T(7-8) Large Scale **CS174 | BM2**

Gil, Claire Francheska & Mercado, Rosselle Mae | Group 17

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

M2 – SA Chapter Assignment

For this assessment, the researchers were tasked to apply the following hyperparameter settings for improvements based on the previous models that were created (Motorcycle Rider Helmet detection). The researchers were assigned to the **YOLOv5 large** scale for the **time frame 7:00 pm – 8:00 pm**.

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

I. Cloning the Github Repository, Installing Dependencies, and Importing Libraries

The first step the researchers worked on was cloning the Github repository to use the requirements needed in this assessment. The researchers also installed dependencies by typing "install - qr requirements.txt" and roboflow by typing "install -q roboflow" to access the dataset from Roboflow. We also imported three libraries namely, "torch", "os", and "IPython.display". For IPython.display, the researchers specifically imported "Image", to display clear images.

```
!git clone https://github.com/ultralytics/yolov5  # clone repo
%cd yolov5
%pip install -qr requirements.txt # install dependencies
%pip install -q roboflow # install roboflow

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

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

II. Setting Up the Environment

In setting up the environment, the researchers created a folder called "datasets", where the directory of the entire project will be located. Based on the image below, we can see that we typed "/content/datasets" to specify where the dataset will be placed as it will be utilized by the researchers.

```
# set up environment
os.environ["DATASET_DIRECTORY"] = "/content/datasets"
```

III. Importing the Dataset from Roboflow

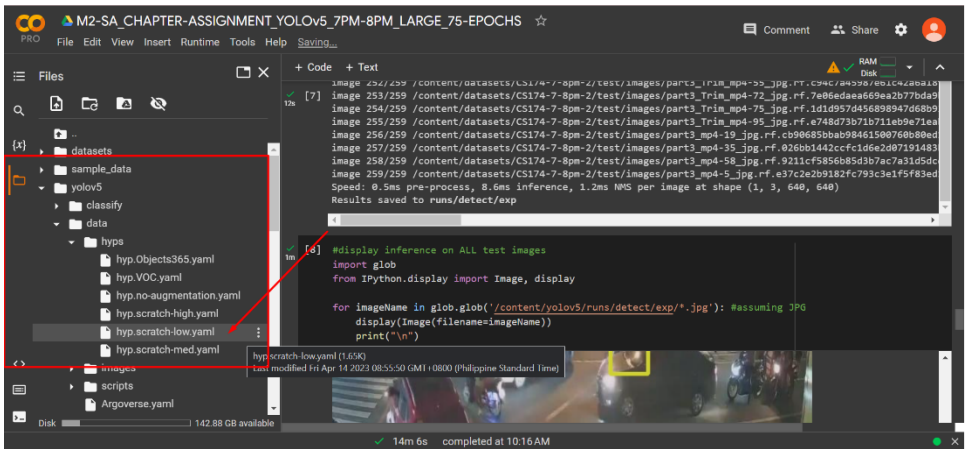
After setting up the environment, the researchers imported the augmented 7:00PM-8:00PM dataset from Roboflow by typing the code found in the image below. The following code includes the dataset from Roboflow along with the required API key to access the dataset the researchers will be utilizing.

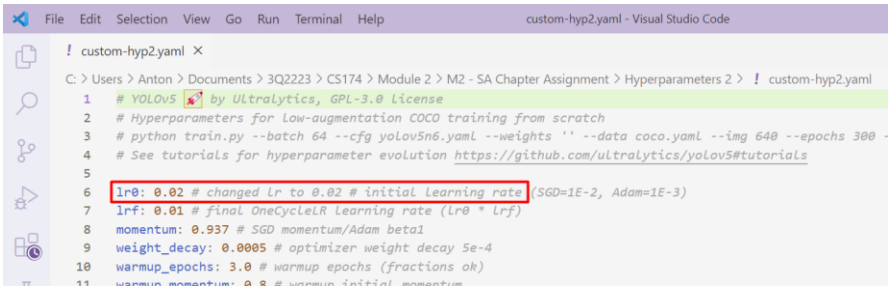
```
!pip install roboflow

from roboflow import Roboflow
rf = Roboflow(api_key="[redacted]")
project = rf.workspace("cs174-t78").project("cs174-7-8pm")
dataset = project.version(2).download("yolov5")
```

IV. Creating the Custom .yaml file

Before the hyperparameter settings are applied, we created a custom .yaml file that corresponded to the specified learning rates of Hyperparameters 2 and 3. (The corresponding file of Hyperparameter 1 did not need to be modified since its assigned learning rate was the default learning rate of the hyp.scratch-low.yaml file.) To create this custom file, we edited the contents of the original .yaml file and changed the value of the initial learning rate to the assigned value of each hyperparameter setting. For example, to change the learning rate for Hyperparameter 2, we changed the initial learning rate from the original file to 0.02 and saved it as a new file called custom-hyp2.yaml. The same process was applied to Hyperparameter 3.





After creating this custom configuration, we uploaded the files to their respective directories using a cloud-based environment, Google Colab.

```
from google.colab import files

uploaded = files.upload()
uploaded
```

Uploading the hyperparameters to a cell within the cloud environment allows us to easily access the configuration from within our code without manually entering them every time we run the model. This allowed us to streamline the training process and experiment with different hyperparameter settings to find the best combination for our specific task or dataset.

Utilizing custom hyperparameter configurations can improve our machine-learning model's performance by fine-tuning the various settings to fit our specific needs better. This is particularly important when working with complex or large-scale datasets, where even small changes to the hyperparameters can significantly impact the model's accuracy and performance.

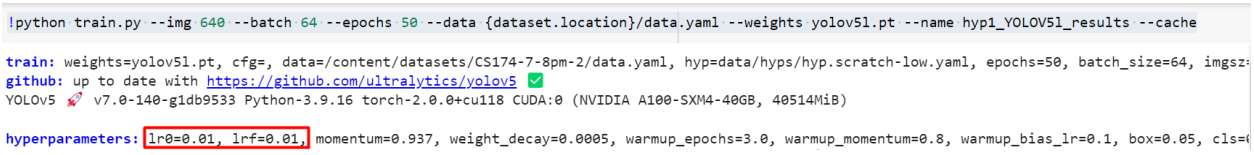
V. Training the Model using Modified Hyperparameters

Using the recommended hyperparameter settings, we modified the learning rate, batch size, epochs, and image size per setting. Since adjusting the learning rate had already been discussed in the previous section, adjusting only the batch sizes, epochs, and image sizes will be accomplished in the training phase of the project. The corresponding hyperparameters were adjusted, as follows:

***Hyperparameter 1:**

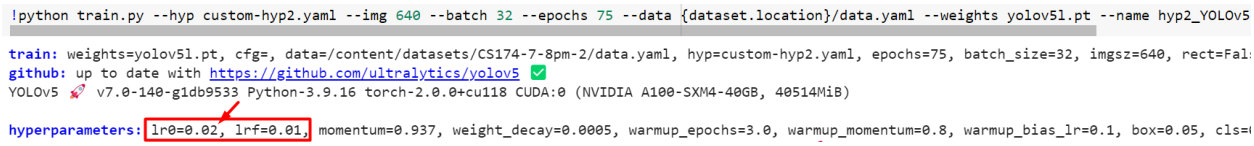
```
!python train.py --img 640 --batch 64 --epochs 50 --
data {dataset.location}/data.yaml --weights yolov5l.pt --
name hyp1_YOLOV5l_results --cache
```

*A custom .yaml file was not included for the first hyperparameter since the default learning rate is set at 0.01.



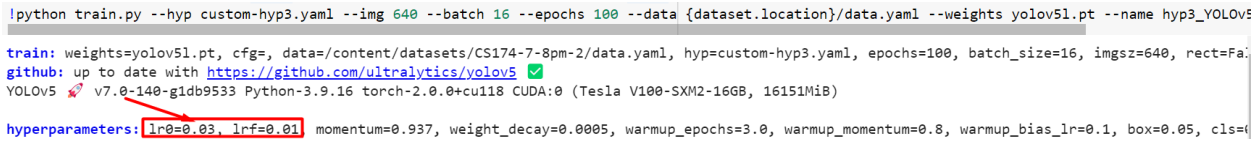
Hyperparameter 2:

```
!python train.py --hyp custom-hyp2.yaml --img 640 --batch 32 --
epochs 75 --data {dataset.location}/data.yaml --weights yolov5l.pt --
name hyp2_YOLOv5l_results --cache
```



Hyperparameter 3:

```
!python train.py --hyp custom-hyp3.yaml --img 640 --batch 16 --
epochs 100 --data {dataset.location}/data.yaml --weights yolov5l.pt --
name hyp3_YOLOv5l_results --cache
```



Note: The YOLOv5 version used in this project was the **Large** scale.

These hyperparameters were selected through an iterative process of experimentation and analysis and were determined to be the best combinations for the specific task and dataset.

During the training process, we will pass several arguments to the train.py script, including the path to our hyperparameters configuration file, the input image size, batch size, number of epochs, the path to our data .yaml file, model configuration, the path to custom weights (if applicable), result names, and whether to cache images for faster training.

Once the training process is complete, we will be able to examine the metrics of our model, such as precision, recall, and mAP score, using TensorBoard. This will allow us to evaluate the performance of our model and make any necessary adjustments before deploying it for use in real-world applications.

VI. Precision, Recall, and Mean Average Precisions (@.5 and @.95)

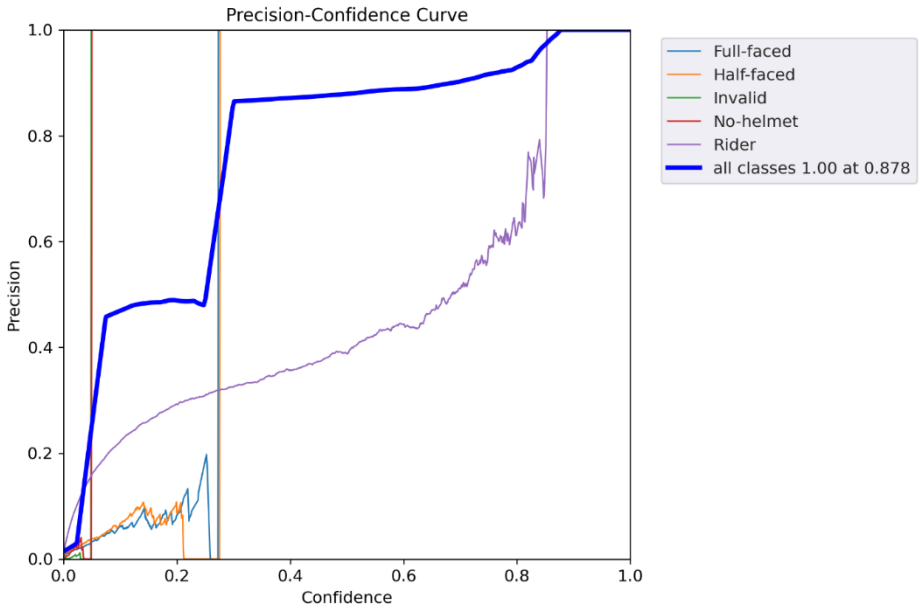
This section includes the results for each Hyperparameter after they had finished the training phase. Their results are represented as numerical values and as graphs generated through TensorBoard.

Hyperparameter 1

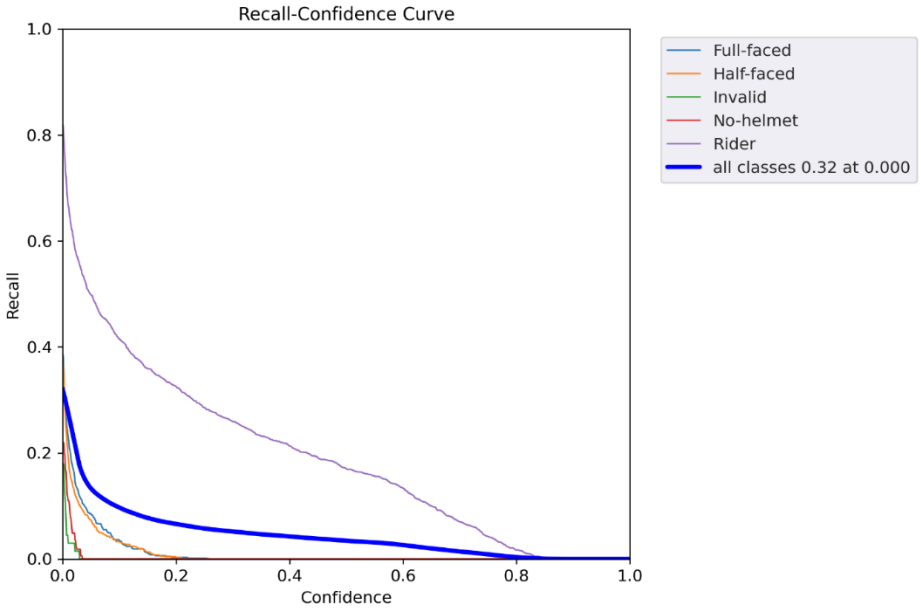
```
50 epochs completed in 0.642 hours.
Optimizer stripped from runs/train/hyp1_YOLOV5l_results/weights/last.pt, 92.9MB
Optimizer stripped from runs/train/hyp1_YOLOV5l_results/weights/best.pt, 92.9MB

Validating runs/train/hyp1_YOLOV5l_results/weights/best.pt...
Fusing layers...
Model summary: 267 layers, 46129818 parameters, 0 gradients, 107.7 GFLOPs
                 Class     Images  Instances          P          R      mAP50   mAP50-95: 100% 6/6 [00:09<00:00,  1.55s/it]
                   all        752       3405      0.472     0.0951     0.0469     0.0156
            Full-faced        752        632     0.0634     0.0332     0.0113    0.00284
            Half-faced        752        912       0.07     0.0307      0.012    0.00323
               Invalid        752         67          1          0   0.000901   0.000314
             No-helmet        752        164          1          0    0.00395    0.00108
                 Rider        752       1630      0.226      0.412      0.206     0.0705
Results saved to runs/train/hyp1_YOLOV5l_results
```

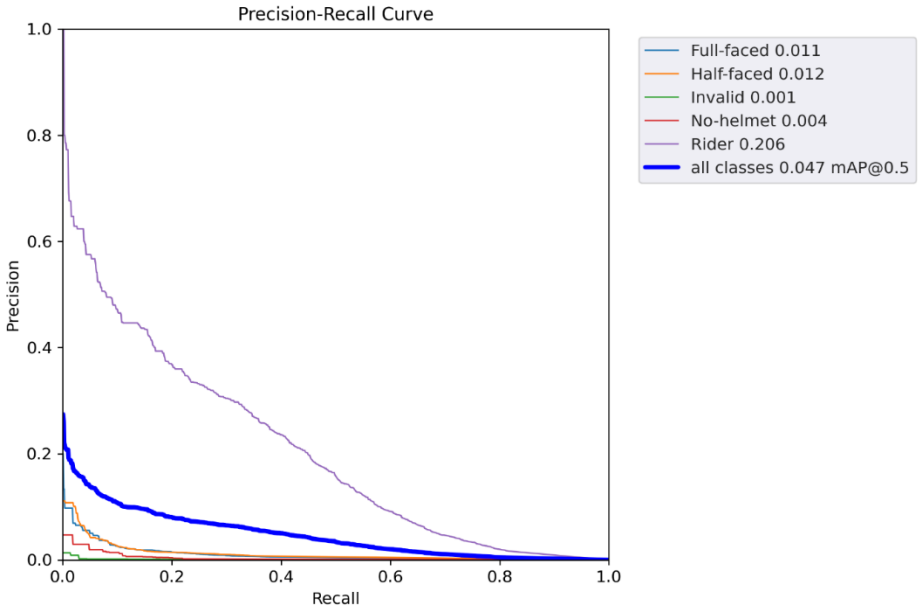
Training Results



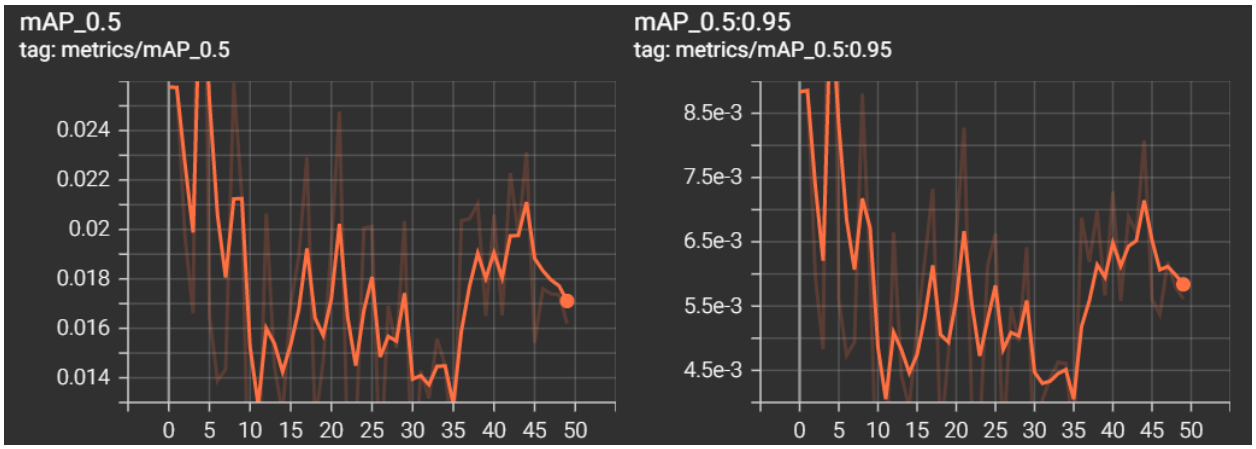
Precision



Recall



Precision-Recall



mAP_0.5 and mAP_0.5:0.95

```
Fusing layers...
Model summary: 267 layers, 46129818 parameters, 0 gradients, 107.7 GFLOPs
val: Scanning /content/datasets/CS174-7-8pm-2/valid/labels.cache... 752 images, 59 backgrounds, 0 corrupt: 100% 752/752 [00:00<?, ?it/s]
                 Class     Images  Instances          P          R      mAP50   mAP50-95: 100% 12/12 [00:10<00:00,  1.09it/s]
                   all        752       3405      0.472     0.0967     0.0471     0.0156
           Full-faced         752        632      0.065     0.0364     0.0117    0.00289
           Half-faced         752        912      0.069     0.0329     0.0122    0.00326
              Invalid         752         67          1          0   0.000836     0.0003
            No-helmet         752        164          1          0    0.00384    0.00105
                Rider         752       1630      0.224      0.414      0.207     0.0706
Speed: 0.2ms pre-process, 3.2ms inference, 3.8ms NMS per image at shape (64, 3, 640, 640)
Results saved to runs/val/exp
```

Testing Results

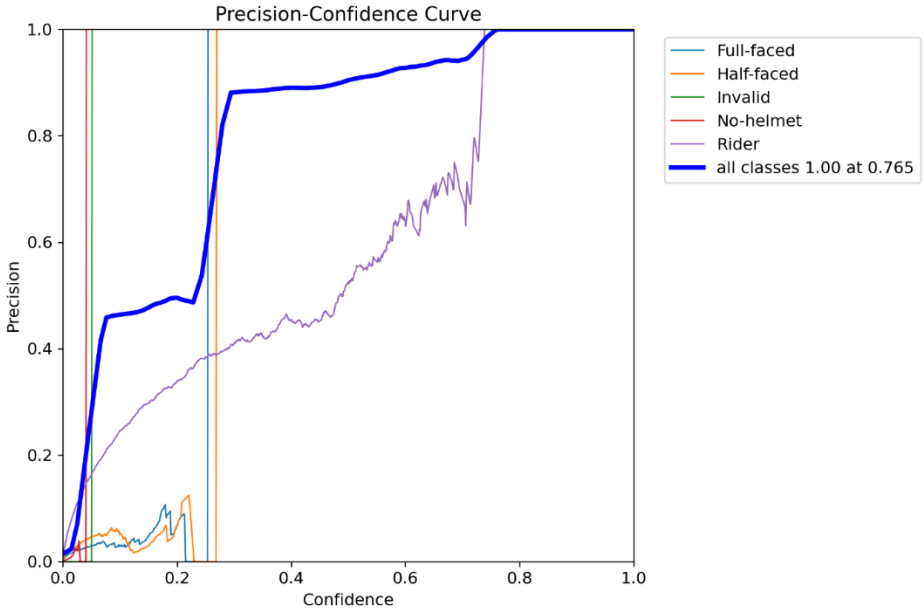
The first hyperparameter settings specified a learning rate of 0.01, a batch size of 64, 50 epochs, and an image size of 640 pixels. These settings were considered as the YOLOv5 model was implemented on a **large** scale. The training results of the model with the specified hyperparameters yield an overall precision of 47.2%, recall of 9.51%, mAP0.5 of 4.69%, and mAP0.5-0.95 of 1.56%. The figures above will illustrate these results more vividly through graphs. According to the training results, classes *Invalid* and *No-helmet* had the highest precision at 100% each and the lowest recall at 0 each. These values indicate that classes *Invalid* and *No-helmet* were too closely labeled to each other (precision), thus no relevant elements were detected (recall) by the model. Next to these classes is the *Rider* class with a precision of 22.6% and a recall of 41.2%. The model was able to identify the *Rider* class with a 41.2% certainty. In fact, the *Rider* class has the highest recall among all the classes, as it was regularly identified on each image in the dataset than the other classes. The same class gained the highest mAP@0.5 and mAP@0.5:0.95, further verifying that the *Rider* class was the only class properly identified by the model.

Hyperparameter 2

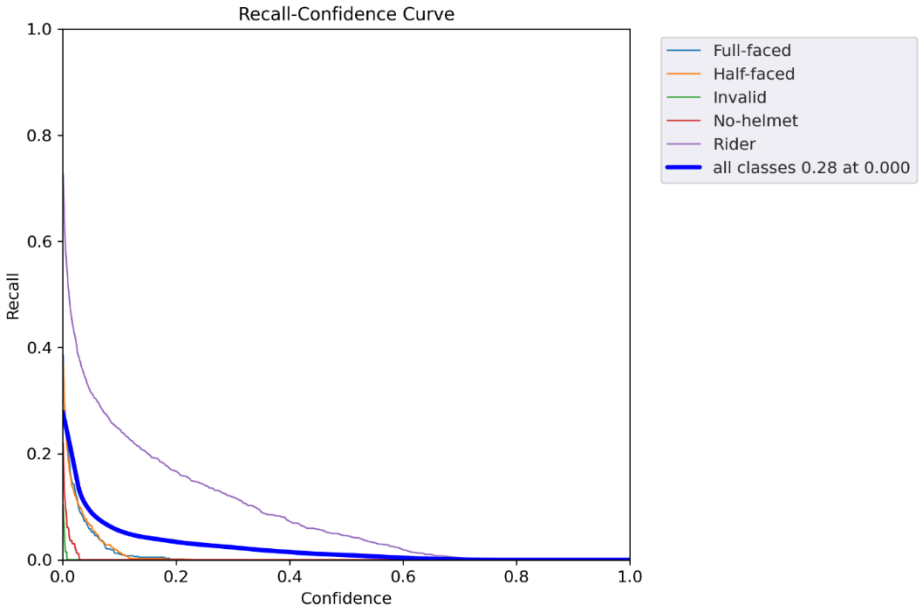
```
75 epochs completed in 1.009 hours.
Optimizer stripped from runs/train/hyp2_YOLOv5l_results/weights/last.pt, 92.9MB
Optimizer stripped from runs/train/hyp2_YOLOv5l_results/weights/best.pt, 92.9MB

Validating runs/train/hyp2_YOLOv5l_results/weights/best.pt...
Fusing layers...
Model summary: 267 layers, 46129818 parameters, 0 gradients, 107.7 GFLOPs
                 Class     Images  Instances          P          R      mAP50   mAP50-95: 100% 12/12 [00:08<00:00,  1.47it/s]
                   all        752       3405      0.457     0.0698     0.0345     0.0102
            Full-faced        752        632     0.0374     0.0348    0.00869    0.00237
            Half-faced        752        912     0.0503      0.034     0.0118    0.00297
               Invalid        752         67          1          0   0.000328   0.000101
             No-helmet        752        164          1          0     0.0027   0.000638
                 Rider        752       1630      0.199       0.28      0.149      0.045
Results saved to runs/train/hyp2_YOLOv5l_results
```

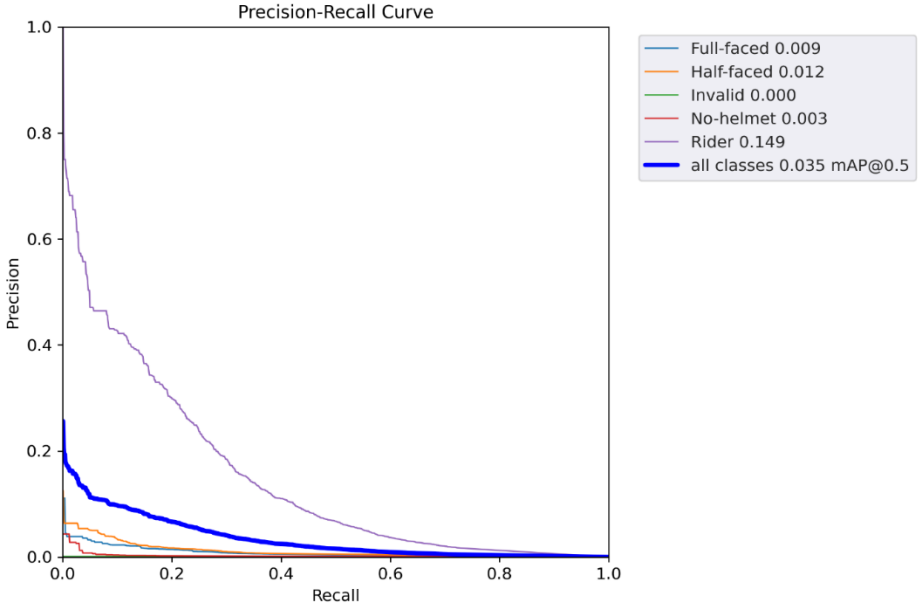
Training Results



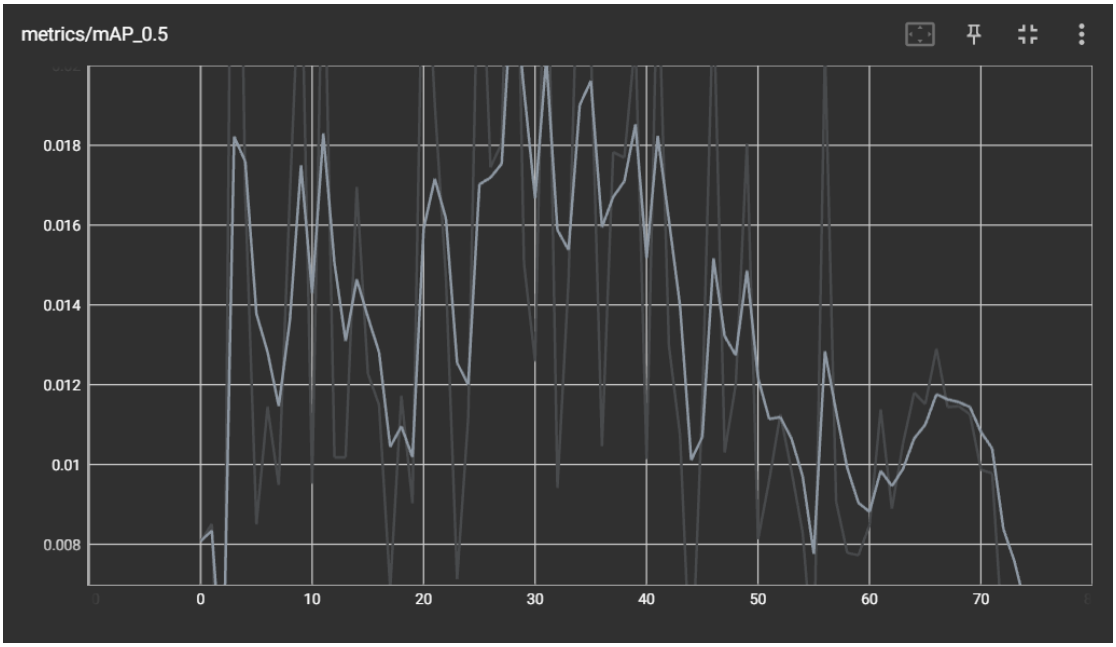
Precision



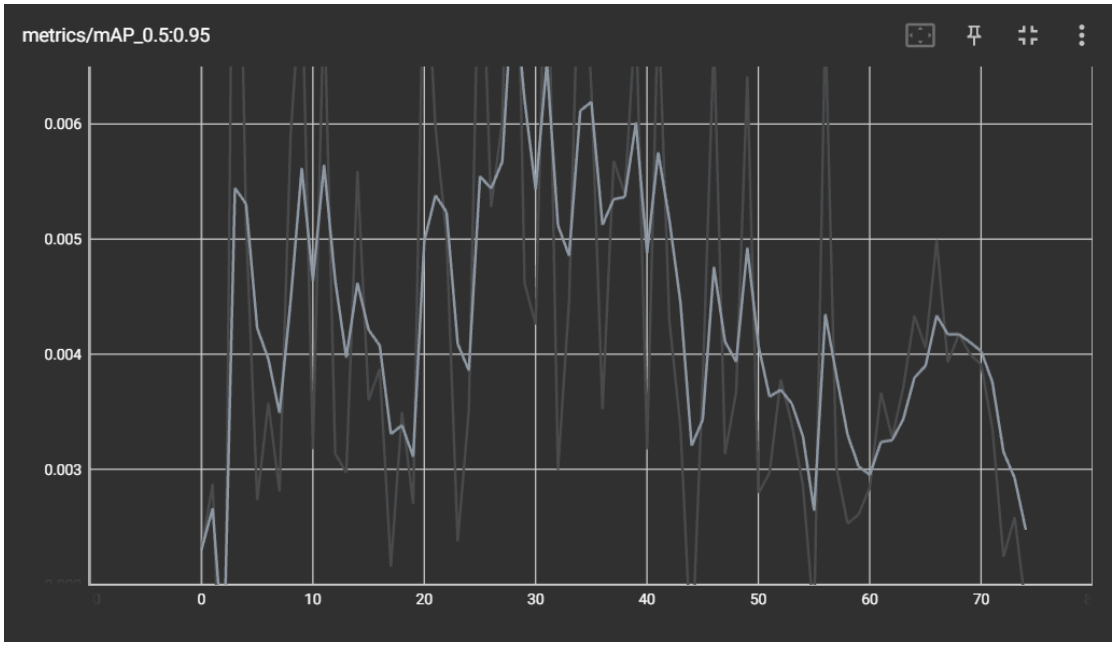
Recall



Precision-Recall



mAP_0.5



mAP_0.5:0.95

```
Fusing layers...
Model summary: 267 layers, 46129818 parameters, 0 gradients, 107.7 GFLOPs
val: Scanning /content/datasets/CS174-7-8pm-2/valid/labels.cache... 752 images, 59 backgrounds, 0 corrupt: 100% 752/752 [00:00<?, ?it/s]
                 Class     Images  Instances          P          R      mAP50   mAP50-95: 100% 24/24 [00:09<00:00,  2.54it/s]
                   all        752       3405      0.458     0.0703     0.0345     0.0103
            Full-faced        752        632     0.0371     0.0348    0.00874     0.0024
            Half-faced        752        912     0.0501      0.034     0.0118    0.00295
               Invalid        752         67          1          0    0.00033   9.68e-05
             No-helmet        752        164          1          0    0.00282   0.000649
                 Rider        752       1630      0.202      0.283      0.149     0.0452
Speed: 0.2ms pre-process, 3.6ms inference, 2.9ms NMS per image at shape (32, 3, 640, 640)
Results saved to runs/val/exp
```

Testing Results

The second hyperparameter setting specified a learning rate of 0.02, a batch size of 32, 75 epochs, and an image size of 640 pixels. As stated in the first hyperparameter, the researchers implemented this YOLOv5 model on a **large** scale. The training results of the model with the specified hyperparameters yield an overall precision of 45.7%, recall of 6.98%, mAP0.5 of 3.45%, and mAP0.5-0.95 of 1.02%. Looking at the figures above, we can see that our model has higher precision than precision, meaning most of its predicted labels are correct compared to the training labels, but it returns very few results. Our mAP results also prove that our model has high false negative and false positive rates.

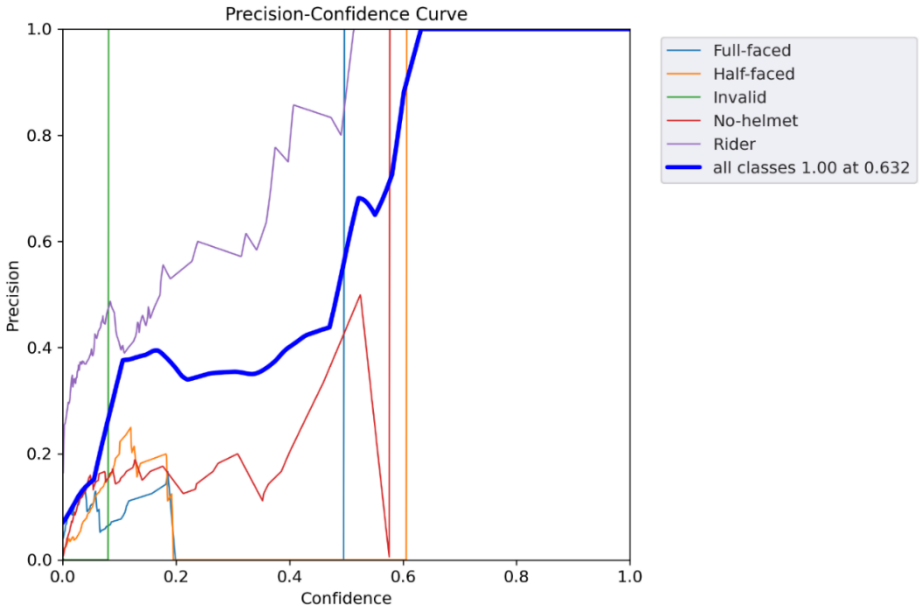
Similar to the first hyperparameter results, *Invalid* and *No-helmet* classes had the highest precision at 100% each and the lowest recall at 0 each. With such high precision, the model predicts these classes 100% correctly, but it is useless due to the low recall, as the model would miss 100% of the classes. Once again, the *Rider* class has the highest recall (28%).

Hyperparameter 3

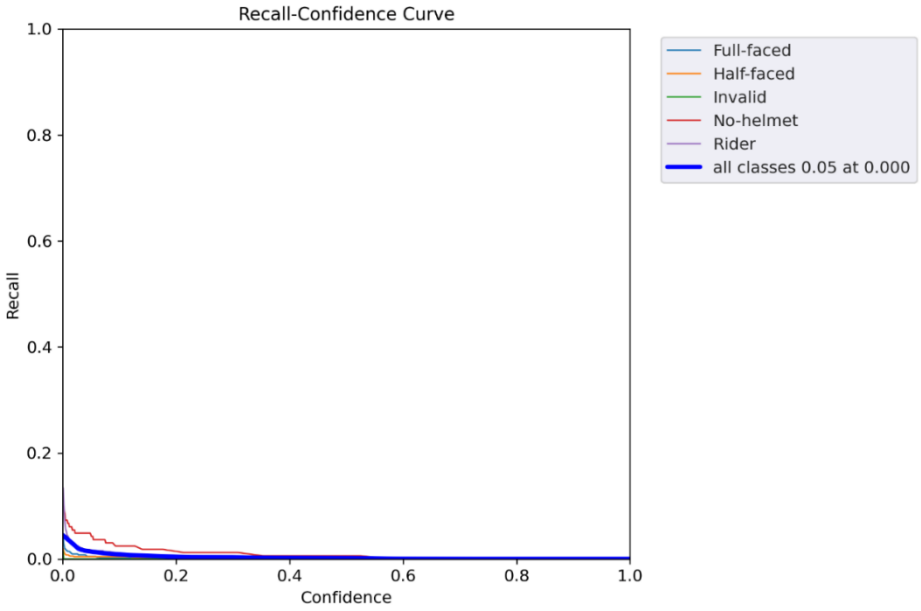
```
100 epochs completed in 2.199 hours.
Optimizer stripped from runs/train/hyp3_YOLOv5l_results/weights/last.pt, 92.9MB
Optimizer stripped from runs/train/hyp3_YOLOv5l_results/weights/best.pt, 92.9MB

Validating runs/train/hyp3_YOLOv5l_results/weights/best.pt...
Fusing layers...
Model summary: 267 layers, 46129818 parameters, 0 gradients, 107.7 GFLOPs
                 Class     Images  Instances          P          R      mAP50   mAP50-95: 100% 24/24 [00:07<00:00,  3.00it/s]
                   all        752       3405     0.0459      0.063     0.0321     0.0106
            Full-faced        752        632     0.0404     0.0427     0.0225    0.00706
            Half-faced        752        912     0.0158     0.0197     0.0093    0.00333
               Invalid        752         67    0.00242     0.0149    0.00123   0.000246
             No-helmet        752        164     0.0069      0.104     0.0155    0.00339
                 Rider        752       1630      0.164      0.134      0.112     0.0389
Results saved to runs/train/hyp3_YOLOv5l_results
```

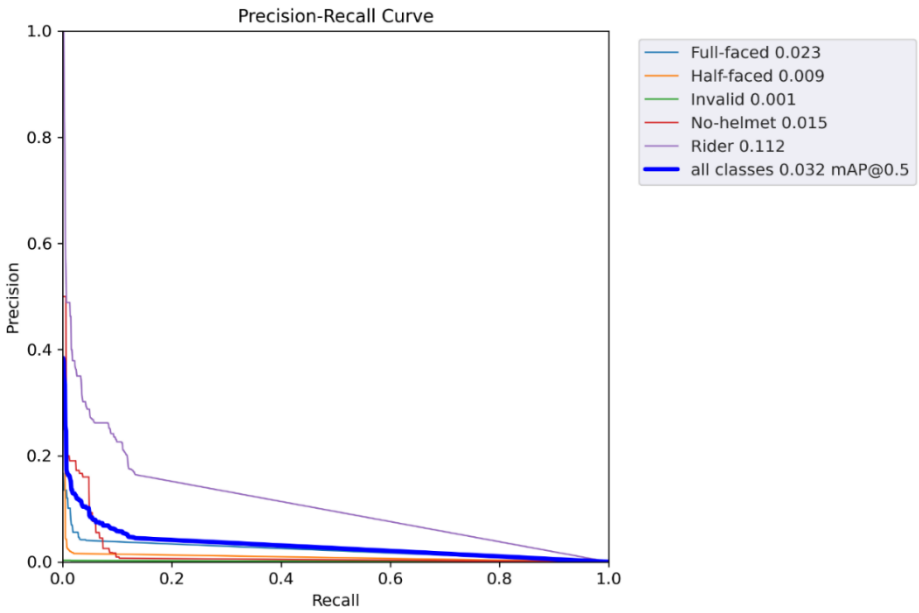
Training Results



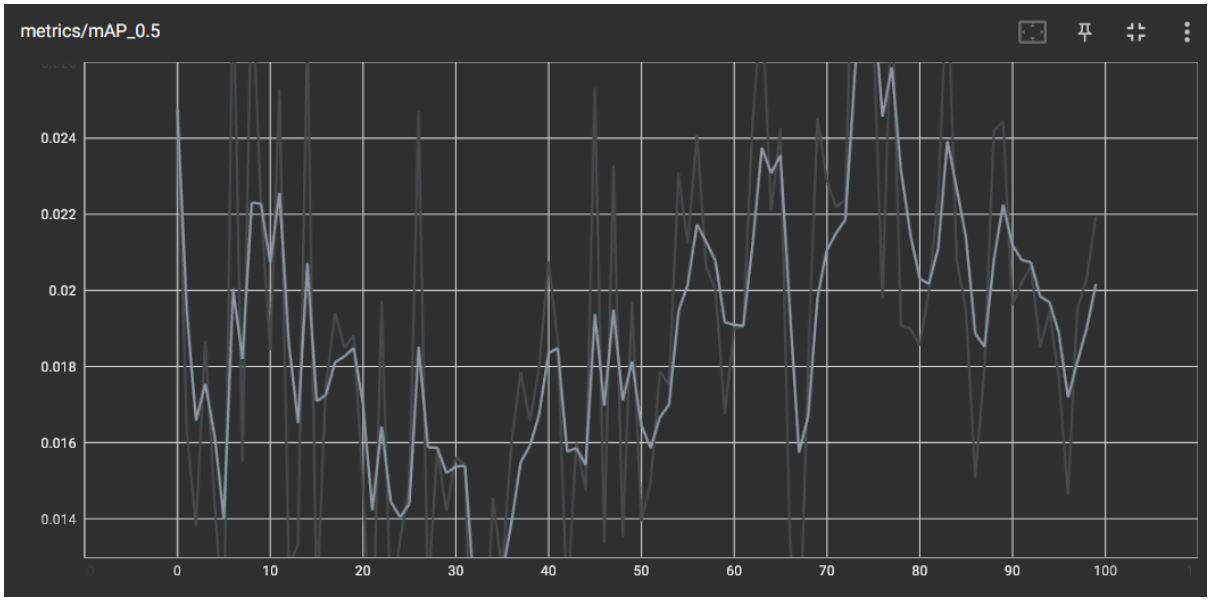
Precision



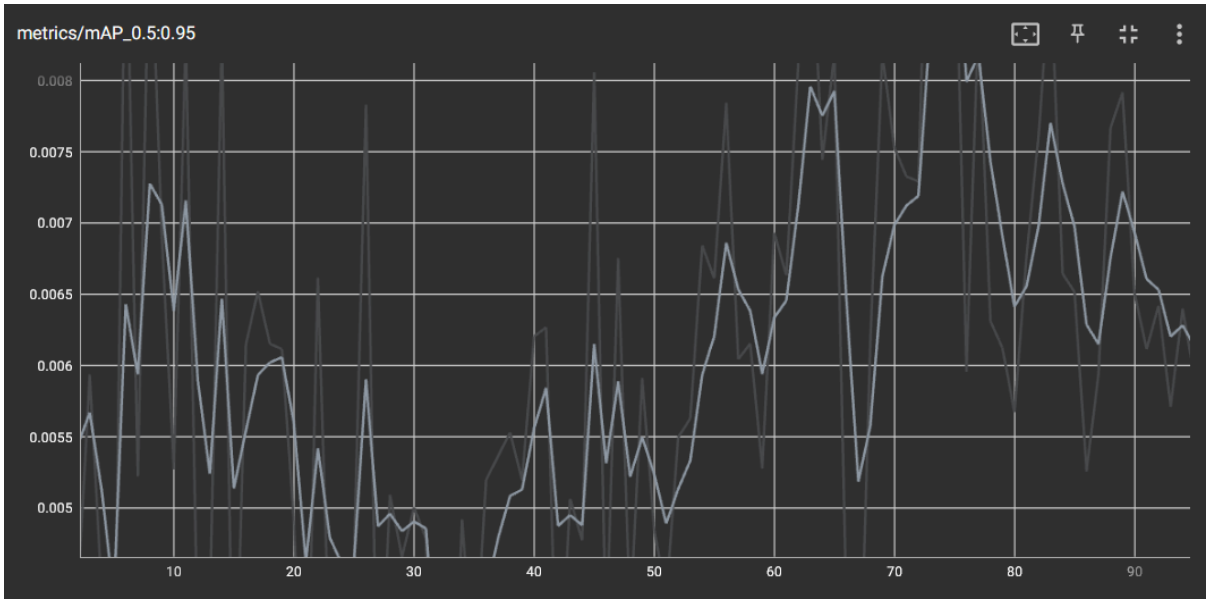
Recall



Precision-Recall



mAP_0.5



mAP_0.5:0.95

```
Fusing layers...
Model summary: 267 layers, 46129818 parameters, 0 gradients, 107.7 GFLOPs
val: Scanning /content/datasets/CS174-7-8pm-2/valid/labels.cache... 752 images, 59 backgrounds, 0 corrupt: 100% 752/752 [00:00<?, ?it/s]
                 Class     Images  Instances          P          R      mAP50   mAP50-95: 100% 47/47 [00:11<00:00,  4.16it/s]
                   all        752       3405      0.046     0.0628      0.032     0.0106
            Full-faced        752        632     0.0409     0.0427     0.0227    0.00673
            Half-faced        752        912     0.0168     0.0208    0.00983    0.00364
               Invalid        752         67    0.00242     0.0149    0.00123   0.000246
             No-helmet        752        164    0.00695      0.104     0.0155    0.00349
                 Rider        752       1630      0.163      0.132      0.111     0.0387
Speed: 0.2ms pre-process, 8.3ms inference, 1.7ms NMS per image at shape (16, 3, 640, 640)
Results saved to runs/val/exp2
```

Testing Results

The third hyperparameter setting specified a learning rate of 0.03, a batch size of 16, 100 epochs, and an image size of 640 pixels. As stated previously, the researchers implemented this YOLOv5 model on a **large** scale. The training results of the model with the specified hyperparameters yield an overall precision of 4.59%, recall of 6.3%, mAP0.5 of 3.21%, and mAP0.5-0.95 of 1.06%. Looking at the figures above, we can see that our model has low precision and recall, meaning that our model returns very few results, and most of its predicted labels are incorrect when compared to the training labels. The graphs further exemplified that the model performs poorly since the area under the curves represents low recall and precision.

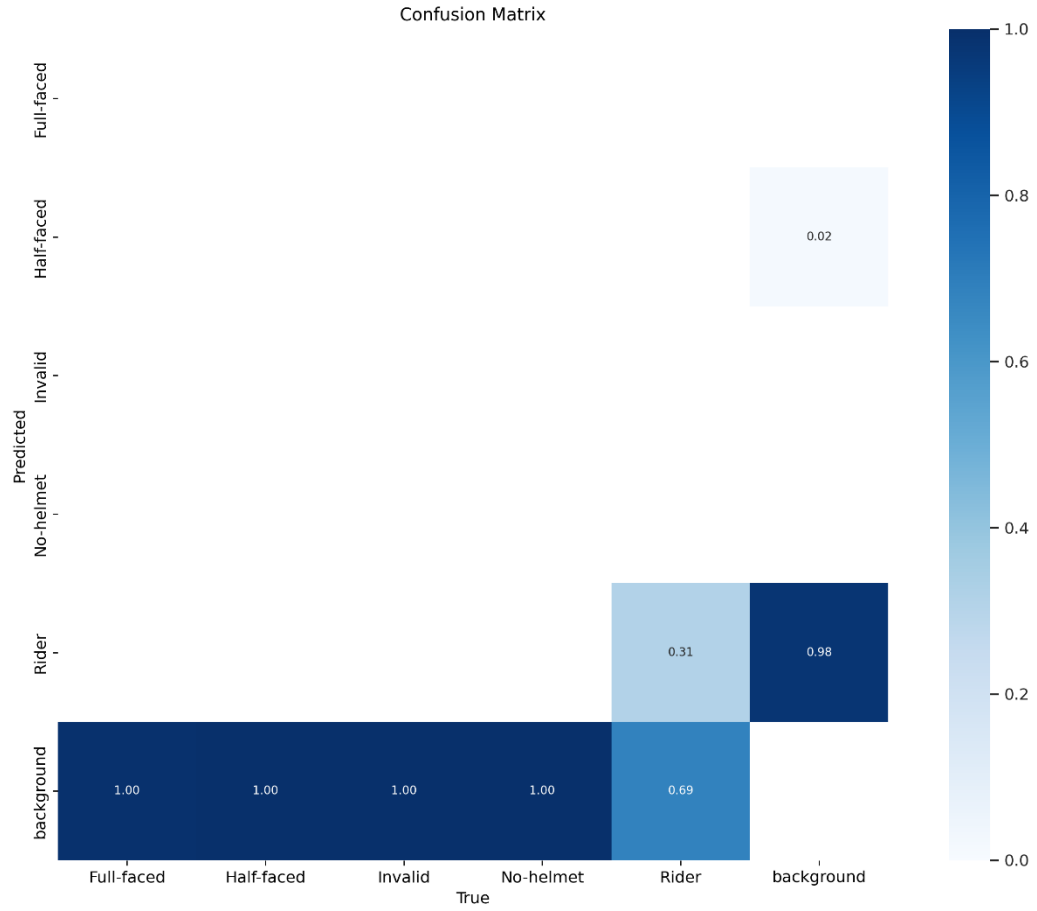
The *Rider* class has the highest recall (13.4%) and precision (16.4%). Unlike the previous two hyperparameter settings, the *Invalid* and *No-helmet* classes have proper recall rates of 1.49% and 10.4%, respectively. However, they have the lowest precision rates of 0.242% and

0.69%, respectively. Despite these results, the model still performs poorly when predicting these classes.

VII. Confusion Matrix Results

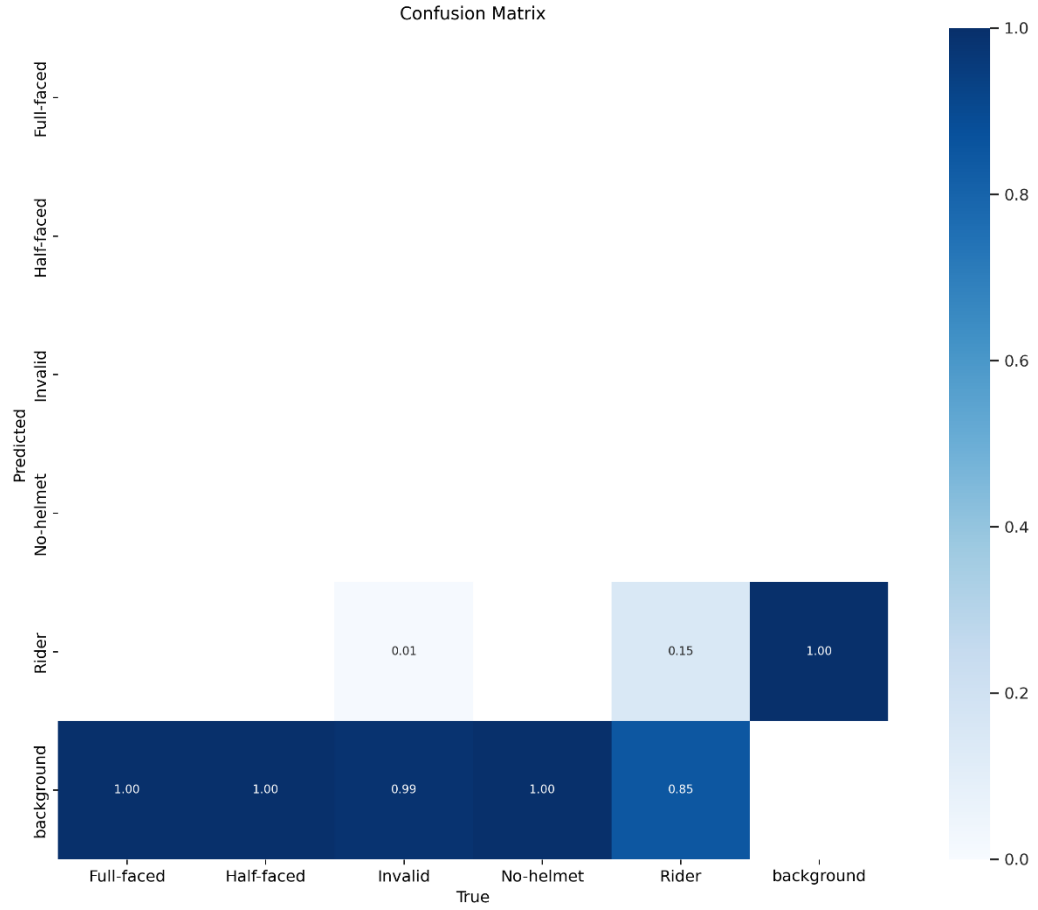
To represent the numerical results of the training phase from the previous section, confusion matrices were generated through TensorBoard and will be discussed in this section.

Hyperparameter 1



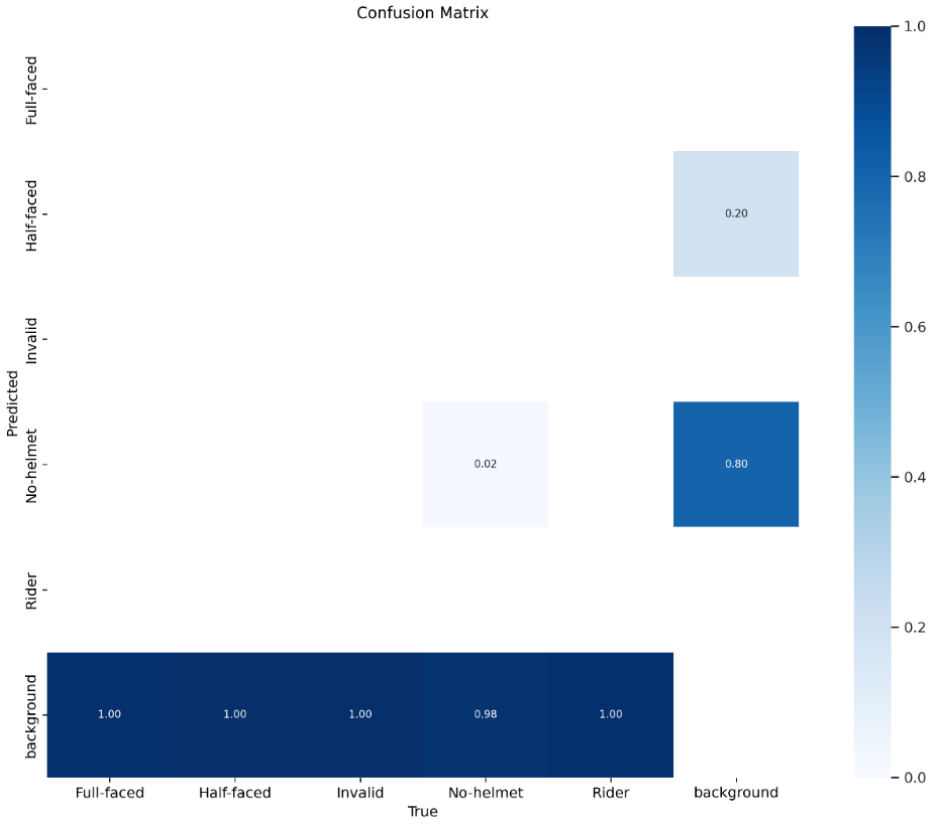
According to the confusion matrix above, the only correctly predicted class with a True Positive value is the *Rider* class (0.31). In fact, it is the only class that did not attain a 100% prediction rate as a background class. However, *Rider* class was predicted at a rate of 98% as a background class, leading to a False Positive rate of the same value. Another class that was incorrectly predicted is the *Half-faced* class, which was identified as a background class at a 2% prediction rate. These results indicate a low-performing model, thus requiring modifications in the dataset and preprocessing phase to improve the model's detection capabilities.

Hyperparameter 2



Like the previous confusion matrix, the only correctly predicted class, with a True Positive value at 0.15, is the *Rider* class. However, the same class was incorrectly identified as a background class with a False Positive value of 1 and a False Negative of 0.85. These values indicate that the model misidentified the *Rider* class more times than it properly identified the said class. All other classes were predicted as a background class, except for the *Invalid* class, which was correctly predicted at a 1% rate. Setting the second hyperparameter settings decreased the performance of the model, as evidenced by the decrease in the True Positive value from 0.31 to 0.15. These results can only be further improved by ensuring that all annotations made in any future dataset are consistent and similar.

Hyperparameter 3



In this confusion matrix, the rows represent the actual class labels, while the columns represent the predicted class labels.

Usually, a confusion matrix would be filled with a range of colors in all rows and columns, thus it can be discerned from the lack of color that the model underperformed. Looking at the last row, all classes were predicted 100% as the background. Except for *No-helmet*, as the model correctly predicted 2% as *No-helmet*. However, there were also 80% *No-helmet* that the model incorrectly predicted as background, leading to 80% false negatives (FN). Moving on to the second row, there were no correctly predicted *Half-faced* helmets; instead, 20% of *Half-faced* helmets were incorrectly predicted as background.

Based on this confusion matrix, there are many areas where the model performs poorly and could have done much better. The performance for Hyperparameter 3 is worse than the previous two hyperparameter as the *Rider* class was no longer correctly predicted.

VIII. Hyperparameter Results (Learning Rate, Batch Size, Epochs, Image Size, and Large Scale)

Hyperparameter 1

The first hyperparameter setting specified a learning rate of 0.01, a batch size of 64, 50 epochs, and an image size of 640 pixels. These settings were considered as the YOLOv5 model was implemented on a **large** scale. A learning rate of 0.01 means that the training process increases at that value for every iteration. Since the training set consisted of 5,667 images, the dataset was divided into 88 batches, each with sixty-four (batch size = 64) samples. One epoch implements 88 batches (or updates) to the model. With 50 epochs, the model will be exposed to or pass through the whole dataset 50 times. That is a total of 4,400 batches during the entire training process. Since the epoch size increased to 50, training completed in nearly 40 minutes. The table below shows the training results of the model after the specified settings were applied.

Class	Precision	Recall	mAP0.5	mAP0.50-0.95
all	0.472	0.0951	0.0469	0.0156
Full-faced	0.0634	0.0332	0.0113	0.00284
Half-faced	0.07	0.0307	0.012	0.00323
Invalid	1	0	0.000901	0.000314
No-helmet	1	0	0.00395	0.00108
Rider	0.226	0.412	0.206	0.0705

Hyperparameter 2

The second hyperparameter setting consisted of a learning rate of 0.02, a batch size of 32, 75 epochs, and an image size of 640 pixels. Since the learning rate was configured before training, then its means at every iteration, the training progress will increase by 0.02. The training set consisted of 5,667 images, and a batch size of 32 and 75 epochs was chosen. This means the dataset will be divided into 177 batches, each with thirty-two (32) samples. In this model, one epoch will involve 177 batches or 177 updates to the model. With 75 epochs, the model will be exposed to or pass through the whole dataset 75 times. That is a total of 13,275 batches during the entire training process. Since the epoch size increased to 75, training completed in 1 hour. The training results for this model is specified on the table below.

Class	Precision	Recall	mAP0.5	mAP0.50-0.95
all	0.457	0.0698	0.0345	0.0102
Full-faced	0.0374	0.0348	0.00869	0.00237
Half-faced	0.0503	0.034	0.0118	0.00297
Invalid	1	0	0.000328	0.000101
No-helmet	1	0	0.0027	0.000638
Rider	0.199	0.28	0.149	0.045

Hyperparameter 3

The third hyperparameter setting had an image size of 640 pixels, a learning rate of 0.03, a batch size of 16, and 100 epochs. At every iteration, the training progress will increase by 0.03. A batch size of 16 and 100 epochs was selected for the training set, which included 5,667 photos. Thus, 354 batches with 16 samples each will be created from the dataset. This model will have 354 batches, or model updates, during one epoch. The entire dataset will be exposed to or run through 100 times using 100 epochs. Throughout the whole training procedure, there were 35,400 batches in all. Training took twice as long (2 hours) as the second hyperparameter since the epoch size was increased to 100. The table below contains information about this model's training performance.

Class	Precision	Recall	mAP0.5	mAP0.50-0.95
all	0.0459	0.063	0.0321	0.0106
Full-faced	0.0404	0.0427	0.0225	0.00706
Half-faced	0.0158	0.0197	0.0093	0.00333
Invalid	0.00242	0.0149	0.00123	0.000246
No-helmet	0.0069	0.104	0.0155	0.00339
Rider	0.164	0.134	0.112	0.0389

IX. Comparison and Discussion of Results and Observations

In this project, different hyperparameter settings were considered and implemented to determine which produced the best results for the YOLOv5 model. After running the model three times, each with the three recommended hyperparameters, we were able to find that the first hyperparameter (Hyperparameter 1) produced the best results overall. The first hyperparameter setting specified a learning rate of 0.01, a batch size of 64, 50 epochs, and an image size of 640 pixels. These settings were considered as the YOLOv5 model was implemented on a large scale. These results indicate that a learning rate of 0.01 paired with a batch size of 64 is an ideal pair in this situation to produce relatively good results. According to the table below, Hyperparameter 1 produced the highest recall of 9.67% than the other Hyperparameters. This result means that setting the model at the specified hyperparameters of Hyperparameter 1 makes the model more capable of correctly detecting objects at a rate of 9.67%. The mAP50 score of the first hyperparameter also shows a higher score, indicating a higher average precision score for the overall model.

However, when these values were compared to the original hyperparameters, we noticed that the original hyperparameters still had better results in terms of precision and recall. The original hyperparameters had a higher recall of 26.3% than Hyperparameter 1. The recall results produced by the original hyperparameters indicate that the original hyperparameters were more ideal for object detection in this study, as objects are more correctly detected at a learning rate of 0.01, batch size of 16, 50 epochs, and an image size of 416 pixels. Moreover, the results of the model could have also been affected by the state of the dataset and its augmentation properties. In fact, the dataset used with the original hyperparameters had more saturated annotations than the dataset that was used for the three hyperparameters, suggesting that the dataset's status is influential on the results of a model. If the dataset used for the modified study were annotated in the same manner as the previous study that utilized the original hyperparameters, it is possible that the results for the current study could still be improved.

Overall Results (*all* class)	**Hyperparameter 1**	**Hyperparameter 2**	**Hyperparameter 3**	**Original Hyperparameter (Large Scale)**
Precision	0.472	0.457	0.0459	0.328
Recall	0.0967	0.0698	0.063	0.263
mAP50	0.0471	0.0345	0.0321	0.271
mAP50-95	0.0156	0.0102	0.0106	0.143

X. Conclusion and Recommendations

In this chapter assignment, the performance of the YOLOv5 model was tested on a large-scale object detection task using three different hyperparameter settings. The results indicate that the model's performance remains poor, despite changing the hyperparameter settings. The *Rider* class was the only class consistently predicted in Hyperparameter 1 and Hyperparameter 2 with some level of accuracy, while the other classes mainly were predicted as background. The analysis suggests that improvements in the dataset and preprocessing phase could help improve the model's performance, and the use of other models or further optimization of hyperparameters may also be necessary.

In conclusion, this analysis tested three different hyperparameter settings for implementing the YOLOv5 model on a large-scale object detection task. The first hyperparameter setting yielded the best results with an overall precision of 47.2%, recall of 9.51%, mAP0.5 of 4.69%, and mAP0.5-0.95 of 1.56%. The second hyperparameter setting showed lower precision and recall, indicating that the model predicted most labels correctly but yielded few results. The third hyperparameter setting yielded the least results with low precision and recall rates. The *Rider* class was the only class adequately identified by the model in all three hyperparameter settings. In contrast, the *Invalid* and *No-helmet* classes had high precision but low recall rates. Despite some improvements in the second hyperparameter setting, the YOLOv5 model's performance remains poor, indicating the need for further optimization of hyperparameters or other models.

Based on the information about the three different hyperparameters, the model needs to perform better in correctly predicting the different classes. The *Rider* class is the only one consistently predicted with some accuracy, while the other classes are mainly predicted as background. Additionally, there are varying false positives and false negatives, indicating that the model needs improvements in the dataset and preprocessing phase. The third hyperparameter setting has the worst performance, as the *Rider* class was no longer correctly predicted. Improving the consistency and quality of the annotations in the dataset could help improve the model's overall performance.

Now, based on the poor results of the models, the researchers formed three theories. The first theory proposes that the results of the model came to be due to the quality of the images. Since the images were taken at 7:00 PM - 8:00 PM, the much lower light levels at night mean that images lead to very narrow focus areas with much of the picture plane subject to blurring. Pictures taken at night do not have the advantage of a greater light source, so it must extend the shutter speed, often resulting in the capture of blurred movement and light-trails evident. Furthermore, the low-resolution images could not capture finer details and produce smoother edges. The clarity and contrast of the images could have been better, as there were no clear distinctions between different areas of an image.

The second theory, on the other hand, focuses on the annotation process of the dataset. The dataset that was utilized for this chapter assignment was taken from the small, medium, and large cluster groups of time 7:00 PM - 8:00 PM. There were eight (8) different datasets in total, and they were all compiled into one whole dataset from 7:00 PM - 8:00 PM. Furthermore, annotating a

dataset is a subjective process, and there is a risk that different annotators could interpret the same data differently. There is a risk of inconsistency in the annotations, which could impact the quality of the results. This theory proposes that due to the varying annotations placed in the dataset, the model could not distinguish the different classes in the dataset.

Finally, the last theory posits that the dataset needs more data as it is unbalanced and at risk of personal biases and assumptions. The researchers noticed that a few of the annotations were inaccurate; undistinguishable helmets were labeled 'Invalid,' and blurry images of the helmets were labeled 'No-helmet' despite the apparent sign of a helmet. Many turned to this method due to the need to find 'Invalid' and 'No-helmet' classes in the images since many motorcycle riders always wore proper helmets. Likewise, specific knowledge or expertise may be required to annotate the data accurately.

Therefore, these theories recommend improving the dataset's annotations' quality and consistency to enhance the model's overall performance. Moreover, improving the quality of the dataset can significantly impact the model's performance. Balancing the dataset by collecting more samples for the underrepresented classes can help improve the model's performance. Exploring other object detection models and optimizing hyperparameters to strengthen the model's performance may also be beneficial. Additionally, improving the dataset and preprocessing phase may help reduce false positives and false negatives, leading to better object detection results.

XI. Google Links

Roboflow Dataset Link:

https://universe.roboflow.com/cs174-t78/cs174-7-8pm

Roboflow Compressed Zip File:

https://drive.google.com/file/d/1Woabqy1u3ni7K7sKDW967CTaH2ToHupw/view?usp=sharing

Google Drive Link of the Sample Video Output:

- Hyperparameter 1
- Hyperparameter 2
- Hyperparameter 3

Google Colab Link of the model:

- Hyperparameter 1 Link
- Hyperparameter 2 Link
- Hyperparameter 3 Link

Reference Compilation:

https://drive.google.com/file/d/1mW3Abb2XvcJ9PoXYZoVXf8NR8fWLSRLu/view?usp=sharing

References

Arie, L. G., PhD. (2023, February 16). The practical guide for Object Detection with YOLOv5 algorithm. *Medium*. https://towardsdatascience.com/the-practical-guide-for-object-detection-with-yolov5-algorithm-74c04aac4843

Gad, A. F. (2021). Mean Average Precision (mAP) Explained | Paperspace Blog. *Paperspace Blog*. https://blog.paperspace.com/mean-average-precision/

How to Train a YOLOv5 Object Detection Model in Google Colab? (n.d.). https://www.forecr.io/blogs/ai-algorithms/how-to-train-a-yolov5-object-detection-model-in-google-colab

Jordan, J. (2023). Setting the learning rate of your neural network. *Jeremy Jordan*. https://www.jeremyjordan.me/nn-learning-rate/

Kundu, R. (2023, February 2). Confusion Matrix: How To Use It & Interpret Results [Examples]. *V7*. https://www.v7labs.com/blog/confusion-matrix-guide

Munawar, M. R. (2022, August 4). How do Hyperparameters of YOLOv5 Work? - Augmented Startups - Medium. *Medium*. https://medium.com/augmented-startups/how-hyperparameters-of-yolov5-works-ec4d25f311a2

Narkhede, S. (2021, June 15). Understanding Confusion Matrix - Towards Data Science. *Medium*. https://towardsdatascience.com/understanding-confusion-matrix-a9ad42dcfd62

Precision-Recall. (n.d.). Scikit-learn. https://scikit-learn.org/stable/auto_examples/model_selection/plot_precision_recall.html

Solawetz, J. (2022). Mean Average Precision (mAP) in Object Detection. *Roboflow Blog*. https://blog.roboflow.com/mean-average-precision/

Tips for Best Training Results - YOLOv8 Docs. (n.d.). https://docs.ultralytics.com/yolov5/tips_for_best_training_results/#training-settings

Ultralytics. (n.d.). *About hyperparameters · ultralytics/yolov5 · Discussion #7319*. GitHub. https://github.com/ultralytics/yolov5/discussions/7319

Yohanandan, S. (2021, December 14). mAP (mean Average Precision) might confuse you! - Towards Data Science. *Medium*. https://towardsdatascience.com/map-mean-average-precision-might-confuse-you-5956f1bfa9e2