version 4.3.1 and will be removed in 5.0. You
      warnings.warn(
    Downloading...
    From: https://drive.google.com/uc?id=180G9fpOwZCERW4r_yTnyxK_PIq61jVKH&confirm=t
    To: /content/yolov7/data/hyp.scratch.p5-h3.yaml
```

Figure 3.4: Upload yaml files to Colab

After the yaml files were uploaded, the hyp flag and other flags like img, batch, and epochs were used to set the hyperparameters in the YOLOv5 model. The code for setting hyperparameters 1 to 3 is shown in the following figures.

```
[ ] # run this cell to begin training
    %cd /content/yolov7
    !python train.py --hyp data/hyp.scratch.p5-h1.yaml --batch 64 --epochs 50 --data {dataset.location}/data.yaml --weights 'yolov7_training.pt' --device
```

Figure 3.5: Hyperparameter 1 settings

```
# run this cell to begin training
%cd /content/yolov7
!python train.py --hyp data/hyp.scratch.p5-h2.yaml --batch 32 --epochs 75 --data {dataset.location}/data.yaml --weights 'yolov7_training.pt' --device
```

Figure 3.6: Hyperparameter 2 settings

```
# run this cell to begin training
%cd /content/yolov7
!python train.py --hyp data/hyp.scratch.p5-h3.yaml --batch 16 --epochs 100 --data {dataset.location}/data.yaml --weights 'yolov7_training.pt' --devic
```

Figure 3.7: Hyperparameter 3 settings

Hyperparameter 1

As required for this activity, we will set the first hyperparameter settings of the YOLOv7 model training to the following specifications:

- The learning rate will be set to 0.01
- The batch size will be set to 64
- The number of epochs will be set to 50
- The image size will be set to 640x640

The screenshot below shows that the hyperparameter 1 settings are applied in the YOLOv7 training.

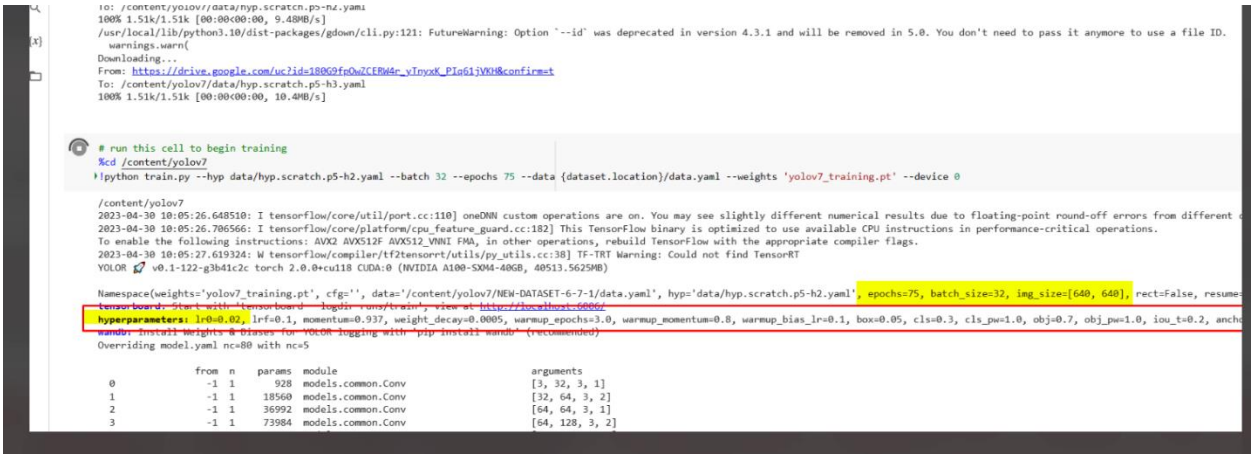


Figure 4.1: Hyperparameter 1 settings being ran

Training

On the hyperparameter 1 model training, we employed the hyperparameter settings of 0.01 on the Learning Rate or LR. The learning rate is often not employed directly while training a learned model. Rather, the learning rate is employed during the training phase to choose the optimal model parameters. When the model has been trained, the learning rate is commonly fixed, or the learning rate schedule is employed to gradually decrease the learning rate over time to fine-tune the performance of the model.

In terms of batch size, we employed 64 batches for model training. Batches are rarely employed directly during model training. Rather, the entire train dataset is typically evaluated all at once, and the model's performance is determined using metrics like accuracy, precision, recall, or F1 score. Certain frameworks or libraries, on the other hand, may use batches during inference to speed up the processing of large datasets.

658 Muralla St., Intramuros, Manila, 1002, Philippines
Tel. No. : +63(2) 247-5000 local 5804
Telefax : +63(2) 891-0800
Web : www.mapua.edu.ph

Moving on to Epochs, we utilized 50 epochs for the hyperparameter 1 settings. The number of epochs utilized during training may have an indirect influence on the model's performance throughout training since it influences the quality of the learnt parameters and the model's capacity for generalization to new data.

Finally, for model training, we selected an image size of 640x640. During model training, the picture size is often set to a fixed number specified by the application's needs or the hardware's limits. To guarantee that the model gets evaluated on the same sort of data that it was trained on, the size of the images in the train dataset should match the size of the images employed during training.

And, as can be determined by the training results, there are six classes: Full-Face, Half-Face, Invalid, No-Helmet, and Rider. As you can see, all of the classes used 1203 images, and the "all" class has the most labels with 7067, followed by the "Rider" class with 3429, the "Full Faced" class with 1681 labels, the "Half Faced" class with 1493 labels, the "No Helmet" class with 234 labels, and finally the "Invalid" class with 230 labels.

Continuing toward precision, the "Rider" class has the highest precision rate, with a precision rate of 0.642, followed by the "Half Faced" class, which has a precision rate of 0.548, the "Invalid" class, which has a precision rate of 0.507, the "all" class, which has a precision rate of 0.488, the "Full Face" class, which has an accuracy rate of 0.413, and the "No Helmet" which has an accuracy of 0.261.

In terms of recall, the "Rider" class has the highest recall rate, with a rate of 0.91, followed by the "Full Faced" class, which has a rate of 0.641, then the "all" class, which has a rate of 0.496, then the "Half Faced" class, which has a rate of 0.412, then the "Invalid" class, which has a rate of 0.261, and finally the "No Helmet" which has an accuracy of 0.329.

The mean Average Precision at IoU=0.5 or mAP@.5 is also shown in the training result the “Half Face” class has the highest Average Precision with an average precision of 0.44 and the lowest mean Average Precision from hyperparameter 1 is the “No Helmet” class with an average precision of 0.176.

658 Muralla St., Intramuros, Manila, 1002, Philippines
Tel. No. : +63(2) 247-5000 local 5804
Telefax : +63(2) 891-0800
Web : www.mapua.edu.ph

```
[5]  # run this cell to begin training
     %cd /content/yolov7
     !python train.py --hyp data/hyp.scratch.p5-h1.yaml --batch 64 --epochs 50 --data {dataset.location}/data.yaml --weights 'yolov7_training.pt' --device 0

Epoch   gpu_mem       box       obj       cls     total    labels  img_size
14/49     40.1G   0.04055   0.01406  0.004536   0.05915       661       640: 100% 66/66 [01:08<00:00,  1.03s/it]
               Class      Images      Labels           P           R      mAP@.5  mAP@.5:.95: 100% 10/10 [00:08<00:00,  1.17it/s]
                 all        1203        7067       0.451       0.384       0.266       0.117

Epoch   gpu_mem       box       obj       cls     total    labels  img_size
15/49     40.1G    0.0402   0.01405  0.004457   0.05871      1165       640: 100% 66/66 [01:09<00:00,  1.05s/it]
               Class      Images      Labels           P           R      mAP@.5  mAP@.5:.95: 100% 10/10 [00:08<00:00,  1.17it/s]
                 all        1203        7067       0.433       0.375       0.233       0.102

Epoch   gpu_mem       box       obj       cls     total    labels  img_size
16/49     40.1G   0.04014   0.01413  0.004386   0.05866       912       640: 100% 66/66 [01:08<00:00,  1.04s/it]
               Class      Images      Labels           P           R      mAP@.5  mAP@.5:.95: 100% 10/10 [00:08<00:00,  1.17it/s]
                 all        1203        7067       0.288        0.36       0.261       0.115

Epoch   gpu_mem       box       obj       cls     total    labels  img_size
17/49     40.1G   0.03968   0.01404   0.00434   0.05806       682       640: 100% 66/66 [01:08<00:00,  1.04s/it]
               Class      Images      Labels           P           R      mAP@.5  mAP@.5:.95: 100% 10/10 [00:08<00:00,  1.18it/s]
                 all        1203        7067       0.506       0.399       0.287       0.129

Epoch   gpu_mem       box       obj       cls     total    labels  img_size
18/49     40.1G   0.03928   0.01394   0.00428    0.0575       668       640: 100% 66/66 [01:08<00:00,  1.04s/it]
               Class      Images      Labels           P           R      mAP@.5  mAP@.5:.95: 100% 10/10 [00:08<00:00,  1.18it/s]
                 all        1203        7067       0.281       0.366        0.27       0.119

Epoch   gpu_mem       box       obj       cls     total    labels  img_size
19/49     40.1G   0.03864   0.01373  0.004316   0.05668       698       640: 100% 66/66 [01:08<00:00,  1.03s/it]
               Class      Images      Labels           P           R      mAP@.5  mAP@.5:.95: 100% 10/10 [00:08<00:00,  1.17it/s]
                 all        1203        7067       0.305       0.444       0.292       0.129

Epoch   gpu_mem       box       obj       cls     total    labels  img_size
20/49     40.1G    0.0386   0.01366  0.004157   0.05642       662       640: 100% 66/66 [01:08<00:00,  1.04s/it]
               Class      Images      Labels           P           R      mAP@.5  mAP@.5:.95: 100% 10/10 [00:08<00:00,  1.17it/s]
                 all        1203        7067       0.313       0.425       0.306       0.138
```

Figure 4.2: Training Hyperparameter 1 (beginning)

```
[5]
44/49     40.1G   0.03217   0.01226  0.003113   0.04755       667       640: 100% 66/66 [01:09<00:00,  1.05s/it]
               Class      Images      Labels           P           R      mAP@.5  mAP@.5:.95: 100% 10/10 [00:08<00:00,  1.21it/s]
                 all        1203        7067       0.495       0.471       0.433       0.199

Epoch   gpu_mem       box       obj       cls     total    labels  img_size
45/49     40.1G   0.03215   0.01244  0.003063   0.04766       801       640: 100% 66/66 [01:09<00:00,  1.05s/it]
               Class      Images      Labels           P           R      mAP@.5  mAP@.5:.95: 100% 10/10 [00:08<00:00,  1.20it/s]
                 all        1203        7067       0.448       0.526       0.443       0.205

Epoch   gpu_mem       box       obj       cls     total    labels  img_size
46/49     40.1G   0.03187   0.01235  0.002998   0.04722       793       640: 100% 66/66 [01:09<00:00,  1.05s/it]
               Class      Images      Labels           P           R      mAP@.5  mAP@.5:.95: 100% 10/10 [00:08<00:00,  1.20it/s]
                 all        1203        7067       0.508       0.474        0.44       0.204

Epoch   gpu_mem       box       obj       cls     total    labels  img_size
47/49     40.1G   0.03174   0.01211  0.003015   0.04686       748       640: 100% 66/66 [01:08<00:00,  1.05s/it]
               Class      Images      Labels           P           R      mAP@.5  mAP@.5:.95: 100% 10/10 [00:08<00:00,  1.20it/s]
                 all        1203        7067       0.457       0.535       0.453       0.208

Epoch   gpu_mem       box       obj       cls     total    labels  img_size
48/49     40.1G   0.03161   0.01205  0.002894   0.04655       837       640: 100% 66/66 [01:08<00:00,  1.04s/it]
               Class      Images      Labels           P           R      mAP@.5  mAP@.5:.95: 100% 10/10 [00:08<00:00,  1.21it/s]
                 all        1203        7067       0.468       0.526       0.455       0.212

Epoch   gpu_mem       box       obj       cls     total    labels  img_size
49/49     40.1G   0.03158   0.01206  0.002901   0.04654       766       640: 100% 66/66 [01:08<00:00,  1.04s/it]
               Class      Images      Labels           P           R      mAP@.5  mAP@.5:.95: 100% 10/10 [00:12<00:00,  1.30s/it]
                 all        1203        7067       0.488       0.496       0.448       0.205
           Full-Face        1203        1681       0.413       0.641       0.477       0.169
           Half-Face        1203        1493       0.548       0.412        0.44       0.142
             Invalid        1203         230       0.507       0.257       0.301       0.155
           No-Helmet        1203         234       0.329       0.261       0.176      0.0496
               Rider        1203        3429       0.642        0.91       0.844       0.511
50 epochs completed in 1.113 hours.

Optimizer stripped from runs/train/exp/weights/last.pt, 74.8MB
Optimizer stripped from runs/train/exp/weights/best.pt, 74.8MB
```

Figure 4.3: Training Hyperparameter 1 (ending)

658 Muralla St., Intramuros, Manila, 1002, Philippines
Tel. No. : +63(2) 247-5000 local 5804
Telefax : +63(2) 891-0800
Web : www.mapua.edu.ph

Testing

On the model testing of the hyperparameter 1, we have used the hyperparameter settings of 0.01 setting on the Learning Rate or LR. When testing a trained model, the learning rate is typically not used directly. Instead, the learning rate is used during the training phase to find the optimal model parameters. Once the model is trained, the learning rate is typically set to a fixed value, or the learning rate schedule may be used to gradually decrease the learning rate over time to fine-tune the model's performance.

As for the batch, we have used 64 batches setting for the model testing. During model testing, batches are not typically used directly. Instead, the entire test dataset is usually evaluated at once, and the model's performance is measured using metrics such as accuracy, precision, recall, or F1 score. However, some frameworks or libraries may use batches during inference to speed up the processing of large datasets.

Moving on to the Epochs we have used 50 epochs for the settings used in hyperparameter 1. The number of epochs used during training may have an indirect effect on the performance of the model during testing, as it determines the quality of the learned parameters and the ability of the model to generalize to new data.

And lastly, we have used 640x640 for the image size used for model testing. During model testing, the image size is usually set to a fixed value, which is determined by the requirements of the application or the constraints of the hardware. The size of the images in the test dataset should match the size of the images used during training, to ensure that the model is evaluated on the same type of data it was trained on.

And as we can see on the testing results there are 6 classes the Full-Face, Half-Face, Invalid, No-Helmet the Rider, all. As we can see all the classes used 1203 images and for the Labels “all” class has the highest number of labels which has 7067 followed by the “Rider” class that has 3429 labels, the “Full Faced” class has 1681 labels, the “Half Faced” has 1493 labels, the “No Helmet” has 234 labels, and finally the “Invalid” class which has 230 labels.

Moving on to the precision, the “Rider” class has the highest precision rate, which has a precision rate of 0.62 followed by the “Half Faced” with a precision rate of 0.514, next is the “Invalid” class which has a precision rate of 0.467, then “all” class that has 0.463 precision rate, “Full Faced” class which has an accuracy rate of 0.401 and lastly with the lowest accuracy rate the “No Helmet” class which has an accuracy rate of 0.314.

As for the recall the “Rider” class has the highest recall rate, with a recall rate of 0.912 followed by “Full Faced” class which has a recall rate of 0.667, then the “all” class that has 0.526 recall rate, then the “Half Faced” class which has recall rate of 0.494, then the “Invalid” class with 0.298 recall rate, and lastly with the lowest recall rate the “No Helmet” class which has 0.261 recall rate.

The mean Average Precision at IoU=0.5 or mAP@.5 is also shown in the testing result the “Rider” class has the highest Average Precision with an average precision of 0.848 and the lowest mean Average Precision from hyperparameter 1 is the “No Helmet” class with an average precision of 0.166.

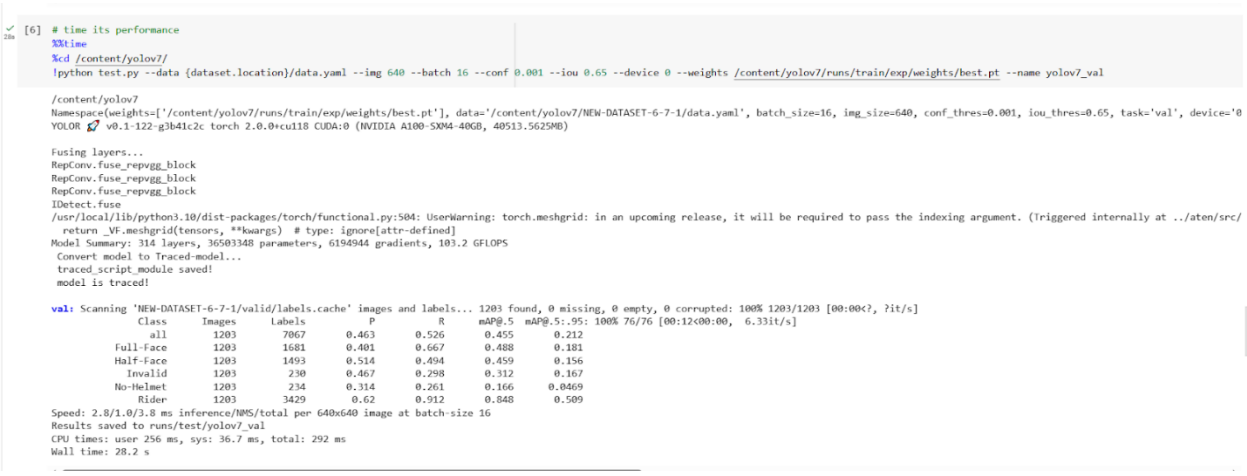


Figure 4.4: Testing Hyperparameter 1

Screenshot and Evaluation of Tensorboard results

F1 Curve

The F1 Confidence Curve shows us a graph with 6 different lines, plotted onto a graph with the F1 score as the y-axis, and the confidence as the x-axis. The different lines represent the 5 classes, as well as one which is the average. The following classes are: Full-face, Half-face, Invalid, No-Helmet, and the Rider class. The legend for which colored line represents which class can be seen in the graph below.

The graph shows us that the model with the first configuration of hyperparameter displays an average score of 0.47 at a 0.298 level for all classes. Aside from the rider class, all the other four classes follow the general trend that the average does, albeit at different scales.

The confidence threshold of 0.298 indicates to us that the model will most likely produce a high recall with low precision. This would mean that the model is more likely to be able to correctly detect more true positives as well as be able to more likely make more false-positive predictions. This threshold that was achieved would be acceptable in certain scenarios wherein the identification of false-positives is less costly than identifying false-negatives. Therefore, scenarios where identifying false-positives would be costlier would therefore be more appropriate of having a higher confidence threshold.

The rider class curve is an outlier, as seen by the graph and the difference in values, mainly because it is overrepresented in the dataset, therefore the model would obviously produce significantly different results as there is more data. The full-face class curve and half-face class curve follow the average curve at lower values, but the half-face class curve deviates as the values increase, while the full-face rider class only deviates considerably at the highest value. The invalid class curve follows a similar shape to that of the average but shifted lower, indicating the same trend at lower values. Finally, the no-helmet class also follows the same trend as the invalid class curve but lowers as the value increases.

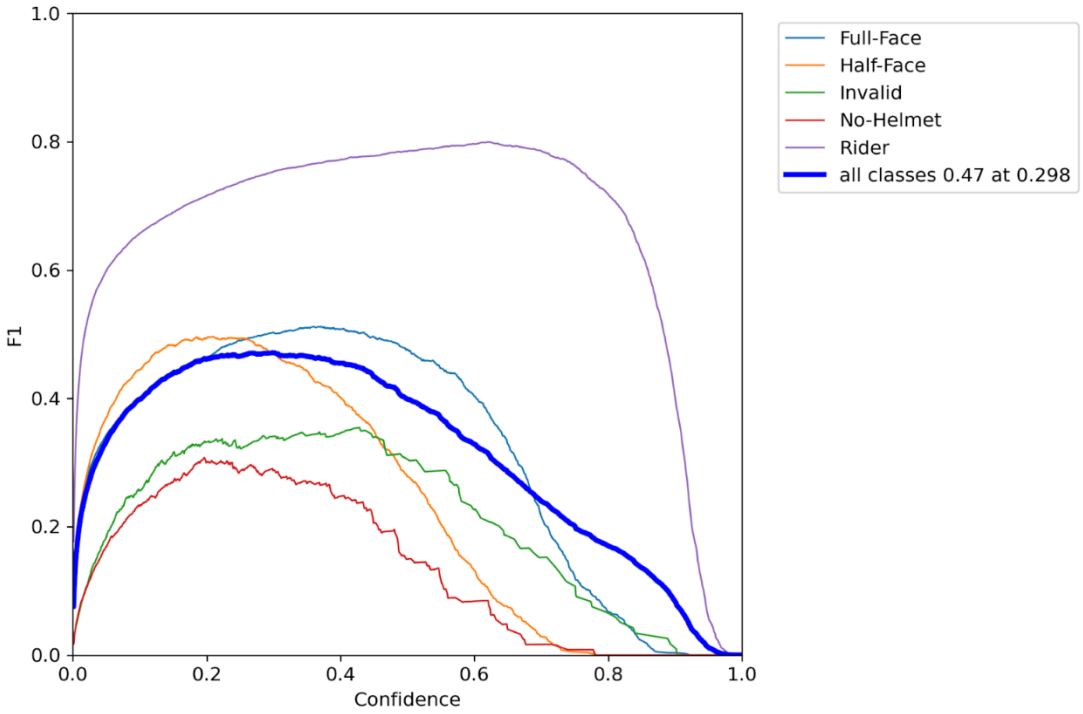


Figure 4.5: Hyperparameter 1 F1 Score

Precision and Recall Curve

The Precision Recall Curve shows us a graph with 6 different lines, plotted onto a graph with the precision as the y-axis, and the recall as the x-axis. The different lines represent the 5 classes, as well as one which is the average, each with a corresponding precision. The following classes are: Full-face, Half-face, Invalid, No-Helmet, and the Rider class. The legend for which colored line represents which class can be seen in the graph below.

The graph of the model with the first configuration of hyperparameter produced the results with the average precision (AP) of 0.448 with a mAP@0.5 for all classes. Only 2 graphs produce a similar curve as to that of the average, mainly being the full-face class curve and the half-face class curve.

The AP of the average class is fairly low at only 0.448, this can be seen by looking at the other classes' curves. The full-face class curve once again as significantly higher values, as observed previously in the F1 confidence curve, and most likely for the same reason as well. The rider class is simply overrepresented in the dataset which means that model has more data to work with, in turn allowing for much higher results. This is the reason why it scored such a high precision at 0.844 compared to the rest of the classes.

The full-face class curve and the half-face class curve, as stated earlier, generally follow the same trend as the average. Both these curves also score fairly similar precision scores at 0.477 and 0.440 respectively. The invalid class curve achieves a higher precision compared at lower recall values, but

precision quickly decreases between the 0.2 and 0.4 recall, in which it continually lowers at a steady rate. The invalid class curve achieves a 0.301 precision score, a bit lower than the average. The no-helmet class curve scores the lowest precision score at 0.176, significantly lower than the second lowest score which was the invalid class.

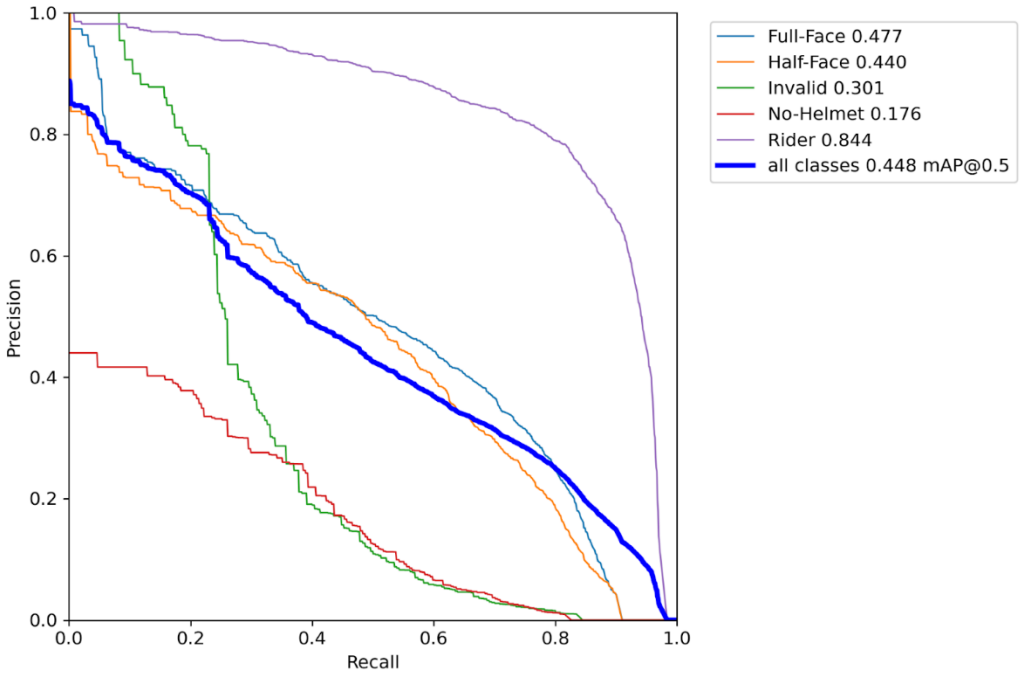


Figure 4.6: Hyperparameter 1 Precision and Recall Score

Precision Curve

The Precision Confidence Curve shows us a graph with 6 different lines, plotted onto a graph with the precision as the y-axis, and the confidence as the x-axis. The different lines represent the 5 classes, as well as one which is the average. The following classes are: Full-face, Half-face, Invalid, No-Helmet, and the Rider class. The legend for which colored line represents which class can be seen in the graph below.

This curve that was produced with the results of the first configuration of hyperparameters shows us a score of 0.968 average for all the classes. This simply indicates that on average, whenever the model achieves a confidence score at 0.943, then the model would be able to perfectly identify the object to the correct class, which is indicated at the 1.00 score. Therefore, the model is highly accurate with the predictions that it makes if the model achieves the indicated confidence score. We can also see how there is a lot of discrepancies at the higher confidence values for most of the graphs.

The most stable graph of the classes is the rider class curve, which for similar reasons stated in the previous graph, produces the most stable graph as the model has trained much more for this class. The invalid class curve scores a lower precision on average at lower confidence values, but precision increases

658 Muralla St., Intramuros, Manila, 1002, Philippines
Tel. No. : +63(2) 247-5000 local 5804
Telefax : +63(2) 891-0800
Web : www.mapua.edu.ph

and surpasses that average as the confidence level increases, eventually achieving the highest value between the 0.6 and 0.8 confidence values.

The no-helmet class curve increases slowly from the 0.0 and 0.4, where it then starts to lower after while achieving random increases. The graph continually lowers after 0.6 confidence and then shoots straight up to the highest precision at 0.8 confidence. This sudden increase in precision also affects the average, and the effects are clearly seen as the average also increases at that confidence score. The full-face class and half-face class curves are once again fairly average, although they do show drops in precision as well at around 0.9 and 0.7 confidence respectively.

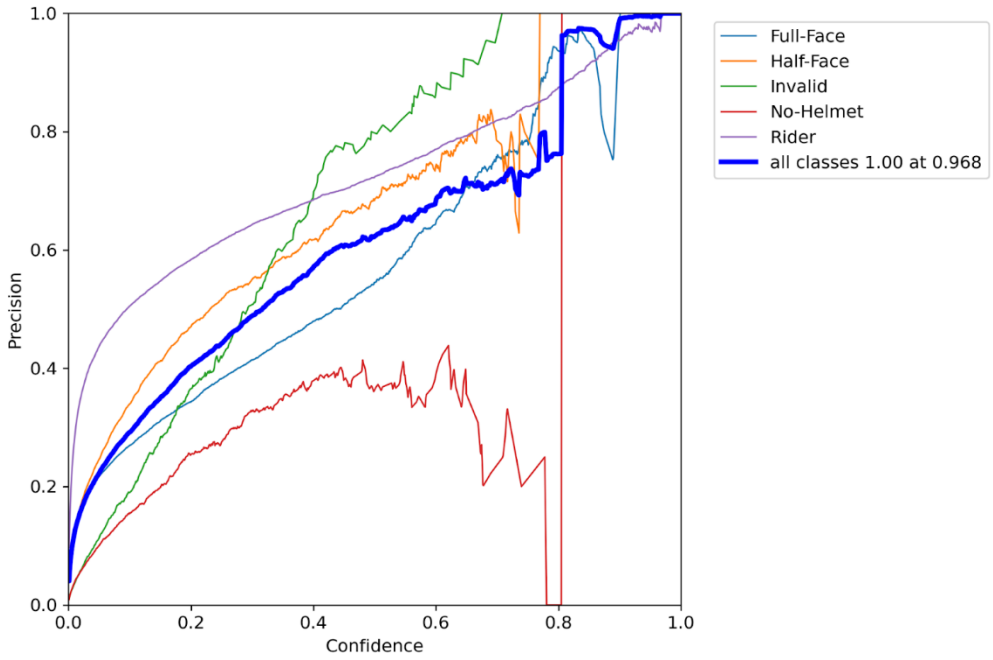


Figure 4.7: Hyperparameter 1 Precision Score

Recall Curve

The Recall Confidence Curve shows us a graph with 6 different lines, plotted onto a graph with the recall as the y-axis, and the confidence as the x-axis. The different lines represent the 5 classes, as well as one which is the average. The following classes are: Full-face, Half-face, Invalid, No-Helmet, and the Rider class. The legend for which colored line represents which class can be seen in the graph below.

This graph shows us that the model with the first configuration of hyperparameters produced a recall score of 0.88 at a confidence threshold of 0.000. This simply means that the model properly detects a large proportion of positive instances correctly as indicated by the high recall score it achieved. This would also mean that the model is more likely to generate false-positive predictions.

658 Muralla St., Intramuros, Manila, 1002, Philippines
Tel. No. : +63(2) 247-5000 local 5804
Telefax : +63(2) 891-0800
Web : www.mapua.edu.ph

The average curve shows us a quick steep decrease at the beginning, which then stabilizes into a more or less straight decrease as it approaches the highest confidence, with a little discrepancy at the end. The rider class curve once again scores the highest, with a very gentle decrease, after which it very quickly falls at around 0.8 confidence. This rapid decrease affects the average graph as seen in the small discrepancy near the end of the average curve. The full-face class curve scores a higher recall than average, but quickly falls below average at 0.6 confidence. The half-face also initially scores higher, but falls below average before it even reaches the 0.2 confidence score. Both the invalid class and no-helmet class curve follow the same trend as having an initial steep decline into a stable decrease, but of course both curves' scores are already much lower than average.

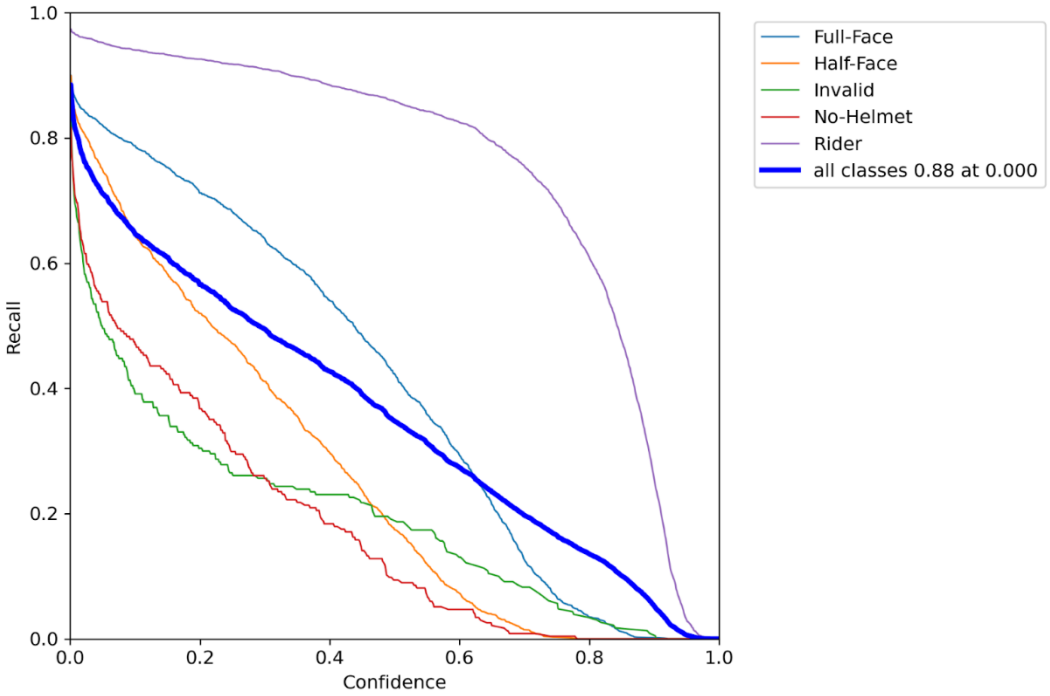


Figure 4.8: Hyperparameter 1 Recall Score

658 Muralla St., Intramuros, Manila, 1002, Philippines
Tel. No. : +63(2) 247-5000 local 5804
Telefax : +63(2) 891-0800
Web : www.mapua.edu.ph

mAP@0.5 and mAP@0.5:0.95 scores

The mAP@0.5 and mAP@0.5:0.95 scores were derived after training the YOLOv7 model. The mAP@0.5 score is the mean average precision calculated at an Intersection over Union (IoU) threshold of 0.5. On the other hand, the mAP@0.5:0.95 score is the mean average mAP over different IoU thresholds from 0.5 – 0.95. Based on the results for the mAP scores, the mAP@0.5 reached a score of 0.4476 after 49 epochs. On the other hand, the mAP@0.5:0.95 reached a score of 0.2052. Generally, a "good" mAP@0.5 score is greater than 0.5 and since the model could only reach a score of 0.4476, it could only detect objects with moderate accuracy. However, the mAP_0.5:0.95 value is only 0.2052, which indicates that the model is struggling to accurately identify objects with high confidence.

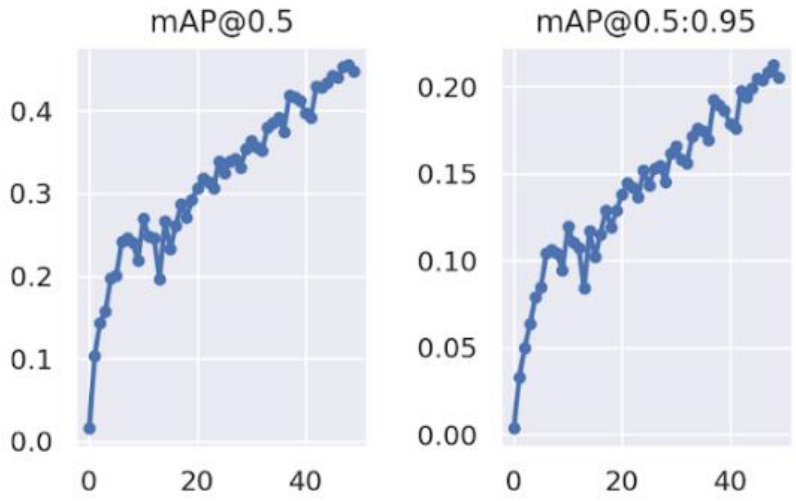


Figure 4.9: Hyperparameter 1 mAP@0.5 and mAP@0.5:0.95 scores

Batch 0 Label

In this section, we discuss the Batch 0 labels on Hyperparameter 1. The labels are used to train the mode to recognize patterns and conduct classification on CCTV footage. There are four (4) categories of Batch 0 labels; they are the following: "Rider," Full-face," Half-face," and" No-helmet." The category "Rider" refers to a person maneuvering a two-wheeled vehicle. Additionally, there are three (3) categories associated with the" Rider" category; these are the" Full-face," Half-face," and" No-helmet" classes. Wherein" Full-face" refers to a person wearing a full-faced helmet," Half-face" refers to a person wearing a half-faced helmet, and lastly," No-helmet" refers to a person wearing neither or no helmet at all. These labels are the foundation for classification in this project. Accurately identifying these objects improves safety measures, helping prevent accidents on transportation.

Figure 4.10: Hyperparameter 1 Batch 0 Label

Batch 0 Predicted

In this section, we will discuss details about the Batch 0 predicted. The predicted results of each category (Rider," Full-face," Half-face," and" No-helmet") provide the group information regarding the performance of the model when identifying the labeled objects. This information consists of each object's confidence level prediction and the detected object's size and aspect. The classification of each object is visualized by surrounding the predicted object using a bounding box and displaying the object's name.

Additionally, the visualizations are overlayed on top of the CCTV footage, thus showing us the location of the detected image and the type of object. The results are also used to calculate performance metrics such as precision, recall, mean average precision (MAE), and root squared mean average precision (RMSAE). The performance metrics are used to determine whether the model needs to be improved or provide insights on additional improvements.

Figure 4.11: Hyperparameter 1 Batch 0 Predicted

658 Muralla St., Intramuros, Manila, 1002, Philippines
Tel. No. : +63(2) 247-5000 local 5804
Telefax : +63(2) 891-0800
Web : www.mapua.edu.ph

MSE

The model includes additional evaluations such as Mean Squared Error (MSE) and Root Mean Squared Error (RMSE). A lower MSE value indicates a more accurate model. In this case, the model achieved an MSE of 0.04677 on the 50th iteration.

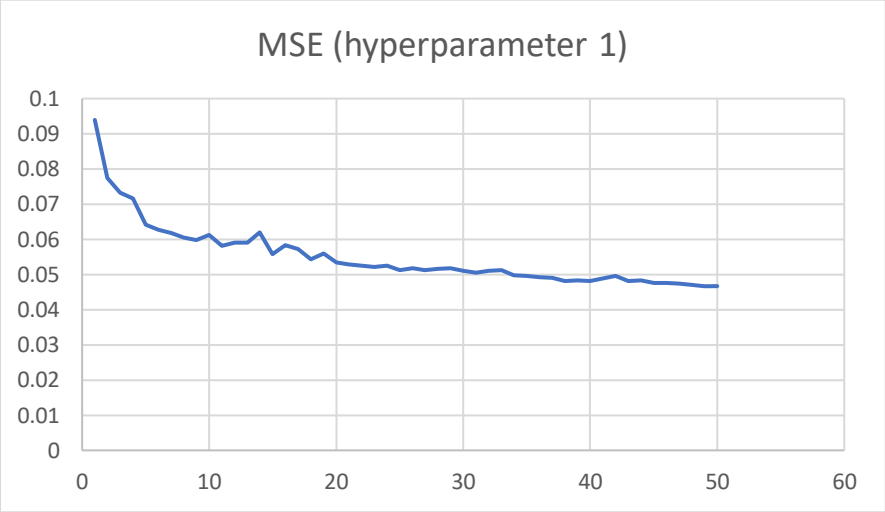


Figure 4.12: Hyperparameter 1 MSE

RMSE

Furthermore, an RMSE (Root Mean Squared Error) of 0.2162 means that, on average, the difference between the predicted values and the actual values is 0.2162 units. Since there is no correct value for MSE, the lower the value the better. This means that the MSE and RMSE dictates that the model was relatively accurate since their scores are close to 0.

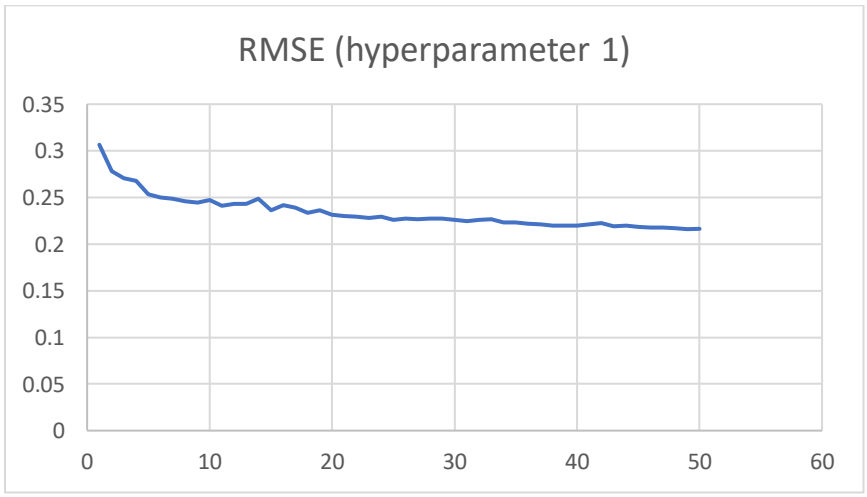


Figure 4.13: Hyperparameter 1 RMSE

Confusion Matrix

From the confusion matrix obtained from applying the first set of hyperparameter settings, the highest true positive value obtained is for the Rider class which has a 0.93 value. The second and third highest true positive value obtained are for the Full-Face and Half-Face classes which have a value of 0.56 and 0.54 respectively. The fourth and fifth highest true positive value obtained are for the No-Helmet and Invalid classes which have a value of 0.31 and 0.28 respectively. From the predicted background FN, it can be observed that there is a significant number of Invalid and No-Helmet (0.60 and 0.59) annotated which have not been detected as part of the class. Conversely, from the predicted background FP, there is a significant number of Full-Face and Rider (0.33 and 0.47) instances which have been detected from the image background which have not been originally annotated. The Full-Face, Half-Face, and Invalid classes have increased false negative values compared to the classes of No-Helmet and Rider, which could be attributed to the similarities in appearance of the three classes. As observed in the Full-Face and Half-Face classes, the Full-Face class has an increased false positive value for the Half-Face class (0.21), while the Half-Face class also has an increased false positive value for the Full-Face class (0.11).

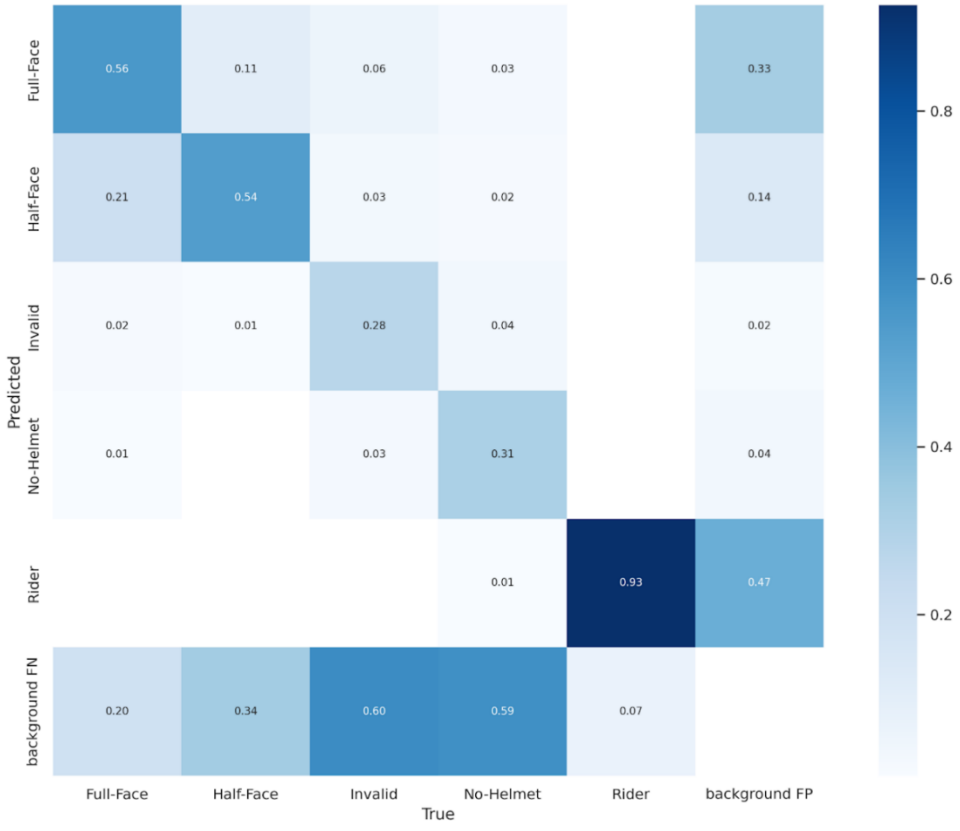


Figure 4.14: Hyperparameter 1 Confusion Matrix

658 Muralla St., Intramuros, Manila, 1002, Philippines
Tel. No. : +63(2) 247-5000 local 5804
Telefax : +63(2) 891-0800
Web : www.mapua.edu.ph

Model Output

The video output describes the testing of a model using Hyperparameter 1 with LR set to 0.01, 64 batches, 50 epochs, and 640x640 image size. The output features 6 classes: Full-Face, Half-Face, Invalid, No-Helmet, Rider, and all. The "all" class had the highest number of labels at 7067, followed by the "Rider" class with 3429 labels, while the "Invalid" class had the least at 230 labels. The precision rate was highest for the "Rider" class at 0.62, followed by the "Half Faced" at 0.514, and the "Invalid" class at 0.467.

Video Output (DeepSORT)

By training the YOLOv7 model with the hyperparameter 1 settings, the group was able to generate a video output of the DeepSORT tracker using the YOLOv7 weights. A frame from the resulting video is shown in the screenshot below

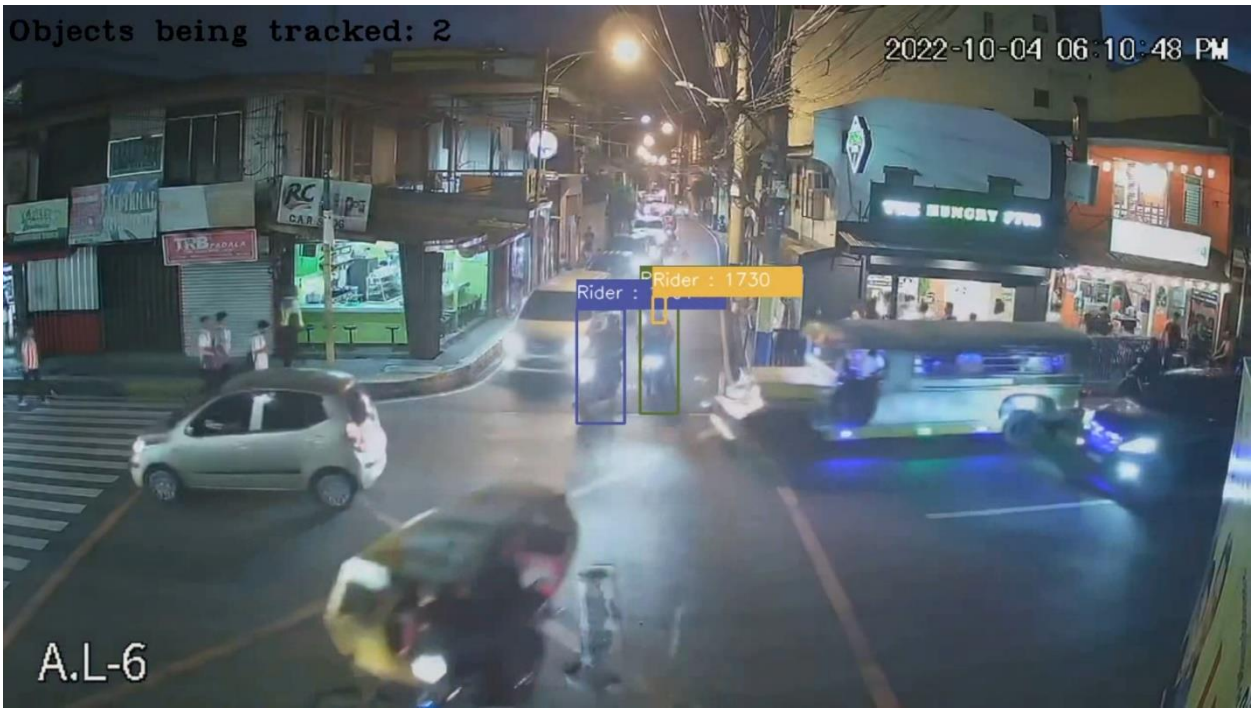


Figure 4.15: Hyperparameter 1 Video output DeepSORT

Based on this frame, we see that the DeepSORT algorithm is working because it was able to detect the objects in the videos and assign them a unique ID for object tracking.

658 Muralla St., Intramuros, Manila, 1002, Philippines
Tel. No. : +63(2) 247-5000 local 5804
Telefax : +63(2) 891-0800
Web : www.mapua.edu.ph

Hyperparameter 2

As required for this activity, we will set the second hyperparameter settings of the YOLOv7 model training to the following specifications:

- The learning rate will be set to 0.02
- The batch size will be set to 32
- The number of epochs will be set to 75
- The image size will be set to 640x640

The screenshot below shows that the hyperparameter 2 settings are applied in the YOLOv7 training.

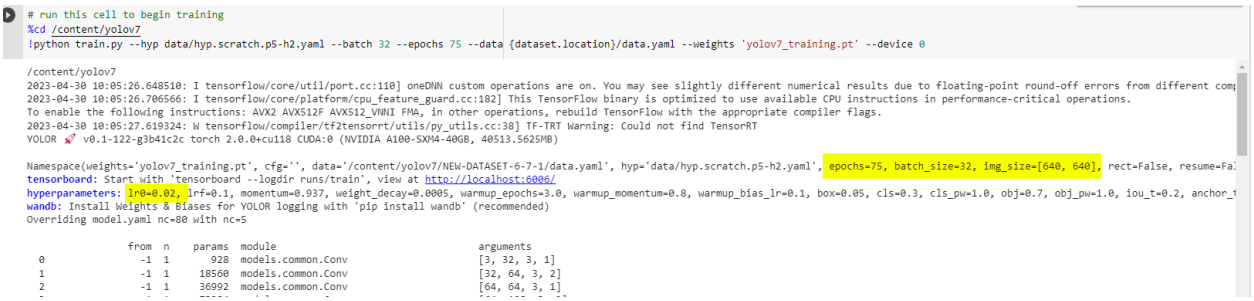


Figure 5.1: Hyperparameter 2 settings being ran

Training

On the hyperparameter 2 model training, we employed the hyperparameter settings of 0.02 on the Learning Rate or LR. The learning rate is often not employed directly while training a learned model. Rather, the learning rate is employed during the training phase to choose the optimal model parameters. When the model has been trained, the learning rate is commonly fixed, or the learning rate schedule is employed to gradually decrease the learning rate over time to fine-tune the performance of the model.

In terms of batch size, we employed 32 batches for model training. Batches are rarely employed directly during model training. Rather, the entire train dataset is typically evaluated all at once, and the model's performance is determined using metrics like accuracy, precision, recall, or F1 score. Certain frameworks or libraries, on the other hand, may use batches during inference to speed up the processing of large datasets.

Moving on to Epochs, we utilized 75 epochs for the hyperparameter 2 settings. The number of epochs utilized during training may have an indirect influence on the model's performance throughout training since it influences the quality of the learnt parameters and the model's capacity for generalization to new data.

Finally, for model training, we selected an image size of 640x640. During model training, the picture size is often set to a fixed number specified by the application's needs or the hardware's limits. To guarantee that the model gets evaluated on the same sort of data that it was trained on, the size of the images in the train dataset should match the size of the images employed during training.

And, as can be determined by the training results, there are six classes: Full-Face, Half-Face, Invalid, No-Helmet, and Rider. As you can see, all of the classes used 1203 images, and the "all" class has the most labels with 7067, followed by the "Rider" class with 3429, the "Full Faced" class with 1681 labels, the "Half Faced" class with 1493 labels, the "No Helmet" class with 234 labels, and finally the "Invalid" class with 230 labels.

Continuing toward precision, the "Rider" class has the highest precision rate, with a precision rate of 0.705, followed by the "Half Faced" class, which has a precision rate of 0.615, the "Invalid" class, which has a precision rate of 0.535, the "all" class, which has a precision rate of 0.535 the same as ”Invalid” class, the "Full Face" class, which has an accuracy rate of 0.464, and the "No Helmet" which has an accuracy of 0.354.

In terms of recall, the "Rider" class has the highest recall rate, with a rate of 0.606, followed by the "Full Face" class, which has a rate of 0.606, then the "all" class, which has a rate of 0.508, then the "Half Face" class, which has a rate of 0.493, then the "Invalid" class, which has a rate of 0.304, and finally the "No Helmet" which has an accuracy of 0.202.

The mean Average Precision at IoU=0.5 or mAP@.5 is also shown in the training result the “Rider” class has the highest Average Precision with an average precision of 0.853 and the lowest mean Average Precision from hyperparameter 2 is the “No Helmet” class with an average precision of 0.216.

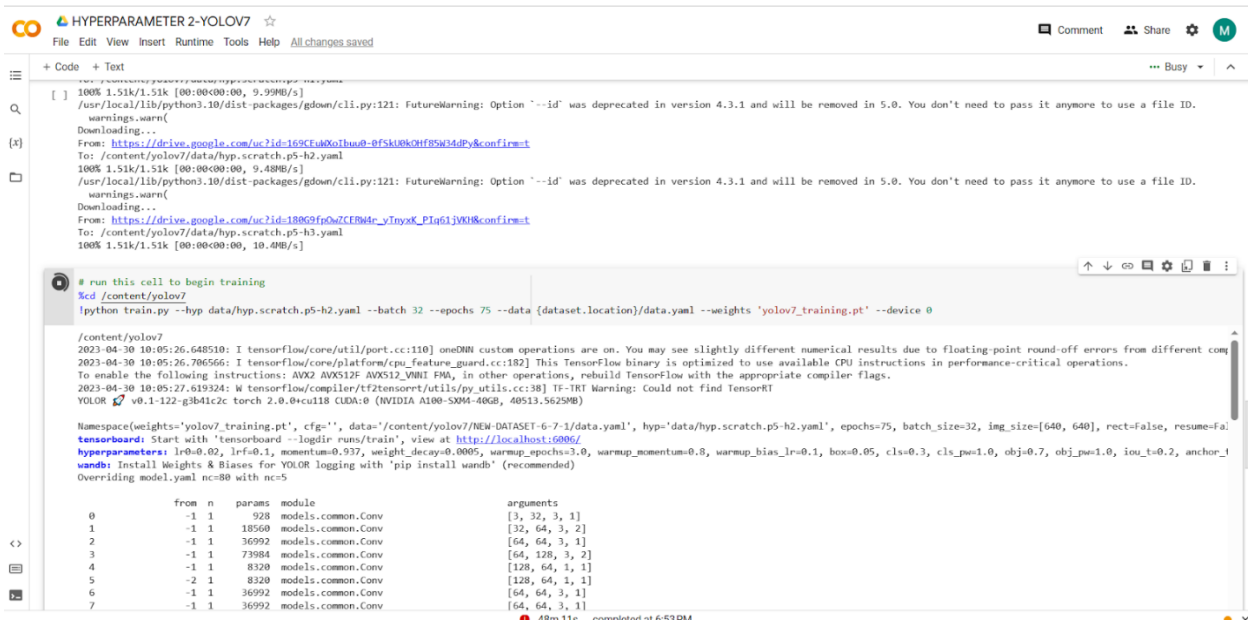


Figure 5.2: Training Hyperparameter 2 (beginning)

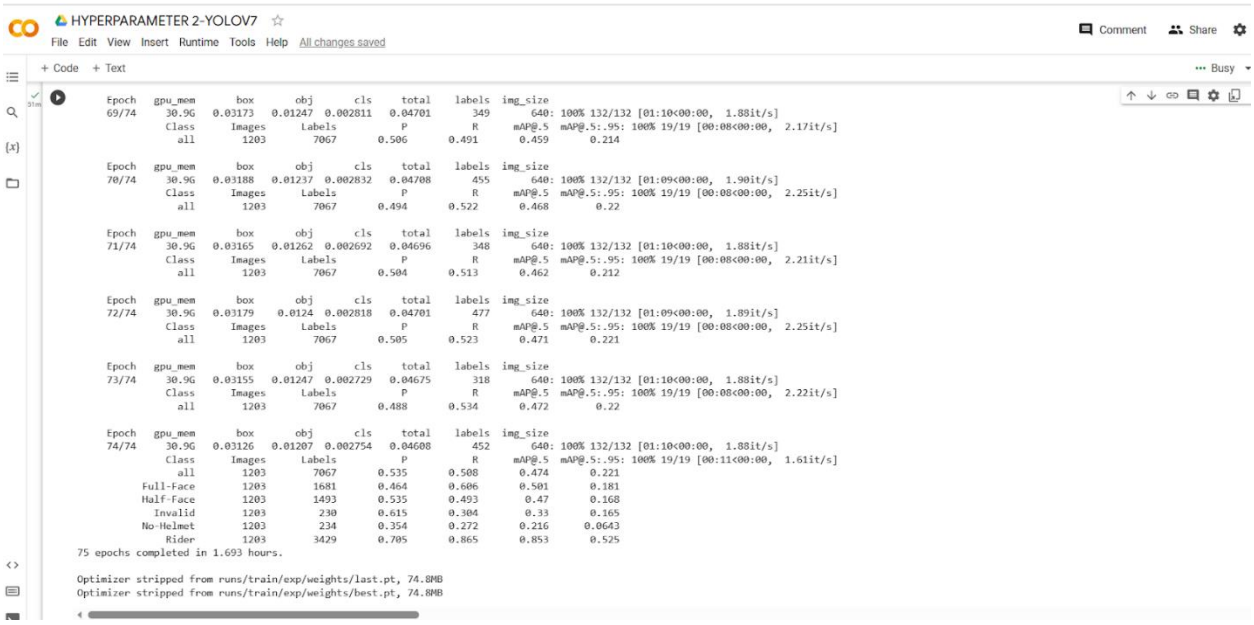


Figure 5.3: Training Hyperparameter 2 (ending)

Testing

On the model testing of the hyperparameter 2, we have used the hyperparameter settings of 0.02 setting on the Learning Rate or LR. When testing a trained model, the learning rate is typically not used directly. Instead, the learning rate is used during the training phase to find the optimal model parameters. Once the model is trained, the learning rate is typically set to a fixed value, or the learning rate schedule may be used to gradually decrease the learning rate over time to fine-tune the model's performance.

As for the batch, we have used 32 batches setting for the model testing of hyperparameter 2. During model testing, batches are not typically used directly. Instead, the entire test dataset is usually evaluated at once, and the model's performance is measured using metrics such as accuracy, precision, recall, or F1 score. However, some frameworks or libraries may use batches during inference to speed up the processing of large datasets.

Moving on to the Epochs we have used 75 epochs for the settings used in hyperparameter 2. The number of epochs used during training may have an indirect effect on the performance of the model during testing, as it determines the quality of the learned parameters and the ability of the model to generalize to new data.

And lastly, we have used 640x640 for the image size used for model testing of hyperparameter 2 as well. During model testing, the image size is usually set to a fixed value, which is determined by the requirements of the application or the constraints of the hardware. The size of the images in the test dataset should match the size of the images used during training, to ensure that the model is evaluated on the same type of data it was trained on.

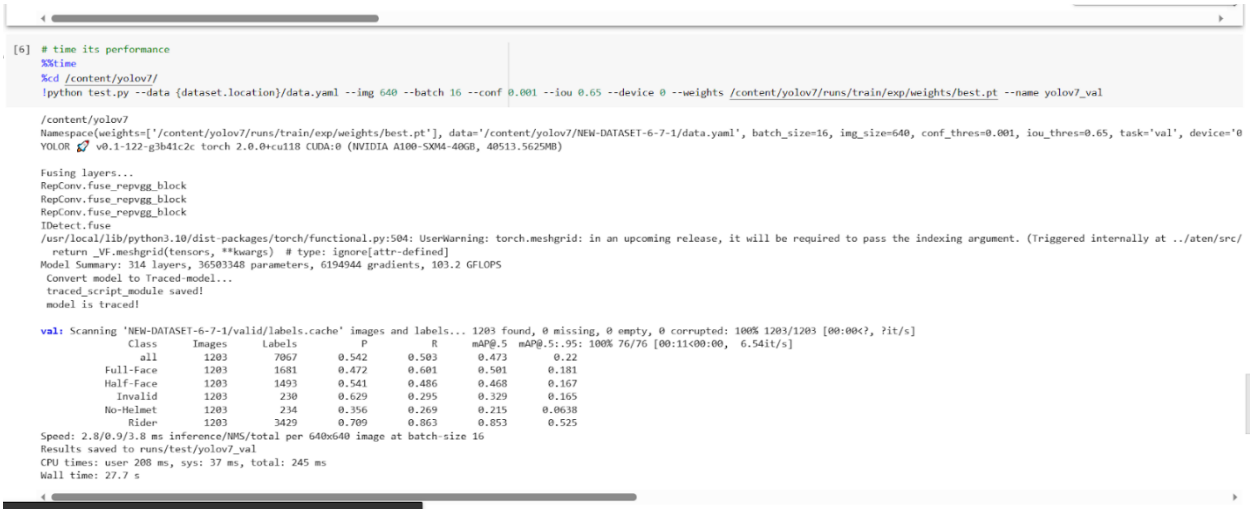
And as we can see on the testing results there are 6 classes the Full-Face, Half-Face, Invalid, No-Helmet the Rider, all. As we can see all the classes used 1203 images and for the Labels “all” class has the highest number of labels which has 7067 followed by the “Rider” class that has 3429 labels, the “Full

Faced" class has 1681 labels, the "Half Faced" has 1493 labels, the "No Helmet" has 234 labels, and finally the "Invalid" class which has 230 labels.

Moving on to the precision, the "Rider" class has the highest precision rate, which has a precision rate of 0.709 followed by the "Invalid" with a precision rate of 0.629, next is the "all" class which has a precision rate of 0.542, then "Half Face" class that has 0.541 precision rate, "Full Faced" class which has an accuracy rate of 0.472 and lastly with the lowest accuracy rate the "No Helmet" class which has an accuracy rate of 0.356.

As for the recall the "Rider" class has the highest recall rate, with a recall rate of 0.863 followed by "Full Faced" class which has a recall rate of 0.601, then the "all" class that has 0.503 recall rate, then the "Half Faced" class which has recall rate of 0.486, then the "Invalid" class with 0.295 recall rate, and lastly with the lowest recall rate the "No Helmet" class which has 0.269 recall rate.

The mean Average Precision at IoU=0.5 or mAP@.5 is also shown in the testing result the "Rider" class has the highest Average Precision with an average precision of 0.853 and the lowest mean Average Precision from hyperparameter 2 is the "No Helmet" class with an average precision of 0.215.



```
[6] # time its performance
    %%time
    %cd /content/yolov7/
    !python test.py --data {dataset.location}/data.yaml --img 640 --batch 16 --conf 0.001 --iou 0.65 --device 0 --weights /content/yolov7/runs/train/exp/weights/best.pt --name yolov7_val

    /content/yolov7
    Namespace(weights=['/content/yolov7/runs/train/exp/weights/best.pt'], data='/content/yolov7/NEW-DATASET-6-7-1/data.yaml', batch_size=16, img_size=640, conf_thres=0.001, iou_thres=0.65, task='val', device='0
    YOLOR 🚀 v0.1-122-g3b41c2c torch 2.0.0+cu118 CUDA:0 (NVIDIA A100-SXM4-40GB, 40513.5625MB)

    Fusing layers...
    RepConv.fuse_repvgg_block
    RepConv.fuse_repvgg_block
    RepConv.fuse_repvgg_block
    IDetect.fuse
    /usr/local/lib/python3.10/dist-packages/torch/functional.py:504: UserWarning: torch.meshgrid: in an upcoming release, it will be required to pass the indexing argument. (Triggered internally at ../aten/src/
      return _VF.meshgrid(tensors, **kwargs)  # type: ignore[attr-defined]
    Model Summary: 314 layers, 36503348 parameters, 6194944 gradients, 103.2 GFLOPS
     Convert model to Traced-model...
     traced_script_module saved!
     model is traced!

    val: Scanning 'NEW-DATASET-6-7-1/valid/labels.cache' images and labels... 1203 found, 0 missing, 0 empty, 0 corrupted: 100% 1203/1203 [00:00<?, ?it/s]
                   Class      Images      Labels           P           R      mAP@.5  mAP@.5:.95: 100% 76/76 [00:11<00:00,  6.54it/s]
                     all        1203        7067       0.542       0.503       0.473        0.22
               Full-Face        1203        1681       0.472       0.601       0.501       0.181
               Half-Face        1203        1493       0.541       0.486       0.468       0.167
                 Invalid        1203         230       0.629       0.295       0.329       0.165
               No-Helmet        1203         234       0.356       0.269       0.215      0.0638
                   Rider        1203        3429       0.709       0.863       0.853       0.525
    Speed: 2.8/0.9/3.8 ms inference/NMS/total per 640x640 image at batch-size 16
    Results saved to runs/test/yolov7_val
    CPU times: user 208 ms, sys: 37 ms, total: 245 ms
    Wall time: 27.7 s
```

Figure 5.4: Testing Hyperparameter 2

Tensorboard results

F1 Curve

The F1 curve shown has an F1 value and confidence value ranging from 0.0 to 1.0 and consists of six lines, each representing a different class. The full-face class is represented by a turquoise line, the half-face class by a gold line, the invalid cl ass by a green line, the no-helmet class by a red line, the rider class by a purple line, and all classes together by a bold blue line.

658 Muralla St., Intramuros, Manila, 1002, Philippines
Tel. No. : +63(2) 247-5000 local 5804
Telefax : +63(2) 891-0800
Web : www.mapua.edu.ph

At a confidence value of 0.260, the all classes line has an F1 value of 0.51, which is its highest value throughout the range. The rider line has a sharp increase in F1 value from 0.0 to 0.6 at a span of 0.2 confidence value and stabilizes at 0.7 F1 value. However, it gradually drops to a 1.0 confidence value at 0.8 F1 value.

The half-face line has a gradual increase in F1 value which peaks at 0.5 at a span of 0.2 confidence value and then continuously declines to 0.0 F1 value at around 0.8 confidence value. The invalid line has a peak increase of 0.3 F1 value and then continuously declines with slight increases at 0.0 F1 value at around 0.82 confidence value.

The no-helmet line has a gradual increase of 0.3 F1 value which declines slowly to 0.0 F1 value at around 0.75 confidence value. The full-face line has a gradual increase of 0.5 F1 value which slowly declines to 0.0 F1 value at 0.825 confidence value. Overall, the F1 curve shows that the different classes have varying levels of performance at different confidence levels. The rider and full-face classes tend to have the highest F1 values, while the half-face and no-helmet classes have lower F1 values. The invalid class has a peak F1 value but then declines rapidly, indicating that it may be difficult to accurately classify instances in this class.

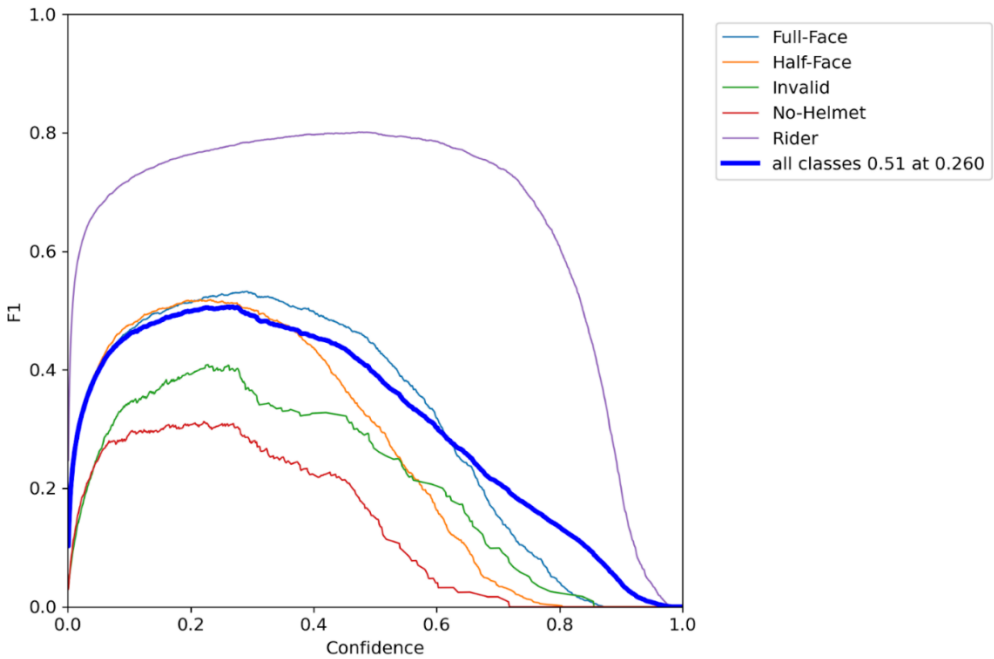


Figure 5.5: Hyperparameter 2 F1 Score

658 Muralla St., Intramuros, Manila, 1002, Philippines
Tel. No. : +63(2) 247-5000 local 5804
Telefax : +63(2) 891-0800
Web : www.mapua.edu.ph

Precision and Recall Curve

The PR curve shown has a Precision value and Recall value ranging from 0.0 to 1.0 and consists of six lines, each representing a different class. The full-face class is represented by a turquoise line, the half-face class by a gold line, the invalid class by a green line, the no-helmet class by a red line, the rider class by a purple line, and all classes together by a bold blue line.

The rider class has the highest mAP@0.5 value of 0.853, indicating that it has the highest overall performance among all the classes. The full-face class has a mAP@0.5 value of 0.501, indicating that it performs relatively well in terms of precision and recall.

The half-face class has a mAP@0.5 value of 0.470, indicating that it performs similarly to the full-face class. The invalid class has a mAP@0.5 value of 0.330, indicating that it has relatively poor performance compared to the other classes.

The no-helmet class has the lowest mAP@0.5 value of 0.216, indicating that it has the poorest performance among all the classes. The all classes line has a mAP@0.5 value of 0.474, indicating the overall performance of the system across all classes. Overall, the PR curve shows the trade-off between precision and recall for each class. The rider class has the highest precision and recall values, while the no-helmet class has the lowest. The other classes fall somewhere in between, with varying levels of precision and recall. The mAP@0.5 value provides an overall measure of the system's performance, taking into account the precision and recall values for all classes.

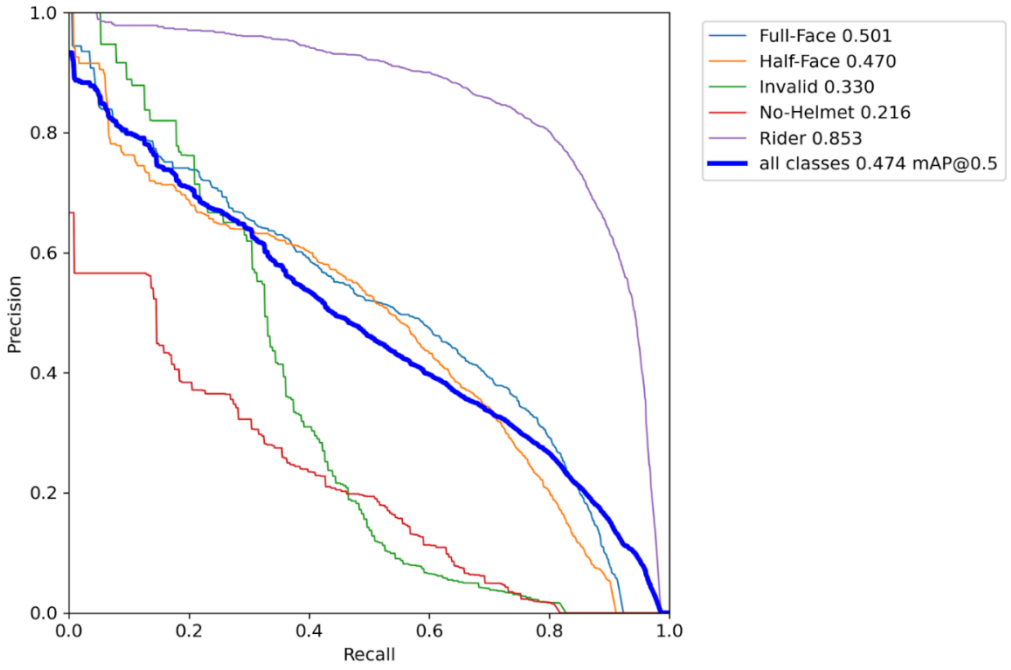


Figure 5.6: Hyperparameter 2 Precision and Recall Score

658 Muralla St., Intramuros, Manila, 1002, Philippines
Tel. No. : +63(2) 247-5000 local 5804
Telefax : +63(2) 891-0800
Web : www.mapua.edu.ph

Precision Curve

The P curve shown has a Precision value and Confidence value ranging from 0.0 to 1.0 and consists of six lines, each representing a different class. The full-face class is represented by a turquoise line, the half-face class by a gold line, the invalid class by a green line, the no-helmet class by a red line, the rider class by a purple line, and all classes together by a bold blue line.

The full-face line has a gradual increase in precision value, which peaks at 1.00 at a confidence value of 0.8. The half-face line also has a gradual increase, which peaks at 1.00 at a confidence value of 0.75. The invalid line has a sharp gradual increase, which peaks at 1.00 at a confidence value of 0.65.

The no-helmet line gradually increases to 0.55 precision value at a confidence value of 0.5, then gradually decreases to 0.25 precision value at a confidence value of 0.6, before rapidly increasing to 1.0 precision value at a confidence value of 0.7.

The rider line has a gradual and stable increase in precision value, which also peaks at 1.00 at a confidence value of 0.9. Lastly, all classes line has a gradual and unstable increase in precision value, which also peaks at 1.00 at a confidence value of 0.9. Overall, the P Curve shows how the precision value changes as the confidence value increases, giving an indication of the model's performance for different classes.

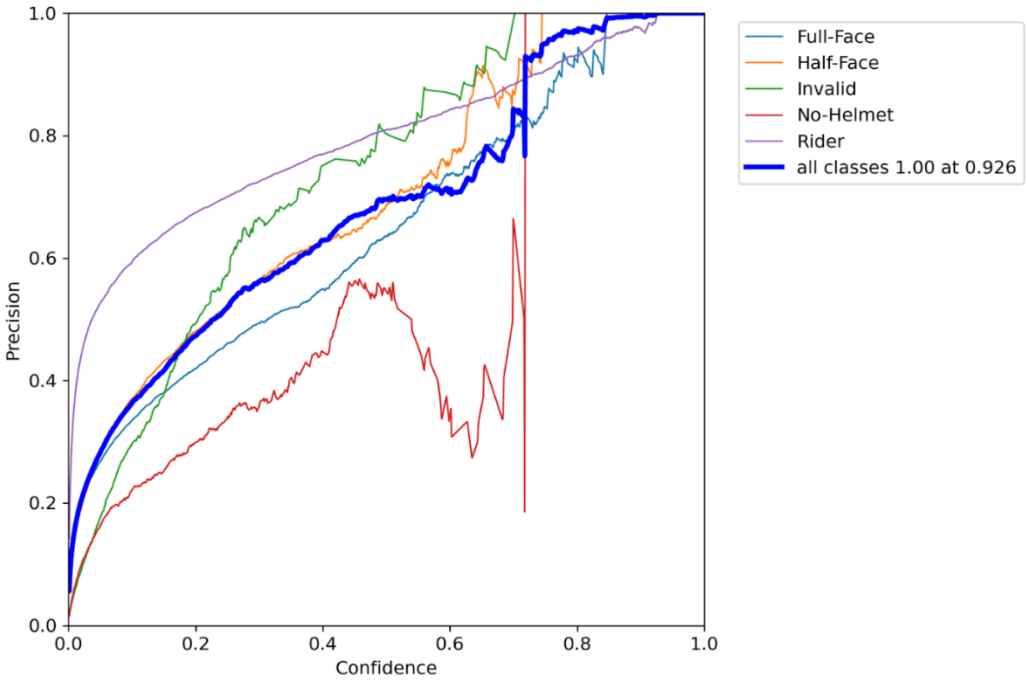


Figure 5.7: Hyperparameter 2 Precision Score

658 Muralla St., Intramuros, Manila, 1002, Philippines
Tel. No. : +63(2) 247-5000 local 5804
Telefax : +63(2) 891-0800
Web : www.mapua.edu.ph

Recall Curve

The R Curve has recall values on the y-axis and confidence values on the x-axis, ranging from 0.0 to 1.0. There are 6 lines on the graph: full-face with a turquoise line, half-face with a gold line, invalid with green line, no-helmet with a red line, rider with a purple line, and all classes with a bold blue line.

The full-face line gradually decreases from a recall value of 0.9 to 0.0 at a confidence value of 0.85. The half-face line also gradually decreases from a recall value of 0.85 to 0.0 at a confidence value of 0.8. Similarly, the invalid line decreases from a recall value of 0.7 to 0.0 at a confidence value of 0.85. The no-helmet line has a gradual decrease from a recall value of 0.8 to 0.0 at a confidence value of 0.7.

In contrast, the rider line has a steep increase from 0.0 to 1.0 recall value at a confidence value of 0.5, and then maintains a constant recall value of 1.0 until a confidence value of 0.9. Lastly, the all classes line has a steady decrease from a recall value of 1.0 to 0.0 as the confidence value increases from 0.0 to 1.0.

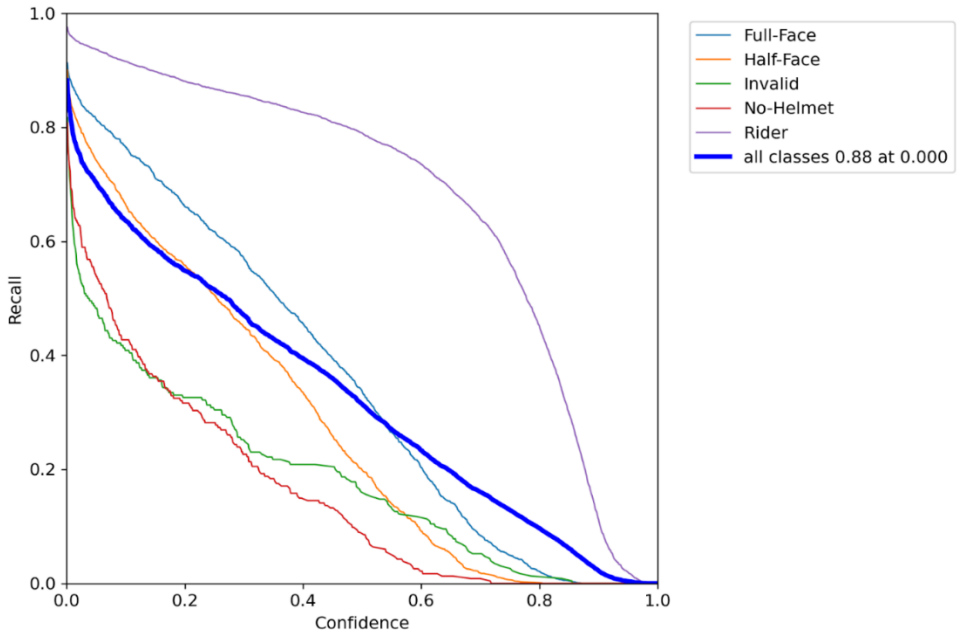


Figure 5.8: Hyperparameter 2 Recall Score

mAP@0.5 and mAP@0.5:0.95 scores

There are two mAP (mean average precision) graphs that display the mAP@0.5 and mAP@0.5:0.95 metrics, which are commonly used to evaluate object detection models. Overall, these graphs provide a comprehensive overview of the performance of the object detection model and can be used to identify areas for improvement. After training the YOLOv7 model, two scores were calculated: mAP@0.5 and mAP@0.5:0.95. The first score measures the mean average precision at an IoU threshold of 0.5 while the second score measures the mean average precision over different IoU thresholds from 0.5 to 0.95. The results showed that the mAP@0.5 score was 0.474 after 74 epochs and the mAP@0.5:0.95 score

was 0.2206. A good mAP@0.5 score is usually above 0.5, so the model's score of 0.474 indicates moderate accuracy in detecting objects. The low mAP@0.5:0.95 score of 0.2206 suggests that the model struggles to accurately identify objects with high confidence.

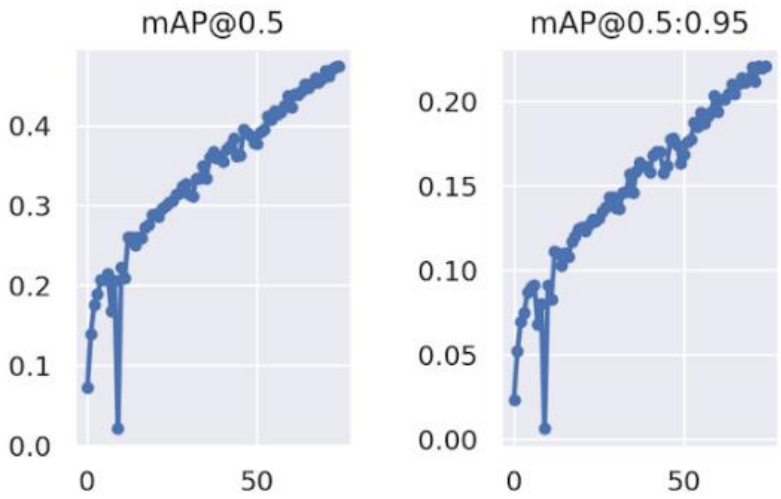


Figure 5.9: Hyperparameter 2 mAP@0.5 and mAP@0.5:0.95 scores

Batch 0 Labels

The Batch 0 labels for hyperparameter 2, which is designed to recognize riders, no riders, and different types of helmets, consist of five categories: rider, no rider, full-face helmet, half-face helmet, and no helmet. These labels are used to train a machine learning model to classify objects in CCTV footage as one of these categories. The rider category refers to a person who is riding a motorcycle, while the no rider category refers to a motorcycle without a rider. The full-face helmet category refers to a helmet that covers the entire face, while the half-face helmet category refers to a helmet that only covers part of the face. Lastly, the no helmet category refers to a person riding without a helmet. These labels are crucial for training the model to accurately identify and classify objects in the CCTV footage, which can help improve safety measures and prevent accidents on the road.

Figure 5.10: Hyperparameter 2 Batch 0 Label

Batch 0 Predicted

The batch 0 predicted values include a list of detections made by the model in the first batch of input data. Each detection may contain information such as the predicted class (rider, no rider, full-face helmet, half-face helmet, no helmet), the confidence score for that prediction, and the location of the detection in the video frame. The model also provides additional information about the prediction, such as the size and aspect ratio of the detected object. The predicted value for the batch is visualized using bounding boxes around the detected objects in the input images or video frames. The visualizations shown are overlaid on top of the original footage to show where the model detected riders and helmets. The predicted values could also be used to calculate various performance metrics, such as precision, recall, and mean average precision, to evaluate how well the model is performing on the given task.

658 Muralla St., Intramuros, Manila, 1002, Philippines
Tel. No. : +63(2) 247-5000 local 5804
Telefax : +63(2) 891-0800
Web : www.mapua.edu.ph

Figure 5.11: Hyperparameter 2 Batch 0 Predicted

658 Muralla St., Intramuros, Manila, 1002, Philippines
Tel. No. : +63(2) 247-5000 local 5804
Telefax : +63(2) 891-0800
Web : www.mapua.edu.ph

MSE

The MSE value of 0.05195 indicates that the model's average squared error in prediction is 0.05195. A lower MSE value indicates better model performance, as it means that the model's predictions are closer to the actual values.

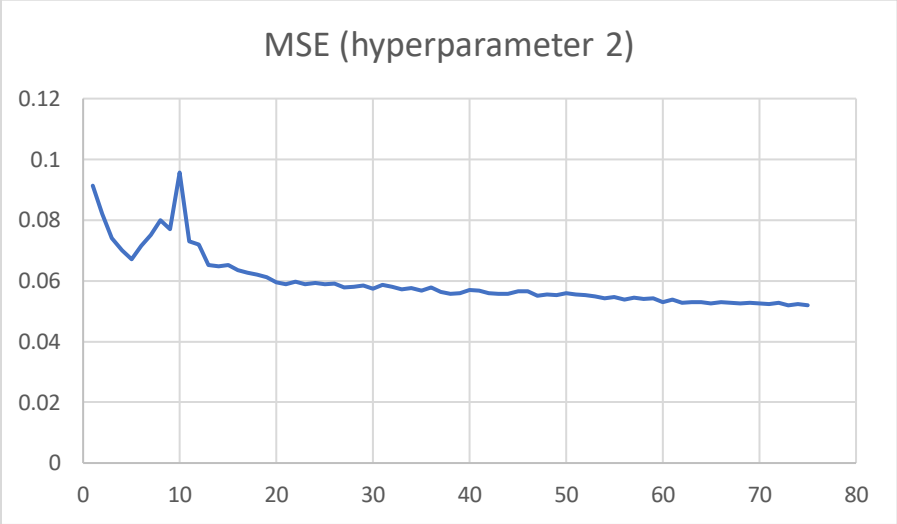


Figure 5.12: Hyperparameter 2 MSE

RMSE

The RMSE value of 0.2279254264 is the square root of the MSE value, and represents the average difference between the predicted and actual values. This metric is preferred over MSE as it has the same unit as the target variable, making it easier to interpret the model's performance.

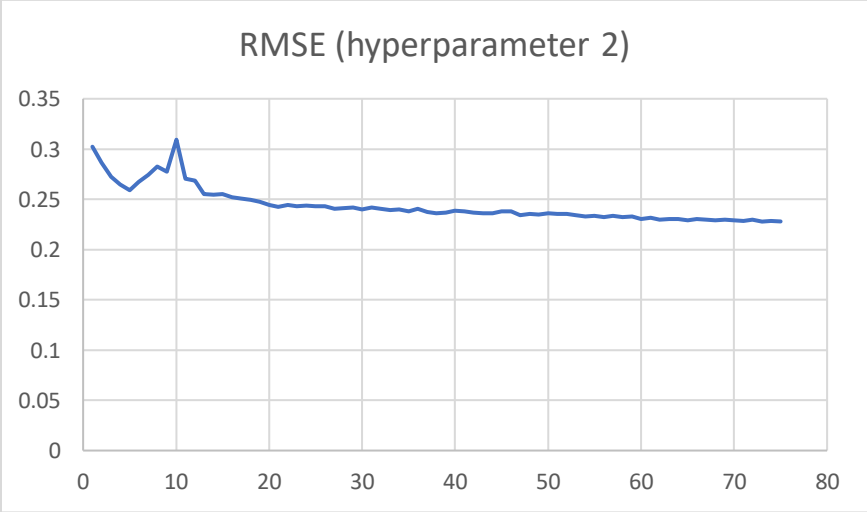


Figure 5.13: Hyperparameter 2 RMSE

658 Muralla St., Intramuros, Manila, 1002, Philippines
Tel. No. : +63(2) 247-5000 local 5804
Telefax : +63(2) 891-0800
Web : www.mapua.edu.ph

Confusion Matrix

From the confusion matrix obtained from applying the second set of hyperparameter settings, the highest true positive value obtained is for the Rider class which has a 0.87 value which has decreased from its value of 0.93 with the first set of hyperparameter settings. The second and third highest true positive value obtained are for the Full-Face and Half-Face classes which have a value of 0.55 and 0.53 respectively. The fourth and fifth highest true positive value obtained are for the No-Helmet and Invalid classes which have a value of 0.30 and 0.29 respectively. The four aforementioned classes have all experienced a TP value increase aside from the Invalid class which had increased 0.001 in value compared from the results with the first set of hyperparameter settings. From the predicted background FN, it can be observed that there is still a significant number of Invalid and No-Helmet (0.59 and 0.64) annotated which have not been detected as part of the class. With the background FN value for the true No-Helmet class, it has increased in value therefore making the second set of hyperparameter settings disregard more instances of the No-Helmet class in detection. Conversely, from the predicted background FP, there is a significant number of Full-Face and Rider (0.32 and 0.45) instances which have been detected from the image background which have not been originally annotated. The Full-Face, Half-Face, and Invalid classes still have increased false negative values compared to the classes of No-Helmet and Rider. As observed in the Full-Face and Half-Face classes, the Full-Face class has a decreased false positive value for the Half-Face class (0.17), while the Half-Face class also has an increased false positive value for the Full-Face class (0.12).

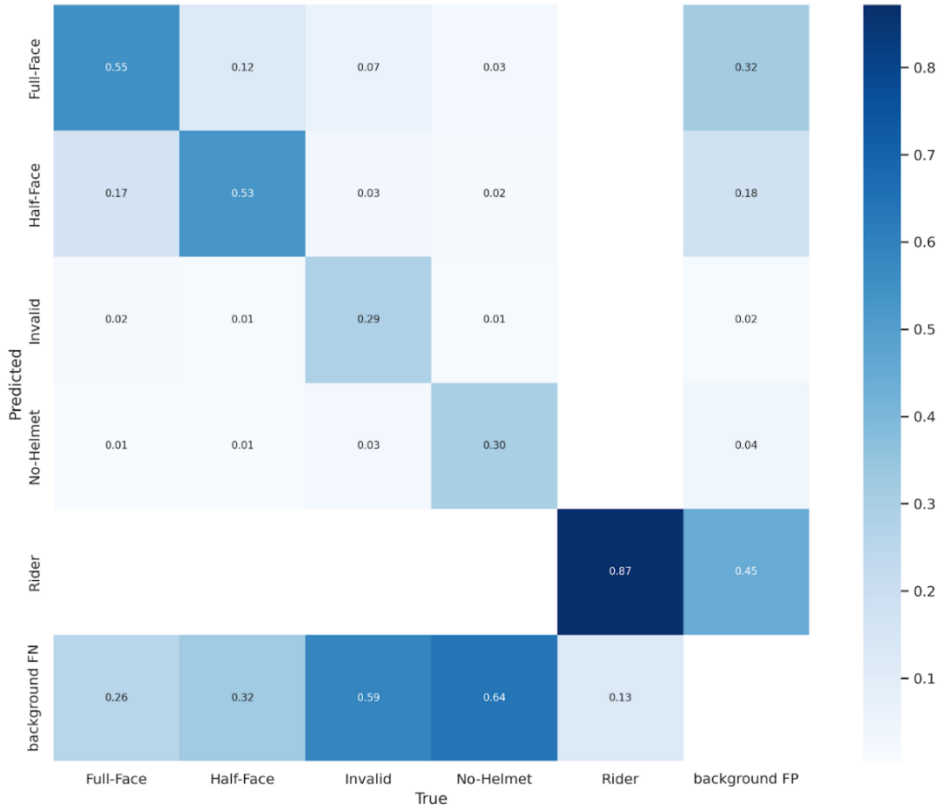


Figure 5.14: Hyperparameter 2 Confusion Matrix

658 Muralla St., Intramuros, Manila, 1002, Philippines
Tel. No. : +63(2) 247-5000 local 5804
Telefax : +63(2) 891-0800
Web : www.mapua.edu.ph

Model Output

The testing of Hyperparameter 2, which employs a Learning Rate setting of 0.02, 32 batches, 75 epochs, and 640x640 picture size for model testing, is shown in the video output. Six classes—Full-Face, Half-Face, Invalid, No-Helmet, Rider, and All—are used to test the model using metrics including accuracy, recall, and mean average precision. According to the F1 curve, the Half-Face class has a gradually increasing F1 value that peaks at 0.5, whereas the Rider class has the greatest F1 value.

Video Output (DeepSORT)

After using the hyperparameter 2 settings to train the YOLOv7 model, the group was able to produce a video output of the DeepSORT tracker using the YOLOv7 weights. The following screenshot displays a single frame from the resulting video.

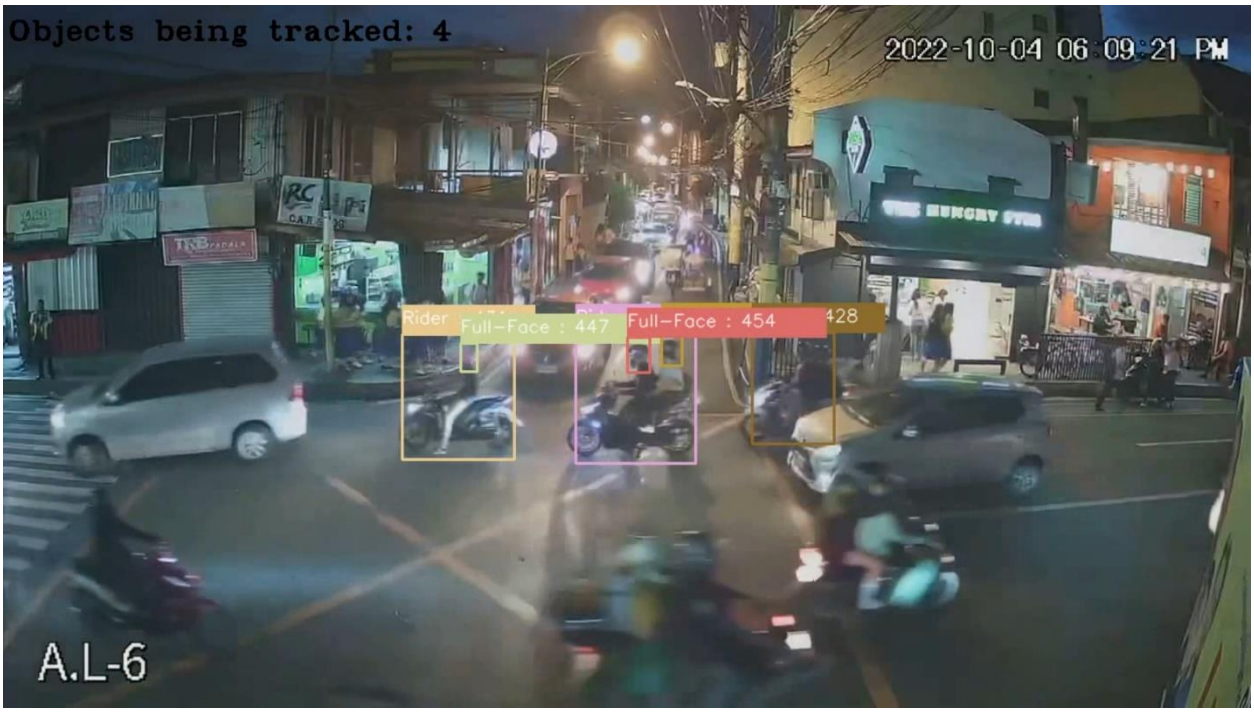


Figure 5.15: Hyperparameter 2 Video output DeepSORT

From this frame, we can see that the DeepSORT works because we are able to detect the objects in the video and uniquely identify objects by their assigned tracker IDs.

Hyperparameter 3

As required for this activity, we will set the third hyperparameter settings of the YOLOv7 model training to the following specifications:

- The learning rate will be set to 0.03
- The batch size will be set to 16
- The number of epochs will be set to 100
- The image size will be set to 640x640

The screenshot below shows that the hyperparameter 3 settings are applied in the YOLOv7 training

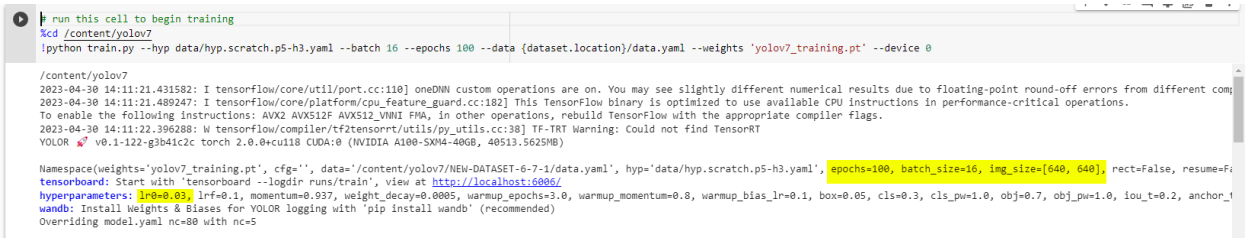


Figure 6.1: Hyperparameter 2 settings being ran

Training

On the hyperparameter 3 model training, we employed the hyperparameter settings of 0.03 on the Learning Rate or LR. The learning rate is often not employed directly while training a learned model. Rather, the learning rate is employed during the training phase to choose the optimal model parameters. When the model has been trained, the learning rate is commonly fixed, or the learning rate schedule is employed to gradually decrease the learning rate over time to fine-tune the performance of the model.

In terms of batch size, we employed 16 batches for model training. Batches are rarely employed directly during model training. Rather, the entire train dataset is typically evaluated all at once, and the model's performance is determined using metrics like accuracy, precision, recall, or F1 score. Certain frameworks or libraries, on the other hand, may use batches during inference to speed up the processing of large datasets.

Moving on to Epochs, we utilized 100 epochs for the hyperparameter 3 settings. The number of epochs utilized during training may have an indirect influence on the model's performance throughout training since it influences the quality of the learnt parameters and the model's capacity for generalization to new data.

Finally, for model training, we selected an image size of 640x640. During model training, the picture size is often set to a fixed number specified by the application's needs or the hardware's limits. To guarantee that the model gets evaluated on the same sort of data that it was trained on, the size of the images in the train dataset should match the size of the images employed during training.

And, as can be determined by the training results, there are six classes: Full-Face, Half-Face, Invalid, No-Helmet, and Rider. As you can see, all of the classes used 1203 images, and the "all" class has the most labels with 7067, followed by the "Rider" class with 3429, the "Full Faced" class with 1681 labels, the "Half Faced" class with 1493 labels, the "No Helmet" class with 234 labels, and finally the "Invalid" class with 230 labels.

Continuing toward precision, the "No Helmet" class has the highest precision rate, with a precision rate of 1, followed by the "Invalid" with a precision rate of 1, same as "No helmet, the "all" class, which has a precision rate of 0.592, the "all" class, which has a precision rate of 0.447, the "Full Face" class, which has an accuracy rate of 0.259, and the "Half-Face" which has an accuracy of 0.253.

In terms of recall, the "Rider" class has the highest recall rate, with a rate of 0.739, followed by the "all" class, which has a rate of 0.221, then the "Full-Face" class, which has a rate of 0.202, then the "Half Faced" class, which has a rate of 0.163, then the "Invalid" class, which has a rate of 0.0, and finally the "No Helmet" which has an accuracy of 0.0.

The mean Average Precision at IoU=0.5 or mAP@.5 is also shown in the training result the "Rider" class has the highest Average Precision with an average precision of 0.579 and the lowest mean Average Precision from hyperparameter 3 is the "Invalid" class with an average precision of 0.00318.

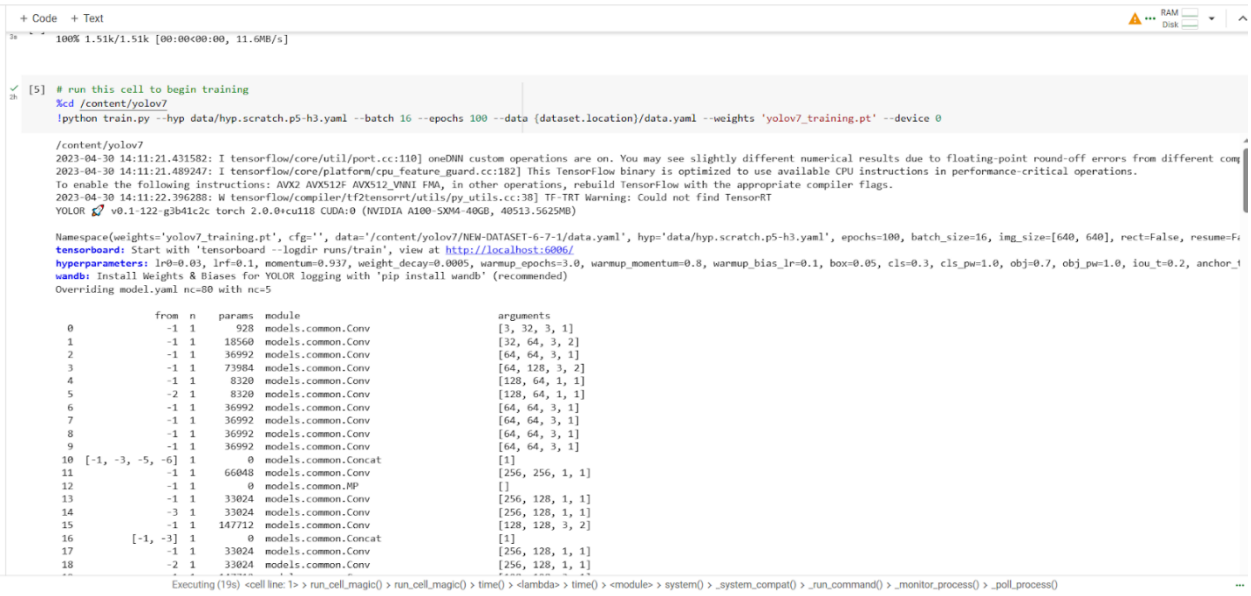


Figure 6.2: Training Hyperparameter 3 (beginning)

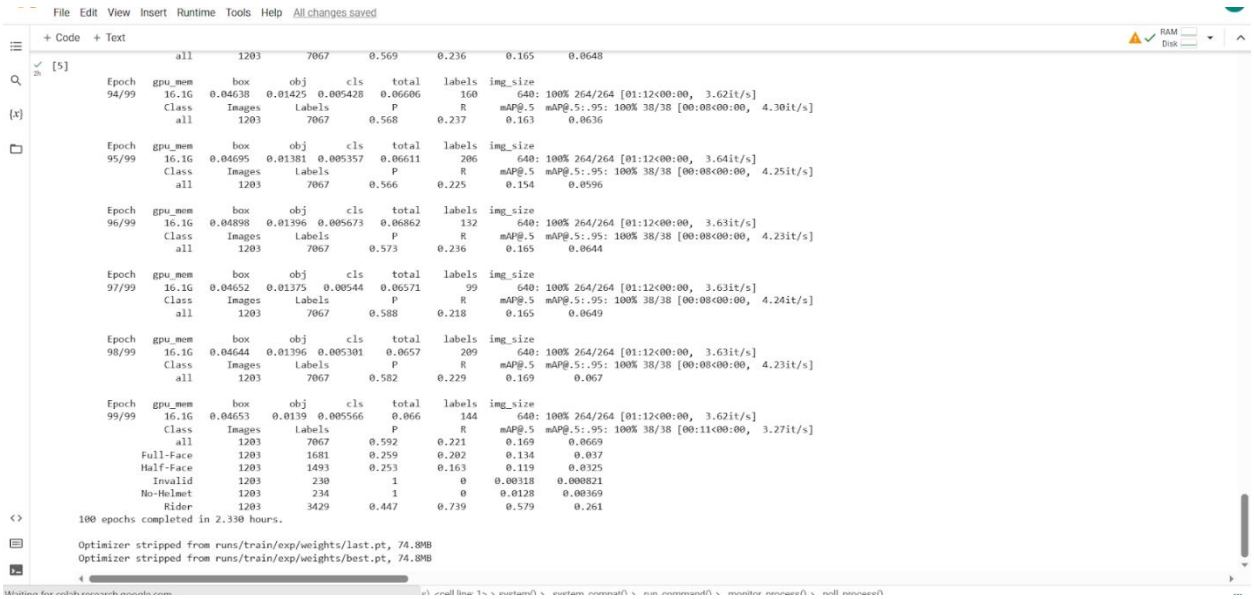


Figure 6.3: Training Hyperparameter 3 (ending)

Testing

On the model testing of the hyperparameter 3, we have used the hyperparameter settings of 0.03 setting on the Learning Rate or LR. When testing a trained model, the learning rate is typically not used directly. Instead, the learning rate is used during the training phase to find the optimal model parameters. Once the model is trained, the learning rate is typically set to a fixed value, or the learning rate schedule may be used to gradually decrease the learning rate over time to fine-tune the model's performance.

As for the batch, we have used 16 batches setting for the model testing of hyperparameter 3. During model testing, batches are not typically used directly. Instead, the entire test dataset is usually evaluated at once, and the model's performance is measured using metrics such as accuracy, precision, recall, or F1 score. However, some frameworks or libraries may use batches during inference to speed up the processing of large datasets.

Moving on to the Epochs we have used 100 epochs for the settings used in hyperparameter 3. The number of epochs used during training may have an indirect effect on the performance of the model during testing, as it determines the quality of the learned parameters and the ability of the model to generalize to new data.

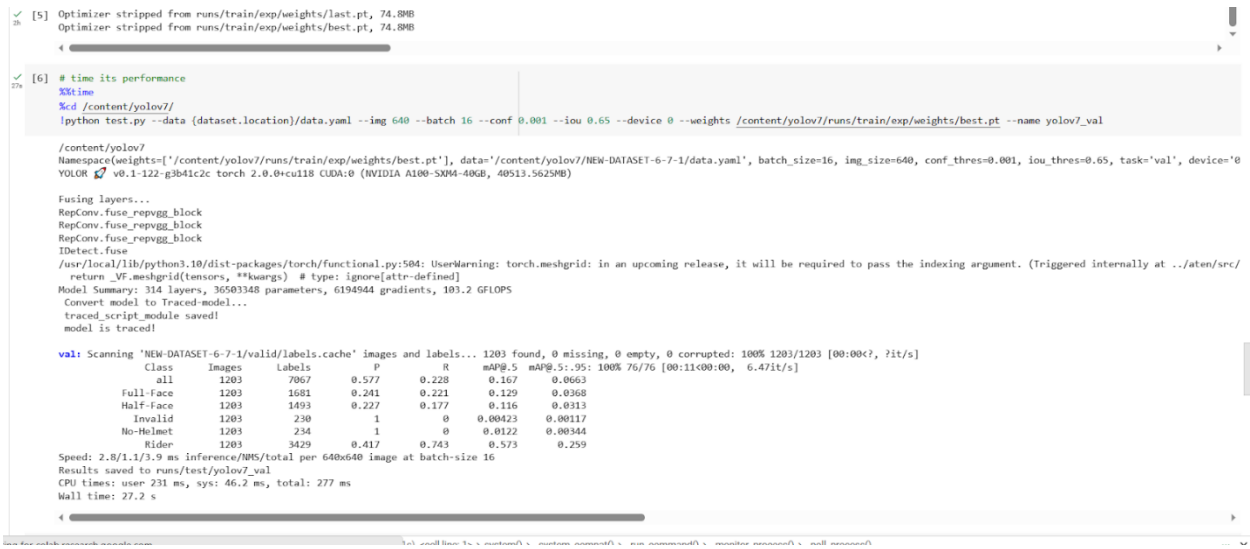
And lastly, we have used 640x640 for the image size used for model testing of hyperparameter 3 as well. During model testing, the image size is usually set to a fixed value, which is determined by the requirements of the application or the constraints of the hardware. The size of the images in the test dataset should match the size of the images used during training, to ensure that the model is evaluated on the same type of data it was trained on.

And as we can see on the testing results there are 6 classes the Full-Face, Half-Face, Invalid, No-Helmet the Rider, all. As we can see all the classes used 1203 images and for the Labels “all” class has the highest number of labels which has 7067 followed by the “Rider” class that has 3429 labels, the “Full Faced” class has 1681 labels, the “Half Faced” has 1493 labels, the “No Helmet” has 234 labels, and finally the “Invalid” class which has 230 labels.

Moving on to the precision, the “Invalid” and “No Helmet” class has the highest precision rate, which has a precision rate of 0.1 followed by the “all” with a precision rate of 0.577, next is the “Rider” class which has a precision rate of 0.417, then “Full Face” class that has 0.241 precision rate, and lastly with the lowest accuracy rate the “Half Face” class which has an accuracy rate of 0.227.

As for the recall the “Rider” class has the highest recall rate, with a recall rate of 0.743 followed by “all” class which has a recall rate of 0.228, then the “Full Face” class that has 0.221 recall rate, then the “Half Faced” class which has recall rate of 0.177, and lastly with the lowest recall rate the “No Helmet” and “Invalid” class which has 0.0 recall rate.

The mean Average Precision at IoU=0.5 or mAP@.5 is also shown in the testing result the “Rider” class has the highest Average Precision with an average precision of 0.573 and the lowest mean Average Precision from hyperparameter 3 is the “No Helmet” class with an average precision of 0.00423.



```
[5] Optimizer stripped from runs/train/exp/weights/last.pt, 74.8MB
    Optimizer stripped from runs/train/exp/weights/best.pt, 74.8MB

[6] # time its performance
    %%time
    %cd /content/yolov7/
    !python test.py --data {dataset.location}/data.yaml --img 640 --batch 16 --conf 0.001 --iou 0.65 --device 0 --weights /content/yolov7/runs/train/exp/weights/best.pt --name yolov7_val

/content/yolov7
Namespace(weights=['/content/yolov7/runs/train/exp/weights/best.pt'], data='/content/yolov7/NEW-DATASET-6-7-1/data.yaml', batch_size=16, img_size=640, conf_thres=0.001, iou_thres=0.65, task='val', device='0
YOLOR 🚀 v0.1-122-g3b41c2c torch 2.0.0+cu118 CUDA:0 (NVIDIA A100-SXM4-40GB, 40513.5625MB)

Fusing layers...
RepConv.fuse_repvgg_block
RepConv.fuse_repvgg_block
RepConv.fuse_repvgg_block
IDetect.fuse
/usr/local/lib/python3.10/dist-packages/torch/functional.py:504: UserWarning: torch.meshgrid: in an upcoming release, it will be required to pass the indexing argument. (Triggered internally at ../aten/src/
  return _VF.meshgrid(tensors, **kwargs)  # type: ignore[attr-defined]
Model Summary: 314 layers, 36503348 parameters, 6194944 gradients, 103.2 GFLOPS
 Convert model to Traced-model...
 traced_script_module saved!
 model is traced!

val: Scanning 'NEW-DATASET-6-7-1/valid/labels.cache' images and labels... 1203 found, 0 missing, 0 empty, 0 corrupted: 100% 1203/1203 [00:00<?, ?it/s]
               Class      Images      Labels           P           R      mAP@.5  mAP@.5:.95: 100% 76/76 [00:11<00:00,  6.47it/s]
                 all        1203        7067       0.577       0.228       0.167      0.0663
           Full-Face        1203        1681       0.241       0.221       0.129      0.0368
           Half-Face        1203        1493       0.227       0.177       0.116      0.0313
             Invalid        1203         230           1           0     0.00423     0.00117
           No-Helmet        1203         234           1           0      0.0122     0.00344
               Rider        1203        3429       0.417       0.743       0.573       0.259
Speed: 2.8/1.1/3.9 ms inference/NMS/total per 640x640 image at batch-size 16
Results saved to runs/test/yolov7_val
CPU times: user 231 ms, sys: 46.2 ms, total: 277 ms
Wall time: 27.2 s
```

Figure 6.4: Testing Hyperparameter 3

658 Muralla St., Intramuros, Manila, 1002, Philippines
Tel. No. : +63(2) 247-5000 local 5804
Telefax : +63(2) 891-0800
Web : www.mapua.edu.ph

Tensorboard results

F1 Curve

The following F1-curve presented above is primarily concerned with the F1-score results using hyperparameter 3 specifications. In this illustration, the five classes are still included which are the following: Full-face, Half-face, Invalid, No-Helmet, and Rider. The color-coded legends also indicate which class is represented in every curve.

As you can see in hyperparameter 3 f1-curve results, all the classes netted an average score of 0.20 at 0.139 confidence level. The behavior of Rider class curve result is similar with other hyperparameters specification wherein the curve trend still represents the highest peak progression compared to the other classes. The highest peak of classes Full-face and Half-face are above the average score in this hyperparameter while the remaining classes No-Helmet and Invalid yielded below average.

Given the curve value results, this indicates that the model's performance utilizing hyperparameter 3 specifications is poor when compared with other hyperparameters. A noticeable lower f1-score occurred as the confidence level keeps rising. Despite of this, the curve trends is still the same with other hyperparameters wherein the trade-off between F1 score and confidence level is still noticeable. There is a higher score when the confidence is at a lower level.

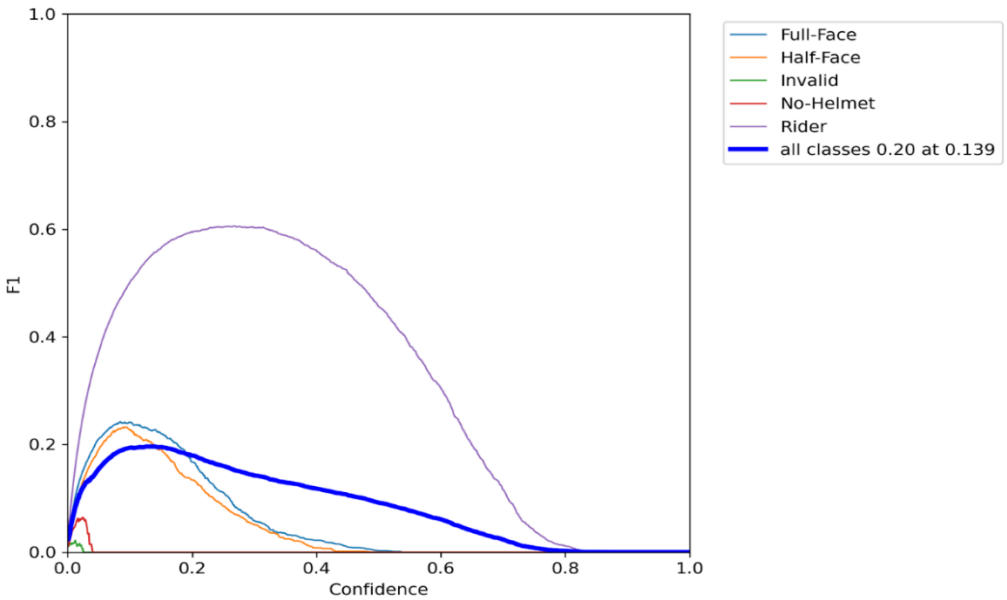


Figure 6.5: Hyperparameter 3 F1 Score

658 Muralla St., Intramuros, Manila, 1002, Philippines
Tel. No. : +63(2) 247-5000 local 5804
Telefax : +63(2) 891-0800
Web : www.mapua.edu.ph

Precision and Recall Curve

The following illustrates the precision-recall curve when using hyperparameter 3 specifications. Similarly, it still includes the four classes: Full-face, Half-face, Invalid, No Helmet, and Rider. Color legends are still the same wherein the average for all classes is the curve-colored blue.

The precision-recall curves for each classes varies. On all classes, the mAP@0.5 is 0.169. This result is noticeably poor when compared to the results from other hyperparameters. Similar with other classes, the curve trend still continues. The rider class still has the highest precision when compared with other classes at 0.579. Invalid class on the other hand has the lowest at 0.003 followed by No-Helmet at 0.013. This is mainly because of lacking detectable data presented for these kinds of classification. Full-face and Half-face classes are both very similar when it comes to their netted precision values. These two classes are just shy under the average mark.

In addition to this curve illustration, the trade-offs between Precision and Recall values are still very noticeable in each class. This curve trend is mainly due to concept that when more true positives are predicted, more false positives can also occur leading to lower precision. However, if less true positives are predicted, this could also result to fewer number of false positives, which will then result to increase the precision category.

The overall results of hyperparameter three when compared to the previous two hyperparameters is less desirable. Possible factor to blame could be the lessened "batch size" or the number of training examples used.

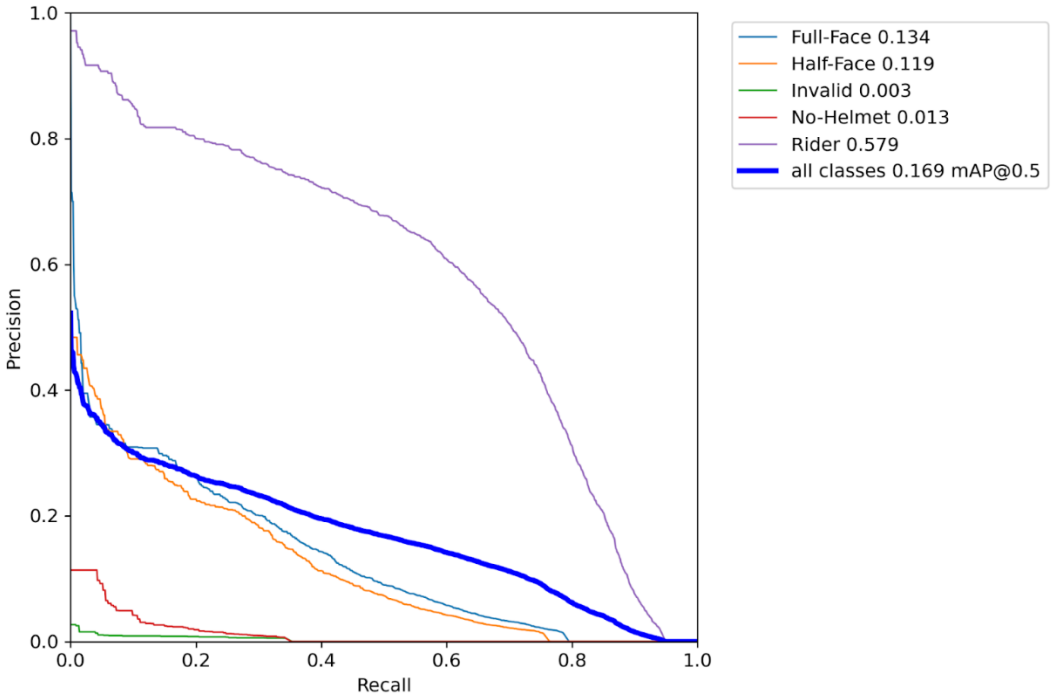


Figure 6.6: Hyperparameter 3 Precision and Recall Score

Precision Curve

The following graphical representation pertains to the P curve result using hyperparameter three specifications. Ranging from 0.0 to 1.0 Precision and Confidence Level, different classes are represented by different curves. The color-coded legends are as is, wherein the average of all classes is represented by the colored-blue curve.

In this P-curve illustration, it shows that the average for all classes is 0.100 precision at 0.862 confidence level. Comparing this to other hyperparameters result, it is noticeable how it yields the same 1.0 precision but at a lower confidence level. This imply that the model using hyperparameter three specifications has a slightly reduced performance. Nevertheless, the precision performance on average is desirable. Other classes such as Rider and Half-face also have a noticeable drop down and peak at certain confidence level. For instance, the curve trend for Half-face class suddenly dropped when it reached the 0.5 confidence level mark but then spiked up immediately. Similar can be said with the Rider class when its precision heavily dropped and progress upwards.

Out of all the classes, the most stable curve trend is the Full-Face class. This could imply that the Full-Face class has the most well represented data in the model. Classes Invalid and No Helmet on the other hand both have a high precision rate but at a very low confidence level. This suggests that there is still uncertainty regarding its accuracy possible due to lack of represented data.

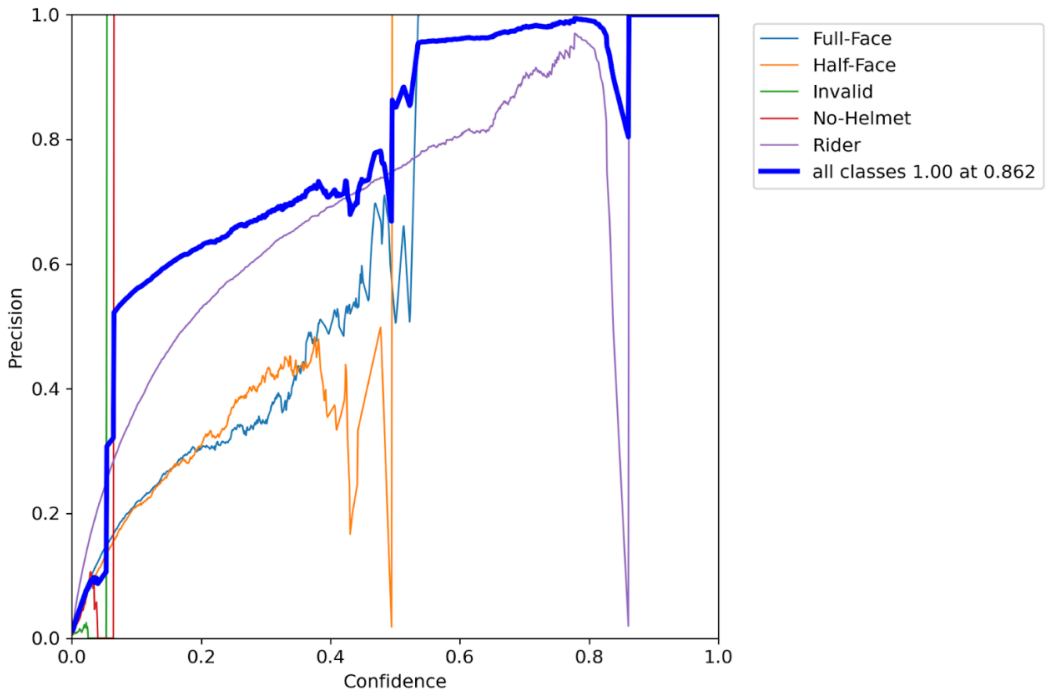


Figure 6.7: Hyperparameter 3 Precision Score

658 Muralla St., Intramuros, Manila, 1002, Philippines
Tel. No. : +63(2) 247-5000 local 5804
Telefax : +63(2) 891-0800
Web : www.mapua.edu.ph

Recall Curve

The following graphical representation presented above refers to the R-curve result of the model using hyperparameter three specifications. Similar with the previous graphical representations, this curve includes the five classes namely: Full-face, Half-Face, Invalid, No-Helmet, and Rider. Each class is represented by a color-coded curve. The legend is also presented.

Based on the presented curve trends, we can see that the average recall is 0.63 at 0.000 confidence level. With this average result, there is a possible scenario where the model using hyperparameter three could detect medium proportion of true positives cases but there is a level of a high level of uncertainty as false positives could also occur since the confidence level is very low. Comparing the average recall of this specific hyperparameter specification, we can see that the recall rate is noticeable lower. This implies that the previous hyperparameters is better at detecting proportions of true positives.

Discussing the curve trends for other classes, it is noticeable how the Rider class has the highest recall rate out of every other classes. This is followed by classes Full-face and Half-face wherein they yielded an above average recall. Classes Invalid and No-Helmet have the lowest recall value even at lower confidence level. Ultimately, the trend related with the concept of trade-off continues as all the curves tend to have higher recall rate at lower confidence levels.

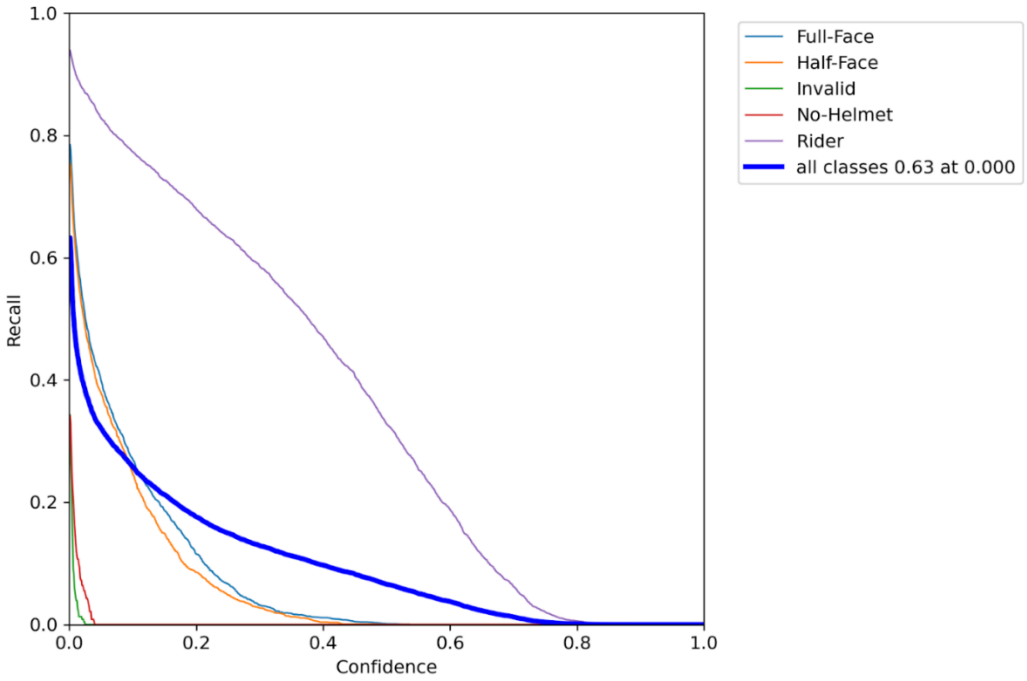


Figure 6.8: Hyperparameter 3 Recall Score

mAP@0.5 and mAP@0.5:0.95 scores

There are two mAP (mean average precision) graphs that display the mAP@0.5 and mAP@0.5:0.95 metrics, which are commonly used to evaluate object detection models. Overall, these graphs provide a comprehensive overview of the performance of the object detection model and can be used to identify areas for improvement. After training the YOLOv7 model, two scores were calculated: mAP@0.5 and mAP@0.5:0.95. The first score measures the mean average precision at an IoU threshold of 0.5 while the second score measures the mean average precision over different IoU thresholds from 0.5 to 0.95. The results showed that the mAP@0.5 score was 0.1695 after 99 epochs and the mAP@0.5:0.95 score was 0.06691. Based on the results of the YOLOv7 model, the mAP recieved relavitely low scores for both IoU thresholds. An mAP@0.5 score of 0.1695 indicates that the model is correctly detecting approximately 16.95% of the objects in the image with a 50% overlap (IoU) between the predicted bounding box and ground truth. Similarly, an mAP@0.5:0.95 score of 0.06691 means that the model is correctly detecting approximately 6.69% of the objects in the image across different IoU thresholds from 0.5 to 0.95.

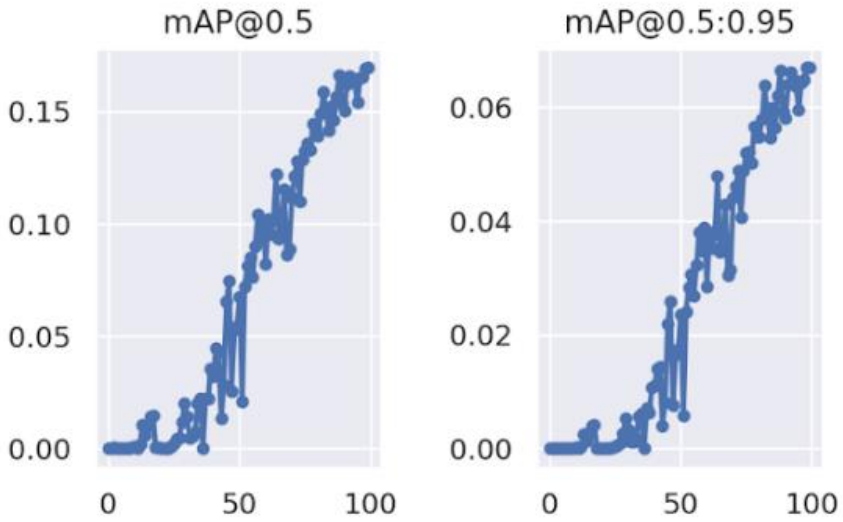


Figure 6.9: Hyperparameter 3 mAP@0.5 and mAP@0.5:0.95 scores

Batch 0 Labels

The Batch 0 labels for hyperparameter 3 have been made to show different types of helmets, riders, and no riders. They are split into five groups: full-face helmet, half-face helmet, no helmet, rider, and no rider. These labels teach a machine learning model how to tell the difference between different things in CCTV footage and put them into one of these groups. When the model is trained, the rider group represents a person driving a motorcycle, and the no-rider group means a motorcycle being driven without a person. A full-face helmet covers the whole face, while a half-face helmet only covers part of the face. Lastly, a person riding without a helmet is in the “no helmet” category. These labels are significant for training the model to correctly recognize and classify objects in CCTV footage, which can improve road safety and stop accidents.

Figure 6.10: Hyperparameter 3 Batch 0 Label

Batch 0 Predicted

The batch 0 predicted values list detections from the first set of data it looks at. Each detection includes information like the expected class, the level of confidence in the prediction, and the detection location in the video frame. The model also gives more information about the prediction, such as the size and shape ratio of the object that has been found. Drawing bounding boxes around the objects found in the input images or video frames shows the predicted values for the batch. When these visualizations are put on top of the original video, they show where the model found riders and helmets. The expected values can also be used to measure several performance metrics, such as accuracy, recall, and average accuracy, to determine how well the model did on the given task.

658 Muralla St., Intramuros, Manila, 1002, Philippines
Tel. No. : +63(2) 247-5000 local 5804
Telefax : +63(2) 891-0800
Web : www.mapua.edu.ph

Figure 6.11: Hyperparameter 3 Batch 0 Predicted

658 Muralla St., Intramuros, Manila, 1002, Philippines
Tel. No. : +63(2) 247-5000 local 5804
Telefax : +63(2) 891-0800
Web : www.mapua.edu.ph

MSE

The MSE value of 0.1076 indicates that the average squared difference between the predicted and actual values is 0.1076.

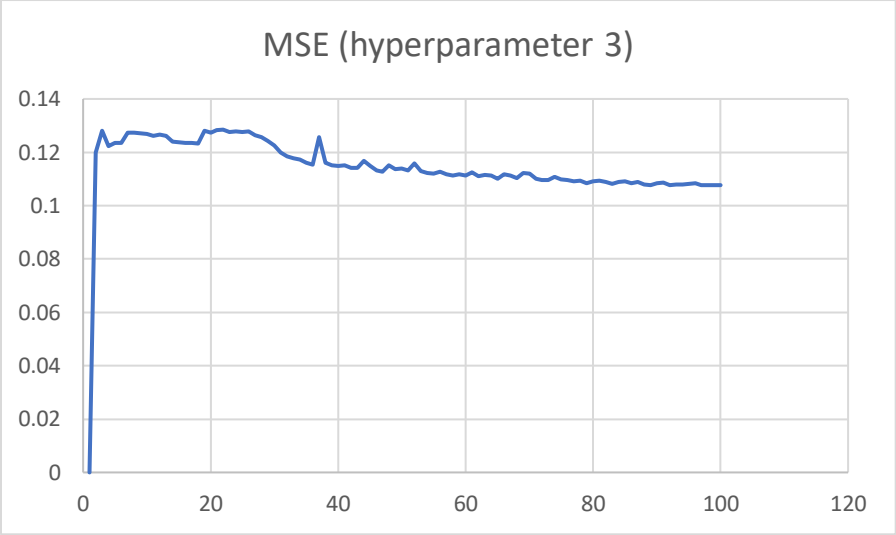


Figure 6.12: Hyperparameter 3 MSE

RMSE

The RMSE value of 0.3280 is the square root of the MSE value and represents the average difference between the predicted and actual values.

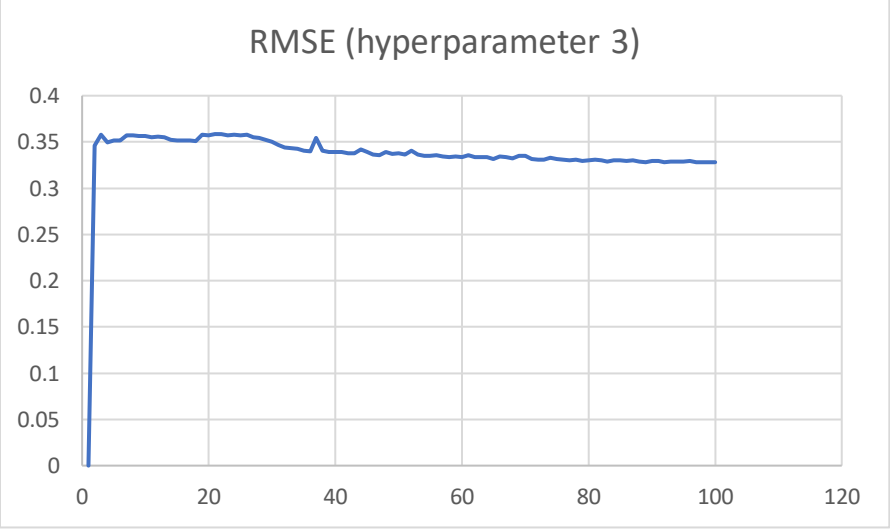


Figure 6.13: Hyperparameter 3 RMSE

658 Muralla St., Intramuros, Manila, 1002, Philippines
Tel. No. : +63(2) 247-5000 local 5804
Telefax : +63(2) 891-0800
Web : www.mapua.edu.ph

Confusion Matrix

From the confusion matrix obtained from applying the third set of hyperparameter settings, the general results are significantly different from the confusion matrices obtained using the first and second set of hyperparameter settings. The highest true positive value is obtained for the Rider class with a value of 0.65. The next true positive values obtained are much lower than the value obtained for the Rider class, which are for the Full-Face and Half-Face classes with a value of 0.06 and 0.04 respectively. The Invalid and No-Helmet classes which are annotated have all been predicted as background elements, therefore none of the annotated instances were detected by the model. Similar behavior is observed for the Full-Face and Half-Face classes, as the annotated instances are not detected by the model and are regarded as background elements of the pictures. The Rider class only has a value of 0.35 for the predicted background FN, which means that this proportion is the number of instances which have not been detected as a Rider class and have not been detected as any other class. Based on the background FP values, there are some unannotated Full-Face and Half-Face helmets which are detected by the model based on the predicted background FP values of the class (0.09 and 0.05).

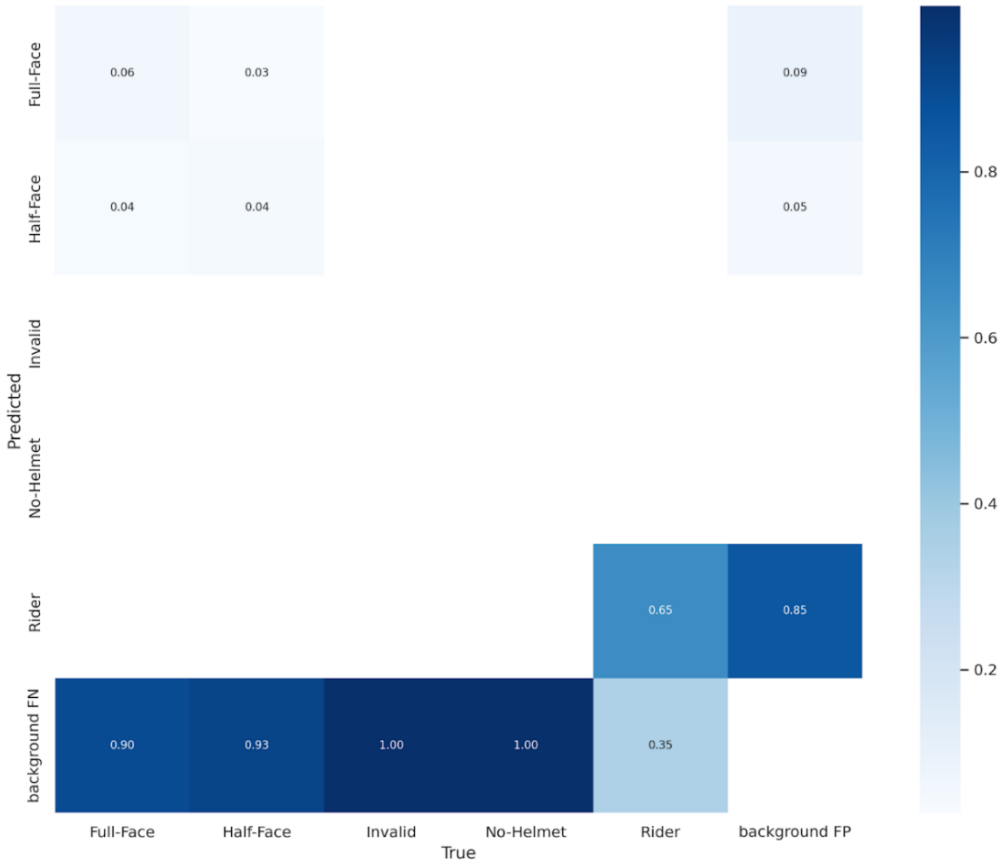


Figure 6.14: Hyperparameter 3 Confusion Matrix

658 Muralla St., Intramuros, Manila, 1002, Philippines
Tel. No. : +63(2) 247-5000 local 5804
Telefax : +63(2) 891-0800
Web : www.mapua.edu.ph

Model Output

The highest true positive value is observed for the Rider class, while the Invalid and No-Helmet classes are predicted as background elements. The analysis identifies unannotated Full-Face and Half-Face helmets that are detected by the model. Overall, the video output provides a comprehensive overview of the model's performance and highlights areas for improvement.

Video Output (DeepSORT)

After running the DeepSORT model using the YOLOv7 hyperparameter 3 settings, the group exported the video output of the DeepSORT tracker. The screenshot below shows a frame of the exported video

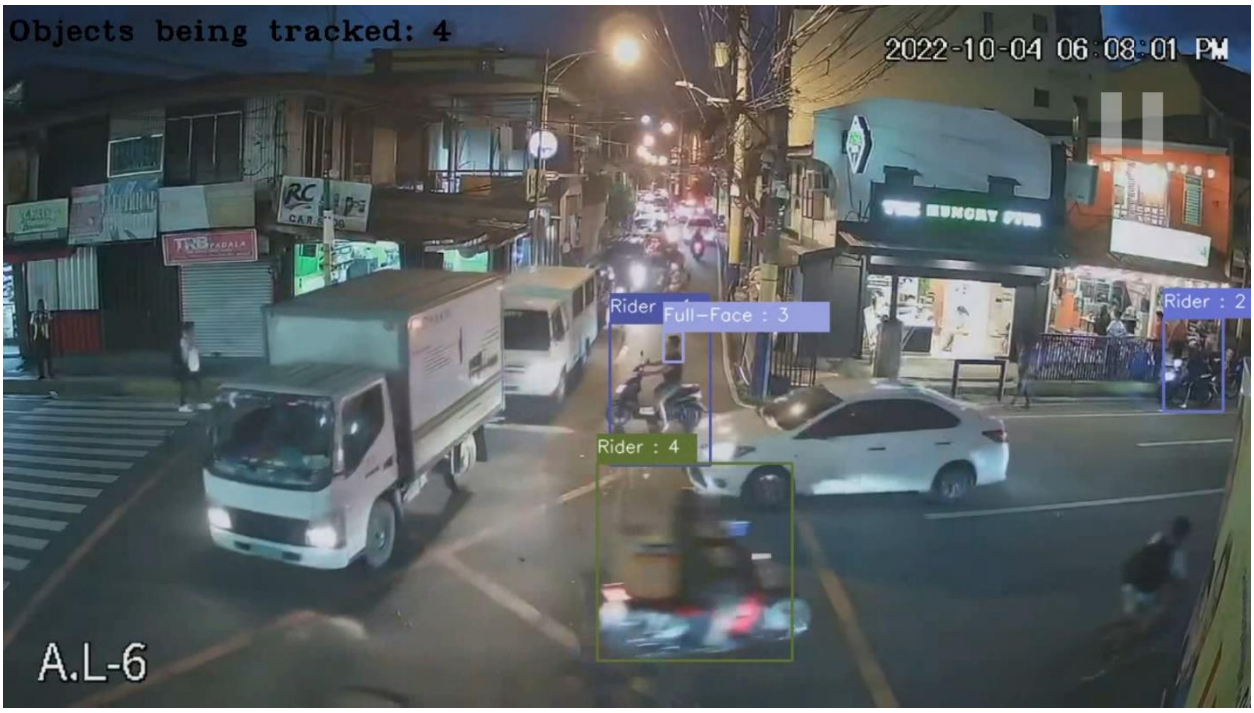


Figure 6.15: Hyperparameter 3 Video output DeepSORT

From this screenshot, we can see that the DeepSORT works because it was able to detect the objects in the video and as well as track unique objects with tracker ID.

658 Muralla St., Intramuros, Manila, 1002, Philippines
Tel. No. : +63(2) 247-5000 local 5804
Telefax : +63(2) 891-0800
Web : www.mapua.edu.ph

Conclusion and Comparison of Hyperparameters

To recap, the objective of this project is to apply different hyperparameter settings to improve the YOLOv7 model and evaluate the impact of the hyperparameter settings to its performance.

Observing the model evaluation as the different hyperparameter settings were applied, there was a decrease in model performance. The first set of configurations achieved a respectable performance with an F1 score of 0.47 at 0.298 level for all classes; an average precision of 0.448 at mAP@0.5 for all classes; an average of 0.968 precision confidence for all classes; and recall score of 0.88 at 0.000 threshold. The second set of configurations achieved a worse performance compared to the first settings while the last settings achieved the worst performance.

With this context in mind, it can be said that the first hyperparameter settings showed the best results for the model improvement. However, it can also be said that a possible reason for the decline of model performance as the second and third hyperparameter settings were applied could be overfitting. As observed, the "rider" class was severely overrepresented which may have resulted in the model becoming too specialized to the training data and performing poorly on new data. Likewise, since the second and third hyperparameter settings had training time of 75 and 100 epochs respectively, it may have attributed to the overfitting to the training data resulting in a decrease in performance.

Recommendation

The aim of this project is to apply the hyperparameter settings on the YOLOv7 model. Accordingly, based on the findings as well as the conclusion, it is highly recommended to apply cross-validation. Cross-validation is a technique used in evaluating the performance of a model on different subsets of the data. With this, cross-validation can help to detect overfitting and improve the generalization ability of the model.

The performance of the YOLOv7 model can be also affected by the quality of the training dataset. In particular, when the dataset contains low light images, YOLOv7 may not perform as well as expected. This is because the model heavily relies on the brightness and contrast of the input images to accurately detect objects. Therefore, to optimize the performance of the YOLOv7 model, researchers recommend using a higher quality dataset with different sets of data augmentations. By doing so, the model can learn to better adapt to different lighting conditions and improve its accuracy in object detection tasks.

Further, it is also recommended to apply further data augmentation in order to help the model generalize better to new and unseen data to prevent overfitting. Lastly, it is also recommended to balance the classes when annotating the data.

658 Muralla St., Intramuros, Manila, 1002, Philippines
Tel. No. : +63(2) 247-5000 local 5804
Telefax : +63(2) 891-0800
Web : www.mapua.edu.ph

Related Literature

Yolov7

YOLOv7: Trainable Bag-of-Freebies Sets New State-of-the-Art for Real-Time Object

[1] Since it is frequently required in computer vision systems like multi-object tracking, autonomous driving, robotics, and medical image analysis, real-time object identification is a significant area of research in the field. In addition to different neural processing units (NPUs) created by major manufacturers, such as the Apple Neural Engine, the Intel Neural Compute Stick, the Nvidia Jetson AI Edge Devices, the Google Edge TPU, the Qualcomm Neural Processing Engine, the MediaTek AI Processing Unit, and the Kneron AI SoCs, the computing devices that perform real-time object detection are typically mobile CPUs or GPUs. These edge devices concentrate on accelerating particular processes such MLP operations, depthwise convolution, and vanilla convolution. This work proposes a real-time object detector to enable mobile GPU and GPU devices from edge to cloud. On GPU V100, YOLOv7 has the highest accuracy of 56.8% AP of all real-time object detectors with 30 FPS or more, outperforming all other known object detectors in the range of 5 FPS to 160 FPS. Both the transformer-based detector SWINL Cascade-Mask R-CNN and the convolutional-based detector ConvNeXt-XL Cascade-Mask R-CNN are outperformed by the YOLOv7-E6 object detector, which performs better by 509% in speed and 2% in accuracy. Additionally, YOLOv7 beats YOLOR, YOLOX, and Scal. The authors only utilized the MS COCO dataset to train YOLOv7; no additional datasets or pre-trained weights were employed.

Autonomous Movement of Wheelchair by Cameras and YOLOv7

[2] Based on the World Health Organization, 15% of the global population has a physical disability. In the United States, 61 million people are currently living with a disability, meaning that one in four adults in the US struggles with a disability. People with disabilities often experience feelings of sadness, a decrease in motivation, and a loss of independence. To address this, various mobility-assisting products, such as wheelchairs, have been developed. The researchers' study aimed to restore autonomy to individuals with disabilities by allowing them to travel independently without the need for assistance from others. The study also discussed the use of cameras in vehicles for purposes such as parking and lane detection, and the use of LiDAR and camera technology in wheelchair navigation. Additionally, researchers have found that the convolution neural network (CNN) and YOLOv5 algorithm are particularly useful for object detection due to their fast convergence, high accuracy, and ability to be customized. YOLOv5 is also known for its strong real-time processing capabilities and low hardware requirements, making it well-suited for use on embedded devices. This publication discusses the development of a smart wheelchair that uses cutting-edge octascopic vision technology to provide a person with limited mobility greater independence. The researchers customized a manually operated wheelchair by adding two monochromic camera arrays mounted around the wheelchair's frame to provide a 360-degree vision. The autonomous wheelchair is also relatively affordable. The main goal of the research was to create a wheelchair that could move around independently within a building, with or without human assistance. The goal was also to make automated wheelchairs more accessible to people who previously couldn't afford them. The researchers collected a large number of otoscopic images during testing and used a Yolov7-based object recognition model to enable the wheelchair to move independently. The study also covers the camera positioning and obstacle detection model that uses octascopic images. All the project design files have an open-source license and can be replicated openly.

658 Muralla St., Intramuros, Manila, 1002, Philippines
Tel. No. : +63(2) 247-5000 local 5804
Telefax : +63(2) 891-0800
Web : www.mapua.edu.ph

A Real-time Kiwi Fruit Detection Based on Improved YOLOv7

[3] The kiwi fruit is one of the most well-known products from New Zealand and is sold globally. However, the industry is facing challenges, such as labor shortages, which have led to losses. To increase productivity in the kiwi fruit sector, one solution is to improve the supply chain for picking, sorting, cleaning, and packing the fruit. To increase the yield of kiwi fruit, improve picking efficiency, and reduce labor costs, the researchers used advanced artificial intelligence, specifically deep learning. The core of kiwi fruit counting is to have fast and accurate models. Historically, fruit identification algorithms used digital image processing to extract key feature values from visual objects, such as color and shape. These algorithms used techniques for image segmentation to detect the visual objects. However, these traditional algorithms are not very reliable. Factors such as fruit position and lighting conditions can affect the model's detection results, making it difficult to estimate fruit yields in orchards under natural conditions. Historically, the yield of kiwi fruit was heavily dependent on human labor as the fruit had to be physically picked by hand and relied heavily on human resources. However, the industry has greatly benefited from agricultural automation due to the rapid advancement of deep learning in this field. Accurately and quickly detecting kiwi fruit can help the industry to move more efficiently. The authors of this research propose an improved kiwi fruit detection algorithm based on YOLOv7. They created a dataset of digital photos of kiwi fruit from real kiwi orchards that were manually tagged and data-augmented. The authors adjusted the weight of visual characteristics and reduced the weight of invalid features in YOLOv7 and added an attention module. The results show that their proposed technique offers faster detection speed suitable for real-time use and greater detection accuracy than the original YOLOv7 model.

Smart Baby Monitoring System using YOLO V7 Algorithm

The paper [7] proposes a novel approach based on an IoT and deep learning framework for baby monitoring. By implementing a YOLO detection model in an IoT device, a baby monitor camera will focus and adjust its movement according to the baby's detected position. The dataset for creating the YOLO-based models was taken directly from Kaggle and Google images. The dataset included images of babies in various settings and environments. Using a Roboflow framework, the researchers labeled the dataset manually. Two models were implemented in this study: YOLOv4-tiny and YOLOv7 models. Both YOLOv4-tiny and YOLOv7 models were trained with the same dataset. On the other hand, a Librosa library was implemented in order to process and classify audio detected by the IoT device. The results have shown that both models were able to make quick and accurate predictions. The effectiveness of the system is evaluated using the mean Average Precision (mAP) metric, with a higher mAP score indicating better detection capabilities. The YOLOv4-tiny model had an mAP of 36.02% with an inference time of 17 milliseconds, and the YOLOv7 model had an mAP of 67.4% with an inference time of 2.8 milliseconds.

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

The paper [8] assesses the performance of different deep learning algorithms in detecting personal protective equipment (PPE) in construction sites, specifically safety helmets. The researchers of the study compared the performance of three YOLO-based object detection models: YOLOv5, YOLOv6, and YOLOv7. Using a Kaggle dataset consisting of 5,000 images of safety helmets used in construction sites, the YOLO-based models were trained and evaluated. All three models were tested to detect construction helmets in various settings, environments, and individuals. Once trained and tested, the researchers concluded that the YOLOv7 model was more effective in detecting safety helmets since both YOLOv5 and

YOLOv6 models struggled to detect helmets on individuals with darker skin tones. Future studies may further enhance the efficiency and accuracy of the construction safety helmet detection YOLOv7 model.

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

The article [9] describes a system for identifying vehicles in a camera network that utilizes YOLOv7 for identifying vehicles in each camera, Re-ID for distinct characteristics of the bounding boxes, DeepSORT to monitor the vehicle's identity across multiple cameras, and Trajectory Clustering. The CityFlow dataset, comprising over 3 hours of synchronized city intersection videos, was employed to train the YOLOv7 model and assess its accuracy and effectiveness. The research indicates that the framework has the potential for use in large-scale systems. The Multi-Target Multi-Camera Tracking (MTMC) framework pipeline is outlined in the paper, and it is demonstrated that the framework is efficient and accurate in detecting vehicles as compared to other YOLO-based models used to test the framework.

YOLObin: Non-Decomposable Garbage Identification and Classification Based on YOLOv7 (Jabed, R. & Shamsuzzaman, M)

[10] Researchers Jabed and Shamsuzzaman propose a method of garbage classification and identification which employs YOLOv7.

The performance of their model is evaluated against two other object detectors. The results show that Mask-RCNN had an f-measurement of 85%, YOLOv5 had an f-measurement of 95.1%, and YOLOv7 had an f-measurement of 95.9%. Furthermore, the researchers used non-decomposable multiclass images of garbage as their dataset. Likewise, four classes of non-decomposable garbage data were appropriately named chips packet, plastic bottle, and polythene. Their data collection involved preparing collected data from campus areas in Begum Rokeya University, Rangpur, Bangladesh. Proceeding this is the data augmentation where they translated a few pixels randomly either by turning the orientation of images to horizontal or vertical. Their experimentation on model comparison showed that their model performed well in classifying images of garbage that have cluttered backgrounds. Afterwards, they performed evaluation of test results between the models. Their findings indicate that the classification framework has a sensible degree of accuracy and segmentation recognition performance is significantly better for point-by-point images, which can efficiently and effectively complete the task of garbage classification.

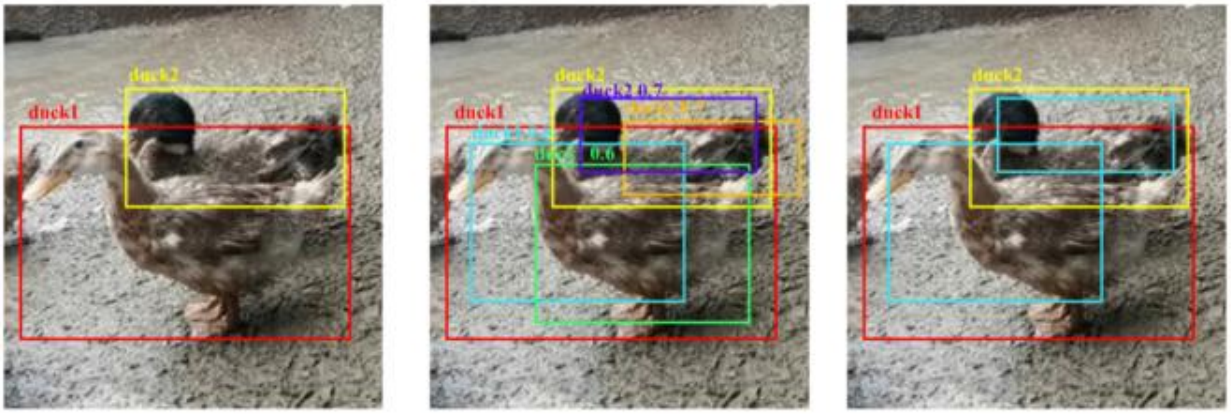
Moreover, their result analysis shows sufficient graphs of loss function in training of the different models. Among those, YOLOv7 performed the best wherein it achieved higher *F* score 1 (0.959 or 95.9%). This means that YOLOv7 can be a suitable choice of model to perform garbage classification in the future for smart garbage management solutions. A good point on the paper was the inclusion of Ground Truth of their data.

An Attention Mechanism-Improved YOLOv7 Object Detection Algorithm for Hemp Duck Count Estimation (Jiang, K., et al)

[11] In this paper, researchers Jiang, et al use deep learning algorithms to address the inefficient and costly labor of counting in the hemp duck breeding industry. They also aim to achieve a better development of the intelligent farming industry. The researchers used a DJI Pocket 2 camera to capture domestic ducks in China. Their dataset was collected in the waterfowl farm located in Ya'an, Sichuan Province in China. Moreover, their dataset was around 1500 images which meant they split 1300 images

658 Muralla St., Intramuros, Manila, 1002, Philippines
Tel. No. : +63(2) 247-5000 local 5804
Telefax : +63(2) 891-0800
Web : www.mapua.edu.ph

for the training set while 200 images for the test set. Since labeling the entire hemp duck body proved to be a challenge as it led to many overlapping labeling boxes, which negatively impacted the accuracy of counting individual hemp ducks, the researchers decided to label only the head of the hemp ducks and compared the results of the two methods. Shown below is figure 1 where it depicts the ground truth boxes of hemp ducks (a); the output prediction boxes of the simulated network for the two hemp ducks (b); and the effect of removing the redundant prediction boxes after non-maximum suppression (c).



(a) (b) (c)

In the data pre-processing phase of their project, the researchers used an unconventional data enhancement method called mixup data augmentation wherein it is built on a data-independent data enhancement principle that employs linear interpolation to construct new training samples and labels. Following this, the parameters of the model training are the following:

- Batch size: 16
- Image size: 640x640
- Learning Rate: 0.01
- Weight Decay: 0.0005
- Momentum: 0.937
- Epochs: 300

The researchers used precision, recall, mean average precision, F1 score and frames per second for the evaluation metrics of the performance of the YOLO algorithms. In critiquing the paper, one of the good points I found was including the ground truth data for the hemp ducks, output prediction boxes of the simulated network and the redundant prediction boxes as it gave me an idea what to do when you encounter many overlapping labeling boxes when annotating your dataset.

A Multiscale Lightweight and Efficient Model Based on YOLOv7: Applied to Citrus Orchard (Chen, J., et al)

[12] Just like the previous literature, his paper also aims to provide a deep learning based solution to solve the human labor of annual citrus production. Researchers Chen, et al argued that traditional object detection methods frequently fall short of providing optimal results across all aspects of citrus picking technology. To address this, the researchers propose a modified YOLOv network model where it incorporates a small detection layer, lightweight convolution, and a CBAM (Convolutional Block Attention Module) attention mechanism, enabling multi-scale feature extraction and fusion while reducing the

658 Muralla St., Intramuros, Manila, 1002, Philippines
Tel. No. : +63(2) 247-5000 local 5804
Telefax : +63(2) 891-0800
Web : www.mapua.edu.ph

number of parameters of the model. The dataset was produced by the researchers themselves. They captured images of citrus fruits using three (3) Huawei NOVA7 mobile phones equipped with a high-definition rear camera. The researchers kept in mind to take photos of the dataset in November of 2021 as it was considered an ideal time for citrus harvest. Moreover, the researchers also took into account the natural daylight conditions for the images. The location of the data acquisition was in Danling Country, Meishan, City in Sichuan Province in China. In summary, the dataset comprises four distinct forms of interference, specifically, overlapping, occlusion, variations in lighting and darkness, and variations in distance. There were a total of 1266 images in their dataset. The ratio for the training, test set and validation set was set to a ratio of 80/10/10.

Dataset preprocessing involved dividing the data into four parts. As such, when images couldn't be recognized as a citrus fruit, the image was extracted without labeling. Similarly, citrus fruits with an occlusion degree of more than 90% and less than 5 pixels were not labeled. A fifth member of the research team reviewed the labeled dataset, and simultaneously, the five research members collaborated to label the unlabeled images through a majority vote. The name of the software used for labeling was LabelImg, and the labeling box was in the form of a rectangle, with the label name being set in Mandarin. As a result, the corresponding XML label file was generated, and the dataset was ultimately constructed according to the COCO dataset standards.

The results and discussions section of the paper included graphs of the performance and accuracy of each of the five object detectors namely Citrus-YOLOv7 (their model), YOLOv7, SSD, Faster R-CNN, and RetinaNet. A quantitative analysis of mean average precision was carried out—Citrus-YOLOv7 achieved an accuracy of 97.29% which is 2.65% higher than YOLOv7 baseline model where it scored 94.64% accuracy. A table of the performance of the models is shown below.

Models	$mAP_{@0.5}$	$mAP_{@[0.5:0.95]}$	Precision	Recall	F1
Faster R-CNN	82.93%	57.25%	69.34%	90.24%	78.42%
RetinaNet	93.67%	67.90%	90.77%	92.56%	91.66%
SSD	92.81%	65.46%	86.12%	89.50%	87.78%
YOLOv7	94.64%	69.19%	93.69%	91.41%	92.54%
Citrus-YOLOv7	97.29%	74.83%	94.25%	93.37%	93.81%

Though the Citrus-YOLOv7 model performed very well, the researchers believe there are still three main improvements methods for their model namely the following:

- Replace traditional convolution module with lightweight GhostConv
- Add small target detection layer to be able to achieve multi-scale feature fusion
- Improve the CBAM module by taking into account channel and spatial dimensions

658 Muralla St., Intramuros, Manila, 1002, Philippines
Tel. No. : +63(2) 247-5000 local 5804
Telefax : +63(2) 891-0800
Web : www.mapua.edu.ph

DeepSort

Multi-target Tracking Based on Deep Sort in Traffic Scene

[4] The term "multi-object tracking" (MOT) refers to a technique for locating and identifying moving objects in a sequence of photographs. The procedure entails examining a series of photos to find any moving things that may be there. Since individuals are a frequent object in many real-world contexts, this technique has been frequently used to monitor people. It may, however, also be used to track other items, such vehicles or animals. In this work, the scientists tracked both people and automobiles in traffic scenes using the DeepSort multi-target tracking algorithm. They utilized YOLOv4 to train models to recognize pedestrians and automobiles in traffic scenes, and they then performed multi-target tracking using the predictions from these models. The researchers compared YOLOv4 and DeepSort to increase tracking performance stability while decreasing ID switching and increasing tracking accuracy. The process of estimating the motion of a target and associating data in target tracking relies solely on the position and size of the detection frame and lacks a method for re-identifying the target. In this study, the researchers did not take into account any visual characteristics of the target being monitored. As a result, when the target is lost, it cannot be found again and can only be updated through detection, which is not a conventional tracking method and needs to be improved. The primary focus of this research is on achieving high speed rather than emphasizing on precision and robust error detection.

Vehicle Tracking Using Deep SORT with Low Confidence Track Filtering

[5] Recent advancements in machine learning have enabled the use of modern tracking-by-detection techniques for the multi-object tracking problem, which is the process of detecting and tracking multiple objects in a video stream. These techniques have become increasingly popular in many fields, such as surveillance, autonomous vehicles, and robotics, due to their ability to accurately and efficiently track objects in complex environments. These algorithms consist of two main components: an object detection algorithm that produces detection results, typically in the form of bounding box coordinates, for each frame, and a data association algorithm that determines if the newly detected object can be linked to an existing track. The object detection algorithm is responsible for identifying the presence and location of objects in the video stream, while the data association algorithm is responsible for linking the detected objects to their corresponding tracks over time. The combination of these two components allows for the accurate and efficient tracking of multiple objects in a video stream, which is essential for many real-world applications. The large range of applications for multi-object tracking (MOT) in traffic monitoring and video surveillance makes the issue appealing. The tracking-by-detection approach has been the focus of recent advancements in MOT. Modern tracking-by-detection algorithms are nevertheless plagued by problems like a high number of false positive tracks since computer vision is a rather complex and interconnected field. In this study, the researchers suggested integrating a low confidence track filtering into the Simple Online and Realtime Tracking with a DeepSort (Deep SORT) algorithm to lessen the impact of erroneous detection systems on vehicle tracking. The researchers provided a dataset for UA-DETRAC vehicle re-identification that they had created themselves. This dataset may be utilized to train DeepSORT's convolutional neural network for data association. On the UA-DETRAC test dataset, the researchers assessed our suggested tracker.

658 Muralla St., Intramuros, Manila, 1002, Philippines
Tel. No. : +63(2) 247-5000 local 5804
Telefax : +63(2) 891-0800
Web : www.mapua.edu.ph

Multi Object Tracking with UAVs using Deep SORT and YOLOv3 RetinaNet Detection Framework

[6] Object tracking is becoming more widely used as a result of ongoing research in machine learning and convolutional neural networks. Applications include detecting potholes, counting people to generate crowd statistics, recognizing license plates, and facial recognition at security checkpoints. Object tracking is also used in unmanned aerial vehicles for search-and-rescue, surveillance, and reconnaissance. Most research in this area focuses on detecting cars, people, and infrastructure elements. Multiple object tracking is the process of determining the movement of multiple objects over time in a video. This can be done in real-time (online) or after the video has been recorded (offline). Online tracking is useful for real-time applications as it requires less computational power compared to more complex algorithms because it only uses detections from the current and previous frames. Object tracking and detection is a crucial component of UAV applications, such as surveillance and reconnaissance, which has grown in importance over time. The researchers have proposed a tracking-by-detection method for real-time multiple object tracking (MOT) using video from a drone-mounted camera. This method, called Deep SORT, is efficient and has enhanced detection techniques. The authors combined YOLOv3 with RetinaNet to provide detections for each frame to overcome the challenges of tracking objects from a high altitude. This approach is the dominant paradigm in the field.

Video object tracking based on YOLOv7 and DeepSORT (Yang, F., et al)

[13] Researchers Yang and others propose YOLOv7-DeepSORT wherein they experiment, evaluate and compare previous YOLO iterations integrated DeepSORT with their own YOLOv7-DeepSORT. Yang et al believes that YOLOv7 fairs excellently well in object detection tasks. They refer to YOLOv5-DeepSORT but replace YOLOv7 with the object detection model of the aforementioned network to obtain YOLOv7 and DeepSORT integration. In evaluating the model, the researchers evaluated the performance of YOLOv7-DeepSORT on the 02, 04, 05, 09, 10, 11, 13 sequences of MOT16 which they then compared it with YOLOv5-DeepSORT. Further, the researchers left the parameters for both models to be the same and among them YOLOv5, YOLOv5s, YOLOv5m, YOLOv5l, and YOLOv7 were used in the object detection part all of which uses the pre-trained model.

The evaluation metrics for the experimentation are the following:

- Multi-Object Tracking Accuracy (MOTA)
- Multi-Object Tracking Precision (MOTP
- ID F1 Score (IDF1)
- Identity Switches (IDs)
- Mostly Lost Targets (ML)
- Mostly Tracked Targets (MT)
- False Positives (FP)
- False Negatives (FN)

658 Muralla St., Intramuros, Manila, 1002, Philippines
Tel. No. : +63(2) 247-5000 local 5804
Telefax : +63(2) 891-0800
Web : www.mapua.edu.ph

Sort and Deep-SORT Based Multi-Object Tracking for Mobile Robotics: Evaluation with New Data Association Metrics (Pereira, R., et al)

[14] In this paper, researchers Pereira et al, evaluate two MOTs utilizing detection algorithms, SORT and DeepSORT with a focus on navigation tasks for assistive mobile robots. To improve the data association of both methods of multi-object tracking, which are addressed as a linear problem with a generated cost matrix, a set of new object tracking data association cost matrices based on intersection over union, Euclidean distances, and bounding box metrics are proposed by the researchers.

In evaluating the Multi-object tracking (MOT) methods by detection in a real-time pipeline, the YOLOv3 is employed to detect and classify objects in images. Further, to execute the proposed evaluation with a focus on assistive platforms, the ISR Tracking dataset is utilized by the researchers. Subsequently, other experimental evaluations were carried out on the MOT17 dataset. Performing those evaluations yielded positive results for the researchers' object tracking data association cost matrices which meant there was improvement in the majority of MOT evaluation metrics. In the same vein, the pipeline composed of the detector and the tracking module reached a positive threshold frame rate value.

Research on Pedestrian Detection and DeepSORT Tracking in Front of Intelligent Vehicle Based on Deep Learning (Chen, X., et al)

[15] Researchers Chen et al propose a smart deep learning based system for improved tracking and detecting of pedestrian targets for intelligent vehicles. Their system will aim to address the issue of tracking failures that are caused by small-target pedestrians and the challenge of detecting pedestrians in crowds and in complex environments. The researchers followed a six phased summary of their improved YOLO Network Design based on Deep Learning.

1. Model building
2. Model parameter setting
3. Data preprocessing
4. Model training
5. Model testing
6. Model evaluation

Their model building was categorized into two modules namely Darknet-53 and detection module (multi-scale pedestrian prediction). Their model parameters are set to comply for intelligent driving scenarios as well as with small targets when detecting in real-time. Data preprocessing involved acquisition of footage by a camera then driving records were collected by vehicle cameras then lastly, single frame images were obtained by frame segmentation. For the images of pedestrians, they were taken from public data sets (named PD-2022).

Evaluation was divided into two—Pedestrian Detection and Pedestrian Tracking evaluation indicators. During the pedestrian detection tasks, the researchers took into account the mean average accuracy as the evaluation indicator. On the other hand, in pedestrian tracking situations, the researchers chose MOTP and MOTA as performance evaluation indicators. Statistical results of pedestrian tracking evaluation indicators can be seen below.

Model	MOTA	MOTP	MT	ML	FM	IDSW	FPS
YOLO+DeepSort	49	60.11	24	22.6	986	611	36
Improved YOLO+DeepSort	53.8	62.12	28.2	21.8	241	120	33

658 Muralla St., Intramuros, Manila, 1002, Philippines
Tel. No. : +63(2) 247-5000 local 5804
Telefax : +63(2) 891-0800
Web : www.mapua.edu.ph

Lastly, the improved YOLO pedestrian detection model and the DeepSORT pedestrian tracking were both evaluated and verified in a similar experimental environment. Results show that the researchers' improved model can provide a robust detection accuracy for small-target pedestrians. It is also able to efficiently handle the challenges of target occlusion and the reduction of the rate of missed detections such as false detection of the targets.

658 Muralla St., Intramuros, Manila, 1002, Philippines
Tel. No. : +63(2) 247-5000 local 5804
Telefax : +63(2) 891-0800
Web : www.mapua.edu.ph

Dataset Link

https://universe.roboflow.com/new-dataset-67/new-dataset-6-7

Google Collab Links

YOLOv7 Training (Hyperparameter 1)

https://colab.research.google.com/drive/1YpGWa7rLg00xteakkQ2qCqj4mTOSQziJ?usp=sharing

YOLOv7 Training (Hyperparameter 2)

https://colab.research.google.com/drive/1i8d9eDPfJcP-itotWJqtICdlwgoCIfJC?usp=sharing

YOLOv7 Training (Hyperparameter 3)

https://colab.research.google.com/drive/1BJN8PELSohEprxONuBKqr1UKCCyg798n?usp=sharing#scrollTo=7ld8Of1FQmYz

YOLOv7 + DeepSORT (Hyperparameter 1)

https://colab.research.google.com/drive/1RbCFKeINk59Vm9jEbx3QlwQyTnaRJl-m?usp=sharing

YOLOv7 + DeepSORT (Hyperparameter 2)

https://colab.research.google.com/drive/1bzQqz5jRaliSSzA9nD1NHvPF7YutzQYv?usp=sharing

YOLOv7 + DeepSORT (Hyperparameter 3)

https://colab.research.google.com/drive/1FGjNB2CHDF4sOFV3ysDodRSgQodQr0Gh?usp=sharing

References

[1] C.-Y. Wang, A. Bochkovskiy, and H.-Y. M. Liao, "Yolov7: Trainable bag-of-freebies sets new state-of-the-art for real-time object detectors," arXiv.org, 06-Jul-2022. [Online]. Available: https://arxiv.org/abs/2207.02696. [Accessed: 14-Apr-2023].

[2] A. B. Sarker, E. Sola-Thomas, C. Jamieson, and M. H. Imtiaz, "Autonomous Movement of Wheelchair by Cameras and YOLOv7," 09-Dec-2022. [Online]. Available: https://sciforum.net/manuscripts/13834/manuscript.pdf. [Accessed: 14-Apr-2023].

[3] Xia, Y., Nguyen, M., & Yan, W. Q. (2021). *A real-time kiwifruit detection based on improved Yolov7*. A Real-time Kiwifruit Detection Based on Improved YOLOv7. https://cerv.aut.ac.nz/wp-content/uploads/2022/11/8544_Xia-Yi.pdf

[4] Duan, C., & Li, X. (2021). Multi-target Tracking Based on Deep Sort in Traffic Scene. Journal of Physics: Conference Series, 1952(2), 022074. doi:10.1088/1742-6596/1952/2/022074

[5] Xinyu Hou, Yi Wang, & Lap-Pui Chau. 2019. Vehicle Tracking Using Deep SORT with Low Confidence Track Filtering. IEEE. DOI: 10.1109/AVSS.2019.8909903

[6] Kapania, S., Saini, D., Goyal, S., Thakur, N., Jain, R., & Nagrath, P. (2020). Multi Object Tracking with UAVs using Deep SORT and YOLOv3 RetinaNet Detection Framework. Proceedings of the 1st ACM Workshop on Autonomous and Intelligent Mobile Systems. doi:10.1145/3377283.3377284

[7] Nandhakumar R. G and Mohanapriya S. 2022. Smart Baby Monitoring System using yolo $\mathrm{v}7$ algorithm. *2022 International Conference on Information Technology Research and Innovation (ICITRI)* (2022). DOI:http://dx.doi.org/10.1109/icitri56423.2022.9970217

[8] Nigel Dale Yung, W.K. Wong, Filbert H. Juwono, and Zee Ang Sim. 2022. 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)* (2022). DOI:http://dx.doi.org/10.1109/gecost55694.2022.10010490

[9] Duong Nguyen-Ngoc Tran, Long Hoang Pham, Huy-Hung Nguyen, and Jae Wook Jeon. 2022. City-scale multi-camera vehicle tracking of vehicles based on Yolov7. ***2022 IEEE International Conference on Consumer Electronics-Asia (ICCE-Asia)* (2022). DOI:http://dx.doi.org/10.1109/icce-asia57006.2022.9954809**

[10] Jabed, M. and Shamsuzzaman, M. (2022) YOLObin: Non-Decomposable Garbage Identification and Classification Based on YOLOv7. *Journal of Computer and Communications*, 10, 104-121. doi: 10.4236/jcc.2022.1010008.

[11] Jiang K, Xie T, Yan R, Wen X, Li D, Jiang H, Jiang N, Feng L, Duan X, Wang J. An Attention Mechanism-Improved YOLOv7 Object Detection Algorithm for Hemp Duck Count Estimation. Agriculture. 2022; 12(10):1659. https://doi.org/10.3390/agriculture12101659

[12] Chen J, Liu H, Zhang Y, Zhang D, Ouyang H, Chen X. A Multiscale Lightweight and Efficient Model Based on YOLOv7: Applied to Citrus Orchard. Plants. 2022; 11(23):3260. https://doi.org/10.3390/plants11233260

[13] Yang, F., Zhang, X., & Liu, B. (2022). Video object tracking based on YOLOv7 and DeepSORT. ArXiv, abs/2207.12202.

[14] Pereira R, Carvalho G, Garrote L, Nunes UJ. Sort and Deep-SORT Based Multi-Object Tracking for Mobile Robotics: Evaluation with New Data Association Metrics. Applied Sciences. 2022; 12(3):1319. https://doi.org/10.3390/app12031319

658 Muralla St., Intramuros, Manila, 1002, Philippines
Tel. No. : +63(2) 247-5000 local 5804
Telefax : +63(2) 891-0800
Web : www.mapua.edu.ph

[15] Chen X, Jia Y, Tong X, Li Z. Research on Pedestrian Detection and DeepSort Tracking in Front of Intelligent Vehicle Based on Deep Learning. Sustainability. 2022; 14(15):9281. https://doi.org/10.3390/su14159281