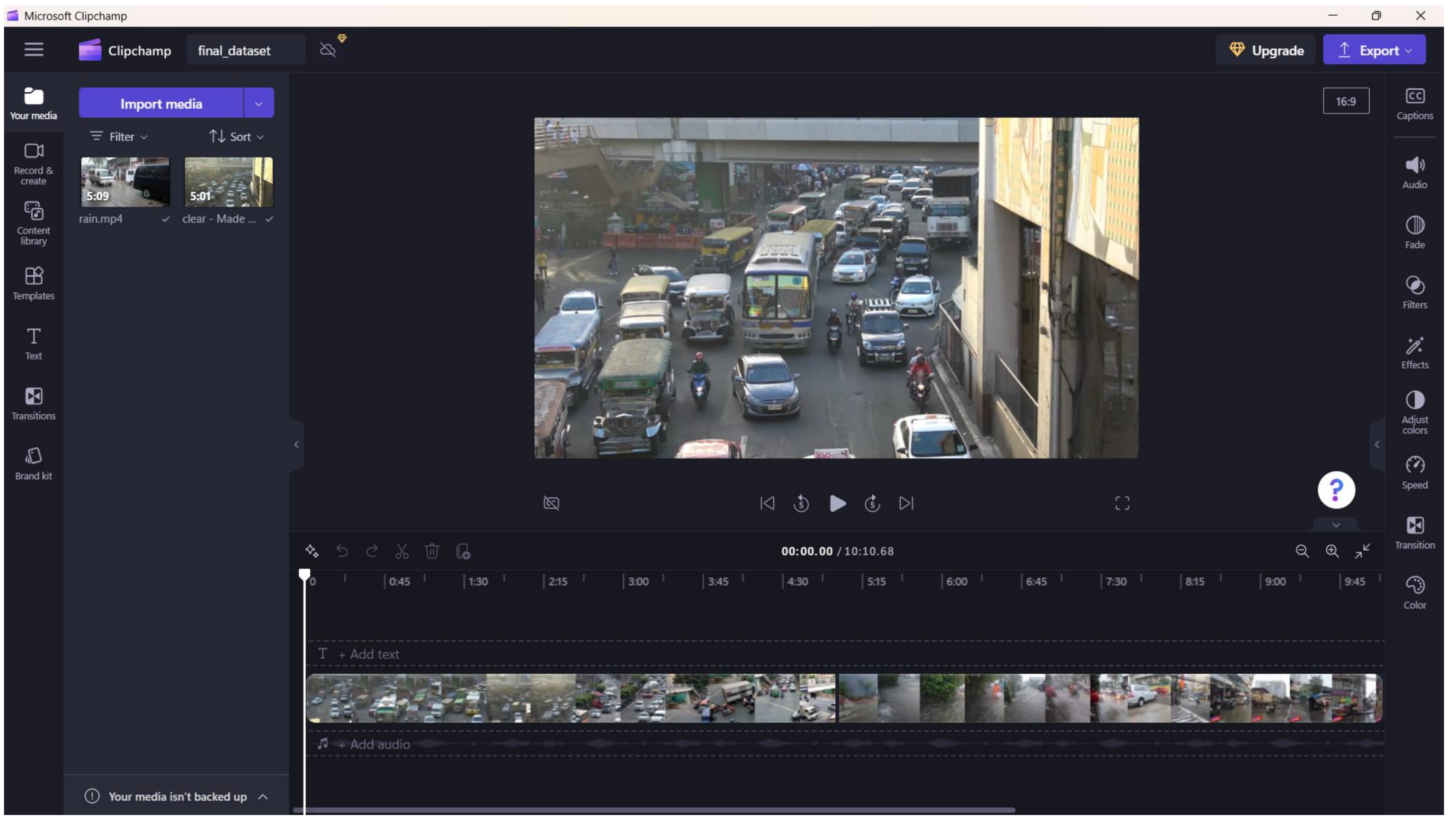
Darryl Babar, Christian Henry Miguel Caruz, Miguel Soniel, Jan Edgar Tupas
CS174

Modelling 1 Documentation

DATA COLLECTION

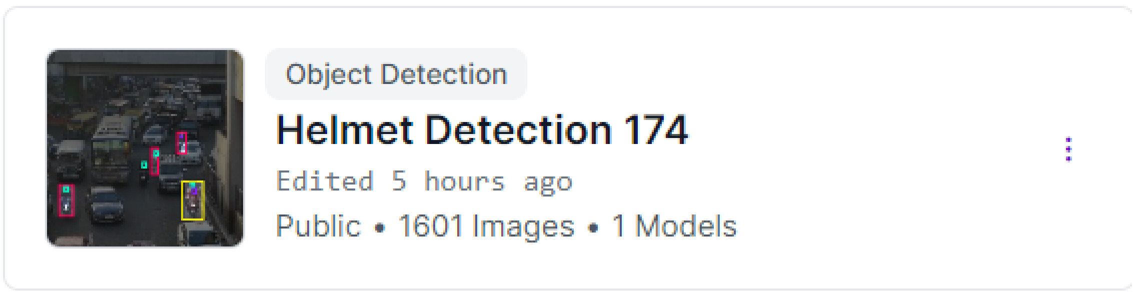
We created a dataset of motorcycle riders by collecting, editing, and merging videos from different sources. We collected and downloaded 4 videos, 2 with non-rainy weather and 2 with rainy weather.

We used Microsoft Clipchamp to cut the videos so that we can only get the sections that contained motorcycle riders. We compiled the videos into one final 10-minute video for the dataset.



With our final video, we uploaded it to roboflow to prepare the dataset. We sampled our video into 3 frame per second and we obtained a total of 1601 images from the sampled video.

DATA PREPROCESSING

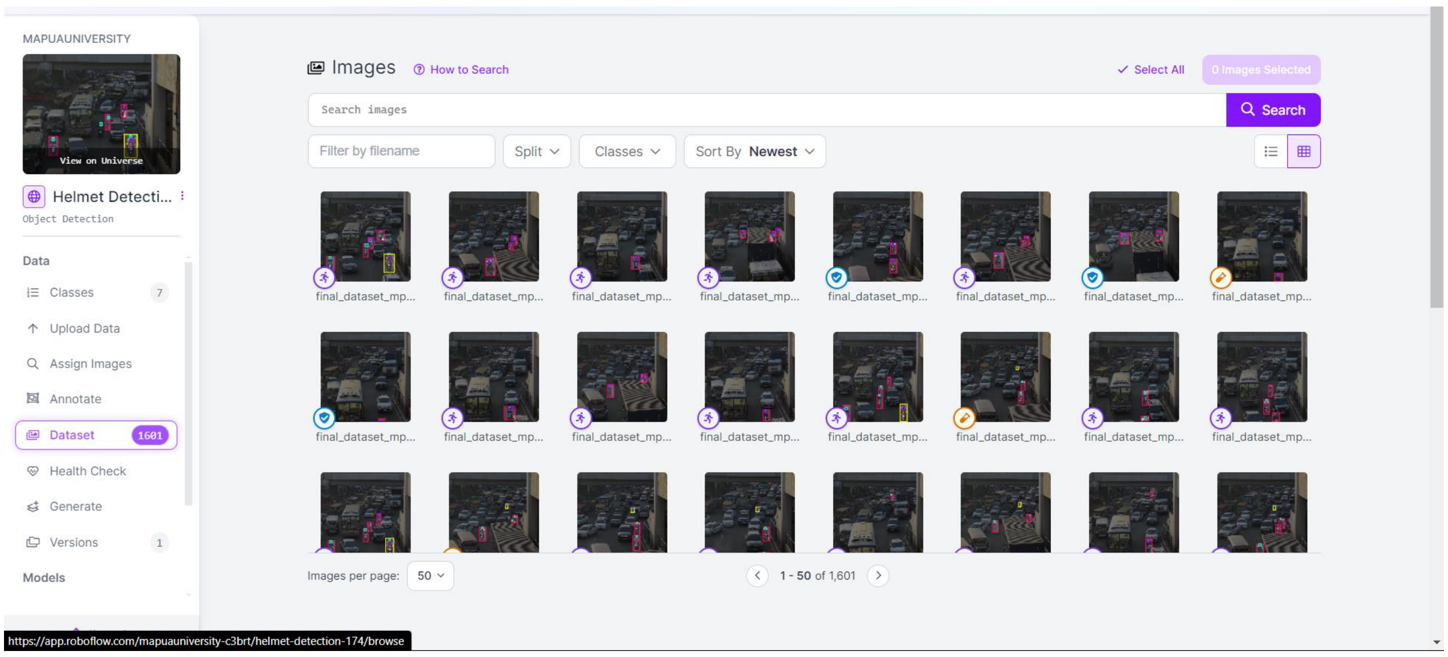


Object Detection

Helmet Detection 174

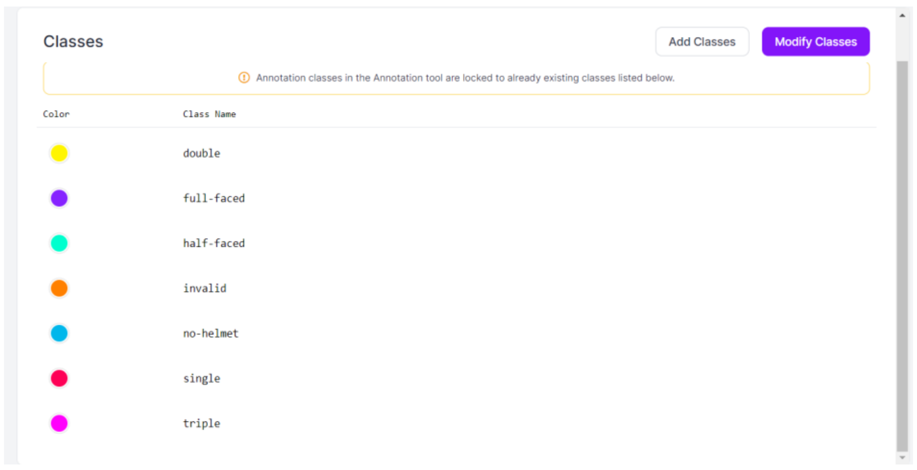
Edited 5 hours ago

Public • 1601 Images • 1 Models



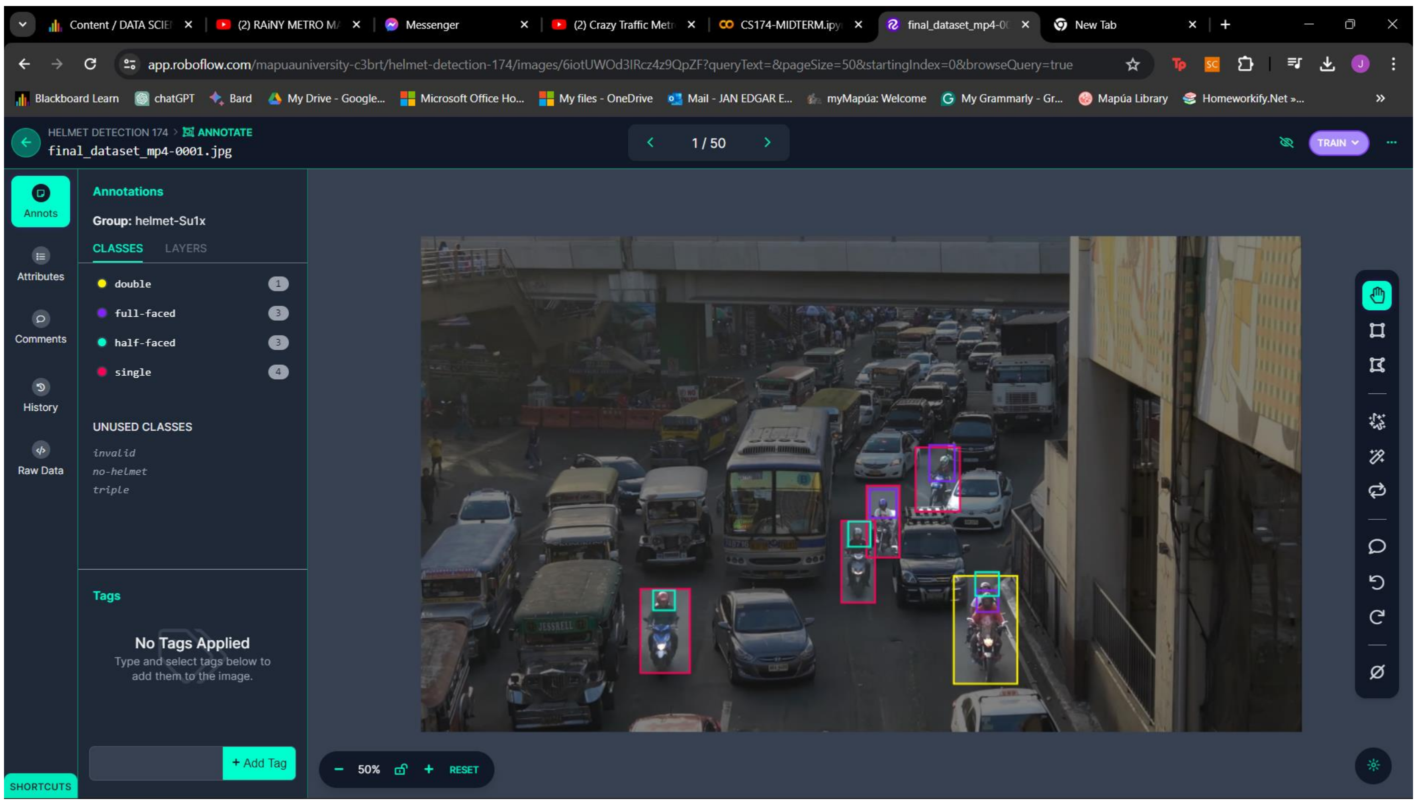
https://app.roboflow.com/mapuauniversity-c3brt/helmet-detection-174/browse

We used roboflow's annotation tool to annotate the dataset. We placed bounding boxes on the motorcycle helmets and classified them according to full-faced, half-faced, invalid, and no-helmet. We also annotated the riders whether they were single, double, or triple. So we had 7 classes in total.

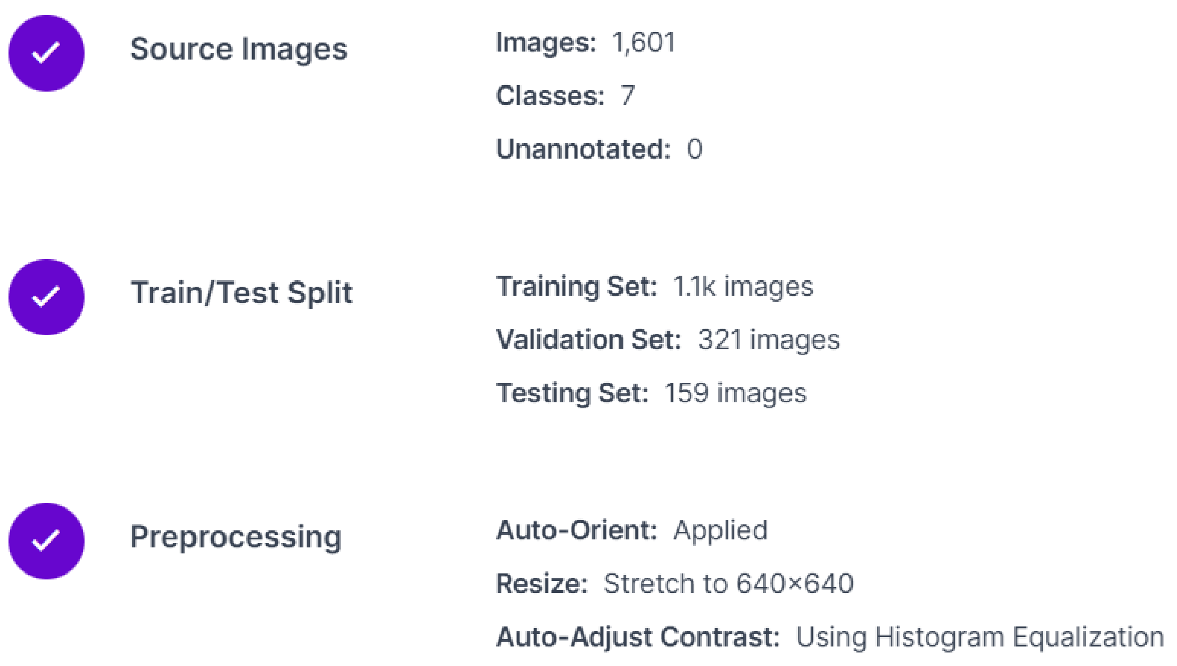


Annotation classes in the Annotation tool are locked to already existing classes listed below.

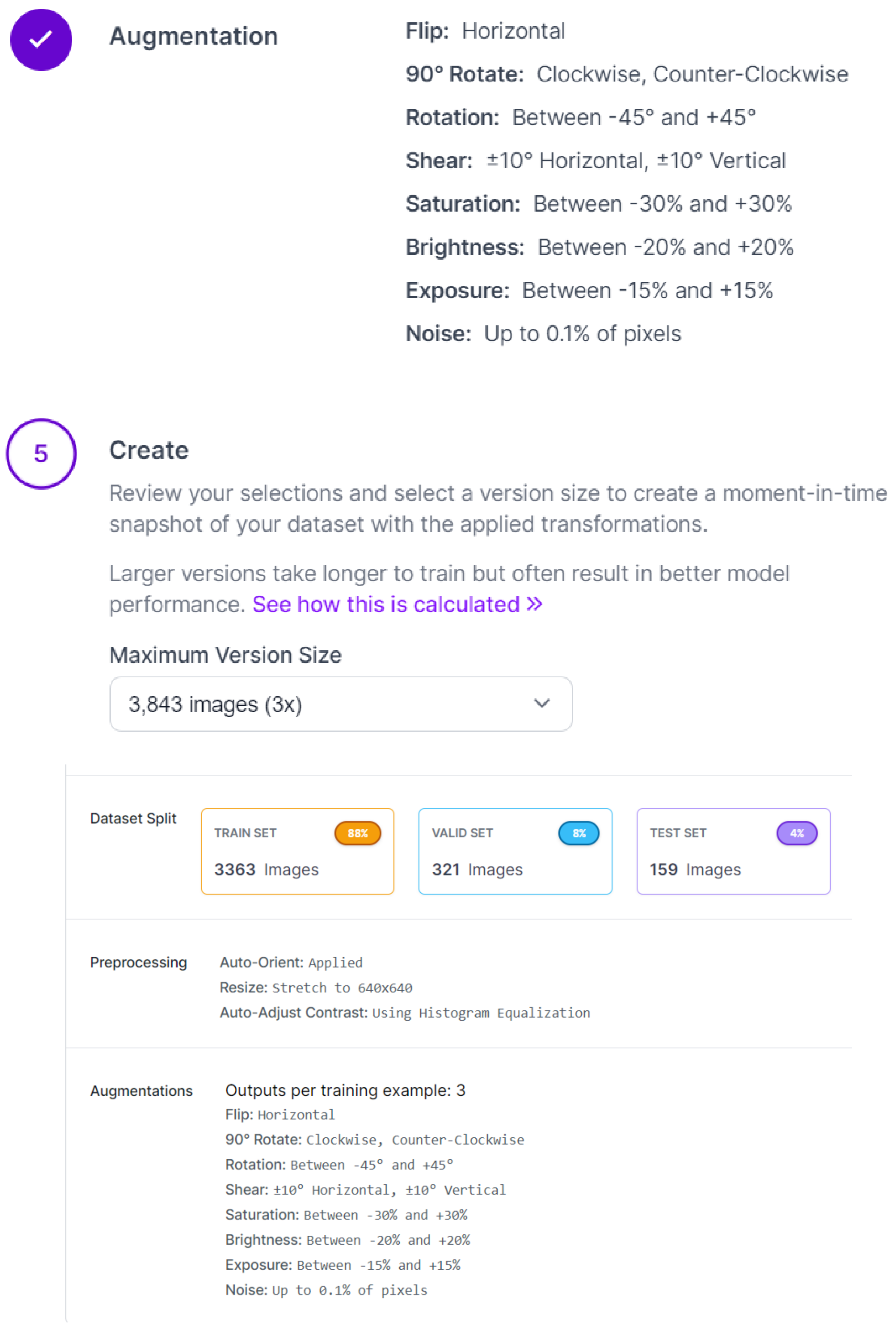
Color	Class Name
🟡	double
🟣	full-faced
🟢	half-faced
🟠	invalid
🔵	no-helmet
🔴	single
🟣	triple


app.roboflow.com/mapuauniversity-c3brt/helmet-detection-174/images/6iotUWOd3lRcz4z9QpZF?queryText=&pageSize=50&startingIndex=0&browseQuery=true
HELMET DETECTION 174 > ANNOTATE
final_dataset_mp4-0001.jpg
1 / 50
TRAIN
Annots
Attributes
Comments
History
Raw Data
Annotations
Group: helmet-Su1x
CLASSES
LAYERS
double 1
full-faced 3
half-faced 3
single 4
UNUSED CLASSES
invalid
no-helmet
triple
Tags
No Tags Applied
Type and select tags below to add them to the image.
+ Add Tag
SHORTCUTS
50%
RESET

Once we finished annotating, we created a version of our dataset. We used a 70:20:10 ratio for splitting the dataset. This means we have 1100 images for training, 321 images for validation and 159 images for testing. We also applied preprocessing steps to enhance the quality of the dataset. We also applied augmentation to increase the dataset since, resulting in 3843 images from 1601 images.



Source Images	**Images:** 1,601 **Classes:** 7 **Unannotated:** 0
Train/Test Split	**Training Set:** 1.1k images **Validation Set:** 321 images **Testing Set:** 159 images
Preprocessing	**Auto-Orient:** Applied **Resize:** Stretch to 640×640 **Auto-Adjust Contrast:** Using Histogram Equalization



Augmentation

Flip: Horizontal
90° Rotate: Clockwise, Counter-Clockwise
Rotation: Between -45° and +45°
Shear: ±10° Horizontal, ±10° Vertical
Saturation: Between -30% and +30%
Brightness: Between -20% and +20%
Exposure: Between -15% and +15%
Noise: Up to 0.1% of pixels

5 Create

Review your selections and select a version size to create a moment-in-time snapshot of your dataset with the applied transformations.

Larger versions take longer to train but often result in better model performance. See how this is calculated »

Maximum Version Size

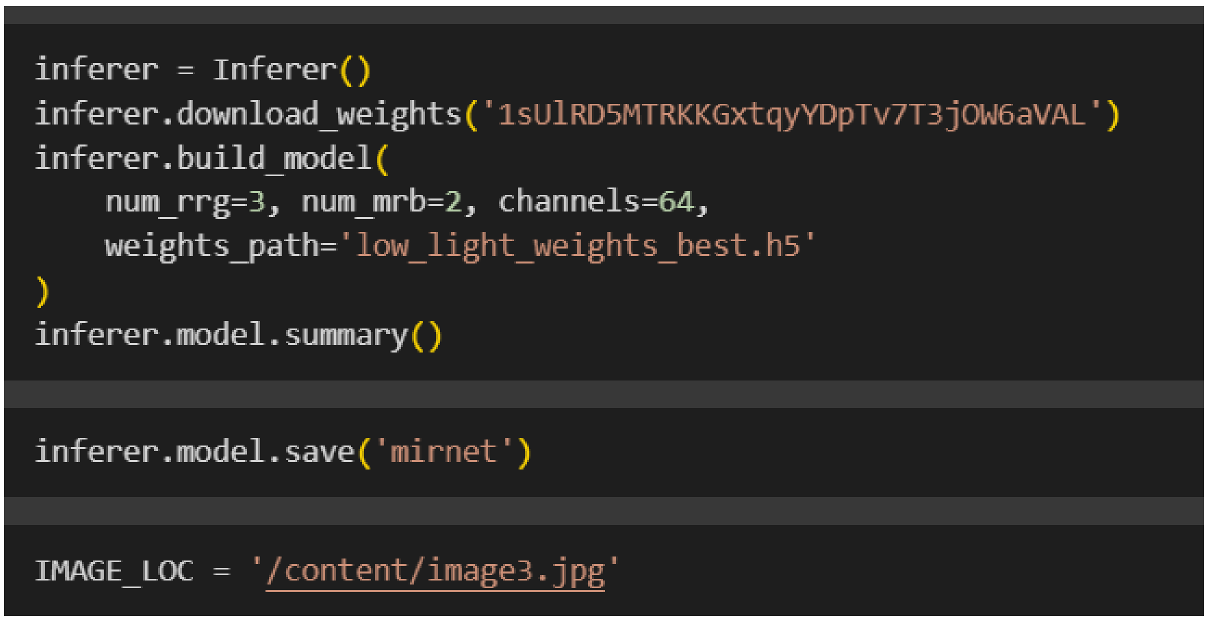
3,843 images (3x)

Dataset Split	TRAIN SET 88% 3363 Images	VALID SET 8% 321 Images	TEST SET 4% 159 Images

Preprocessing
Auto-Orient: Applied
Resize: Stretch to 640x640
Auto-Adjust Contrast: Using Histogram Equalization

Augmentations
Outputs per training example: 3
Flip: Horizontal
90° Rotate: Clockwise, Counter-Clockwise
Rotation: Between -45° and +45°
Shear: ±10° Horizontal, ±10° Vertical
Saturation: Between -30% and +30%
Brightness: Between -20% and +20%
Exposure: Between -15% and +15%
Noise: Up to 0.1% of pixels

Using the MIRNet model, we also applied low-light image enhancement. This is because some images have low light especially with those from the rainy video. We used the following snippet codes to implement the low-light enhancement model on the low-light images.



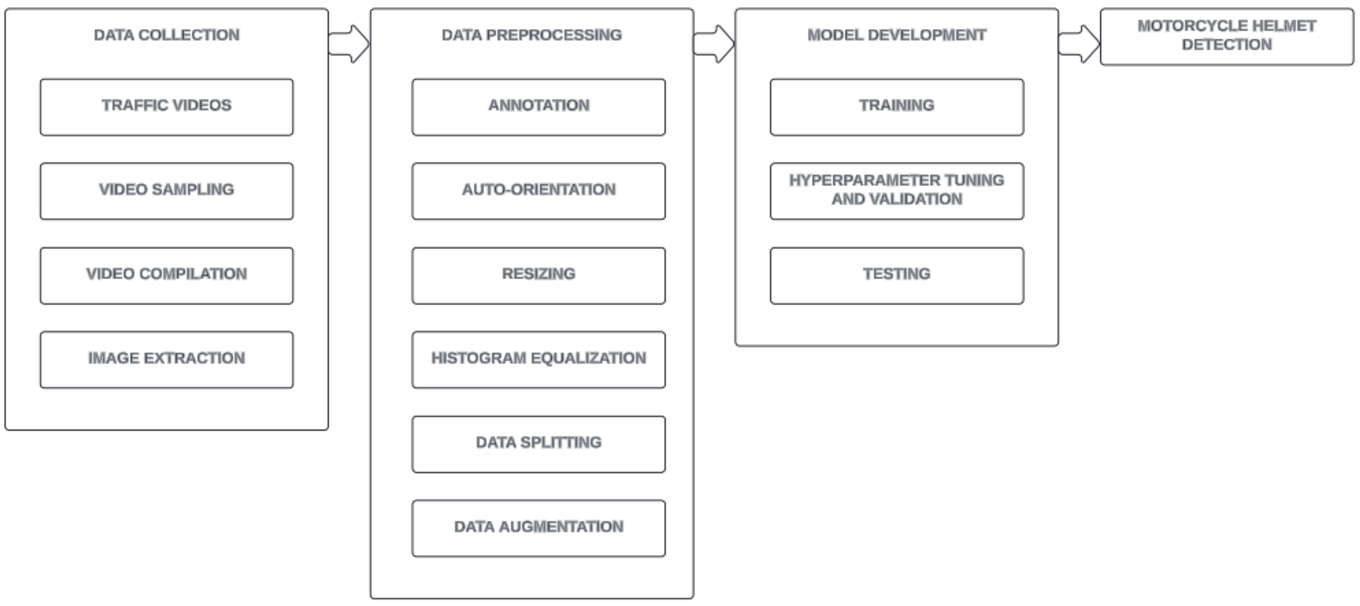
```
inferer = Inferer()
inferer.download_weights('1sUlRD5MTRKKGxtqyYDpTv7T3jOW6aVAL')
inferer.build_model(
    num_rrg=3, num_mrb=2, channels=64,
    weights_path='low_light_weights_best.h5'
)
inferer.model.summary()
```

```
inferer.model.save('mirnet')
```

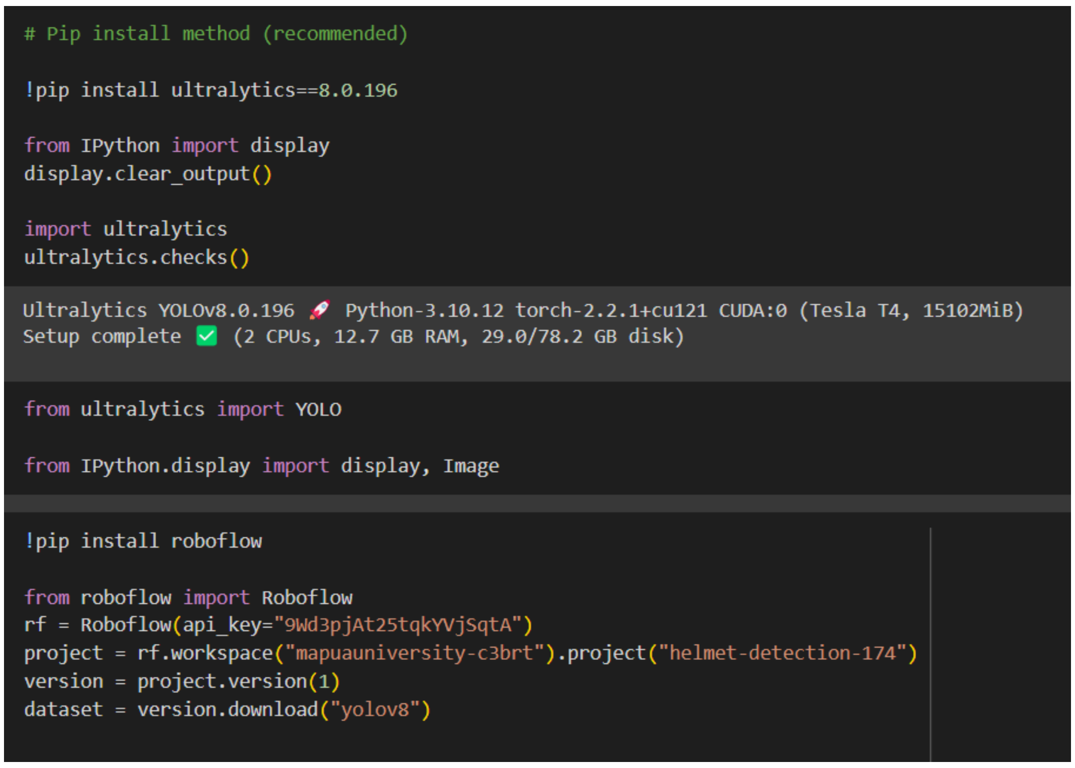
```
IMAGE_LOC = '/content/image3.jpg'
```

MODELLING

For our research objectives, we decided to detect and classify helmet types and detect and classify the number of riders on the motorcycle. Here we have our conceptual framework.



In our model development, we used the YOLOv8 algorithm. Here we imported all needed libraries to develop the model. We installed ultralytics for implementing YOLOv8 and roboflow so we can easily get our dataset.



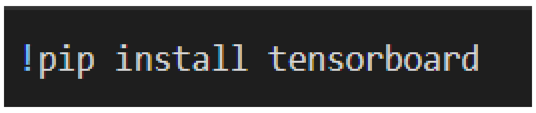
```
# Pip install method (recommended)

!pip install ultralytics==8.0.196

from IPython import display
display.clear_output()

import ultralytics
ultralytics.checks()
```

```
Ultralytics YOLOv8.0.196 🚀 Python-3.10.12 torch-2.2.1+cu121 CUDA:0 (Tesla T4, 15102MiB)
Setup complete ✅ (2 CPUs, 12.7 GB RAM, 29.0/78.2 GB disk)
```

```
from ultralytics import YOLO

from IPython.display import display, Image
```

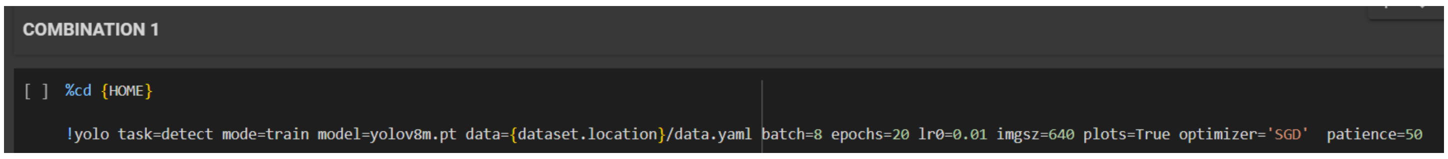
```
!pip install roboflow

from roboflow import Roboflow
rf = Roboflow(api_key="9Wd3pjAt25tqkYVjSqtA")
project = rf.workspace("mapuauniversity-c3brt").project("helmet-detection-174")
version = project.version(1)
dataset = version.download("yolov8")
```

We also installed tensorboard since we used this to visualize the training and validation process.



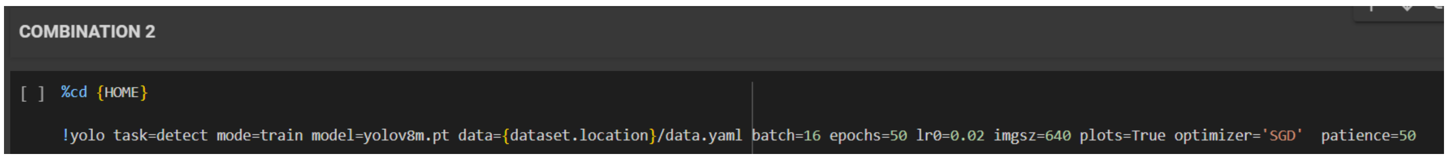
```
!pip install tensorboard
```

We used three different hyperparameter combinations in developing the model. Combination 1 used a batch size of 8, epoch of 20, and learning rate of 0.01. Combination 2 used a batch size of 16, epoch of 50, and learning rate of 0.02. Combination 3 used a batch size of 32, epoch of 100, and learning rate of 0.03



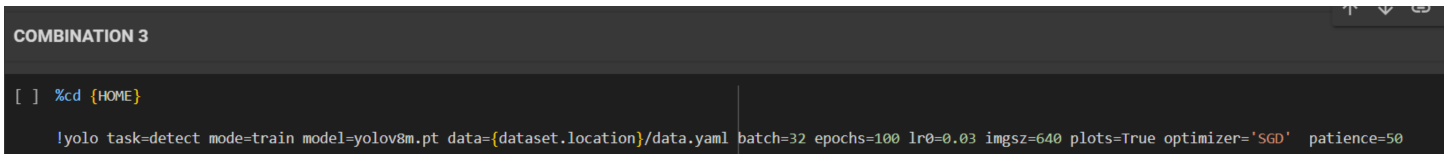
COMBINATION 1

```
[ ] %cd {HOME}

    !yolo task=detect mode=train model=yolov8m.pt data={dataset.location}/data.yaml batch=8 epochs=20 lr0=0.01 imgsz=640 plots=True optimizer='SGD'  patience=50
```



COMBINATION 2

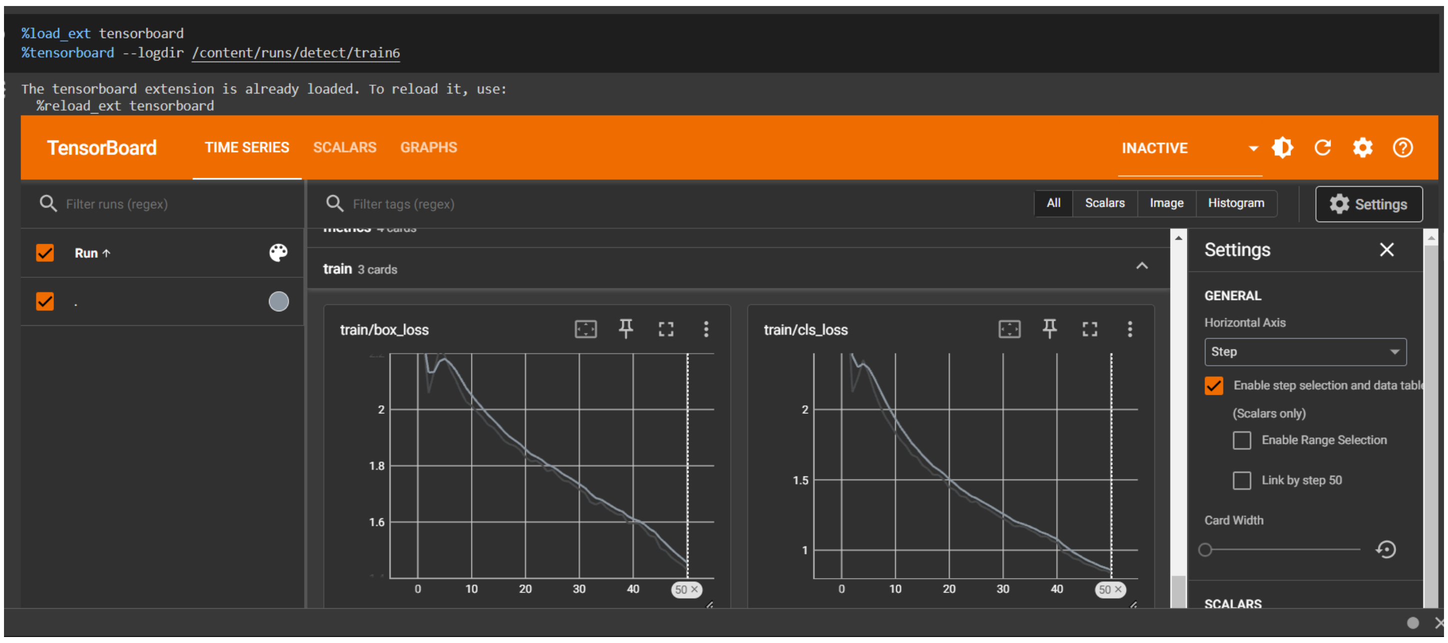
```
[ ] %cd {HOME}

    !yolo task=detect mode=train model=yolov8m.pt data={dataset.location}/data.yaml batch=16 epochs=50 lr0=0.02 imgsz=640 plots=True optimizer='SGD'  patience=50
```



COMBINATION 3

```
[ ] %cd {HOME}

    !yolo task=detect mode=train model=yolov8m.pt data={dataset.location}/data.yaml batch=32 epochs=100 lr0=0.03 imgsz=640 plots=True optimizer='SGD'  patience=50
```

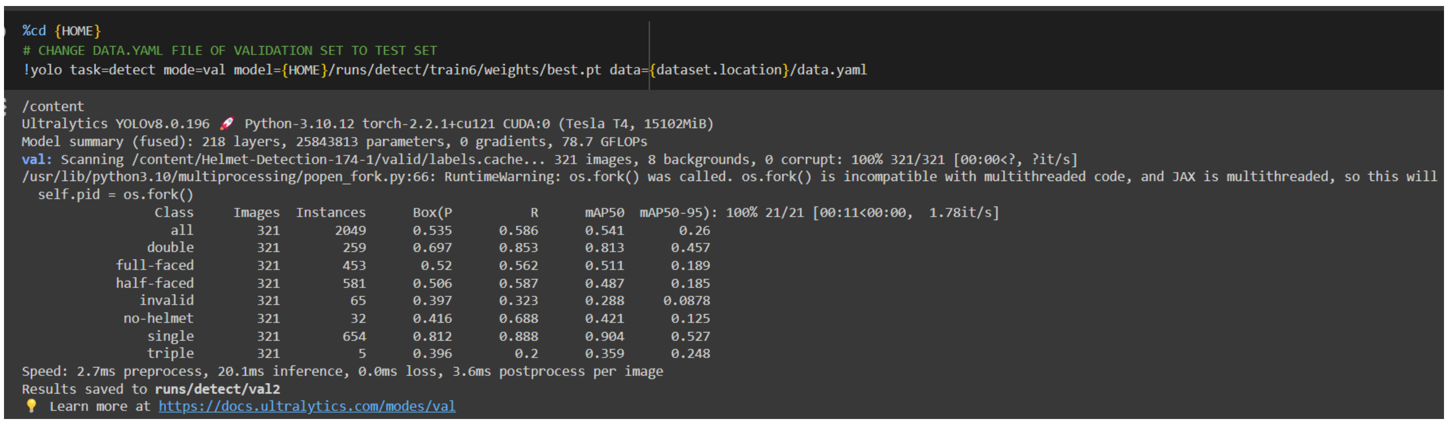
After we completed the training, we used tensorboard to visualize the results. We have the loss visualizations which tells how well the model learned during the training phase.



```
%load_ext tensorboard
%tensorboard --logdir /content/runs/detect/train6
```

```
The tensorboard extension is already loaded. To reload it, use:
  %reload_ext tensorboard
```

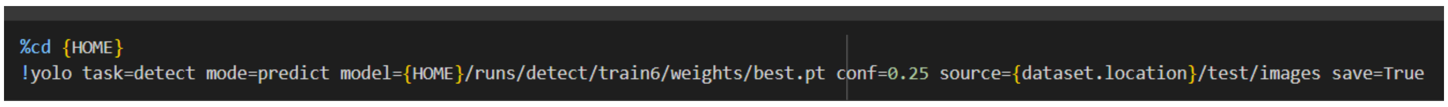
With the trained model, we used it to perform validation. We used the validation set as input to the model and evaluated its performance.



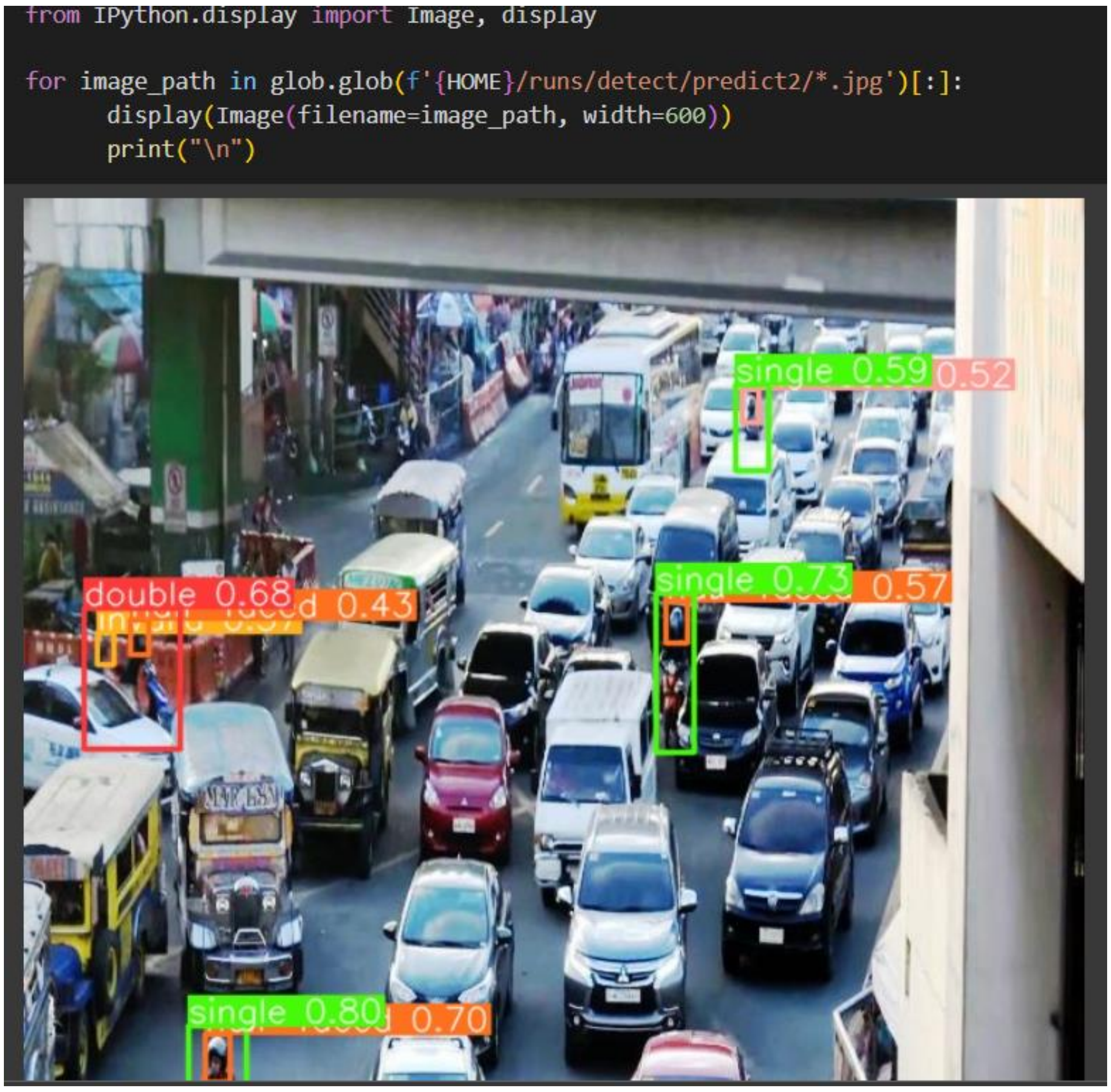
```
%cd {HOME}
# CHANGE DATA.YAML FILE OF VALIDATION SET TO TEST SET
!yolo task=detect mode=val model={HOME}/runs/detect/train6/weights/best.pt data={dataset.location}/data.yaml
```

```
/content
Ultralytics YOLOv8.0.196 🚀 Python-3.10.12 torch-2.2.1+cu121 CUDA:0 (Tesla T4, 15102MiB)
Model summary (fused): 218 layers, 25843813 parameters, 0 gradients, 78.7 GFLOPs
val: Scanning /content/Helmet-Detection-174-1/valid/labels.cache... 321 images, 8 backgrounds, 0 corrupt: 100% 321/321 [00:00<?, ?it/s]
/usr/lib/python3.10/multiprocessing/popen_fork.py:66: RuntimeWarning: os.fork() was called. os.fork() is incompatible with multithreaded code, and JAX is multithreaded, so this will
  self.pid = os.fork()
                 Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100% 21/21 [00:11<00:00,  1.78it/s]
                   all        321       2049      0.535      0.586      0.541       0.26
                double        321        259      0.697      0.853      0.813      0.457
            full-faced        321        453       0.52      0.562      0.511      0.189
            half-faced        321        581      0.506      0.587      0.487      0.185
               invalid        321         65      0.397      0.323      0.288     0.0878
             no-helmet        321         32      0.416      0.688      0.421      0.125
                single        321        654      0.812      0.888      0.904      0.527
                triple        321          5      0.396        0.2      0.359      0.248
Speed: 2.7ms preprocess, 20.1ms inference, 0.0ms loss, 3.6ms postprocess per image
Results saved to runs/detect/val2
💡 Learn more at https://docs.ultralytics.com/modes/val
```

After validation, we performed inference to see the results of the model. Here we used the test images.



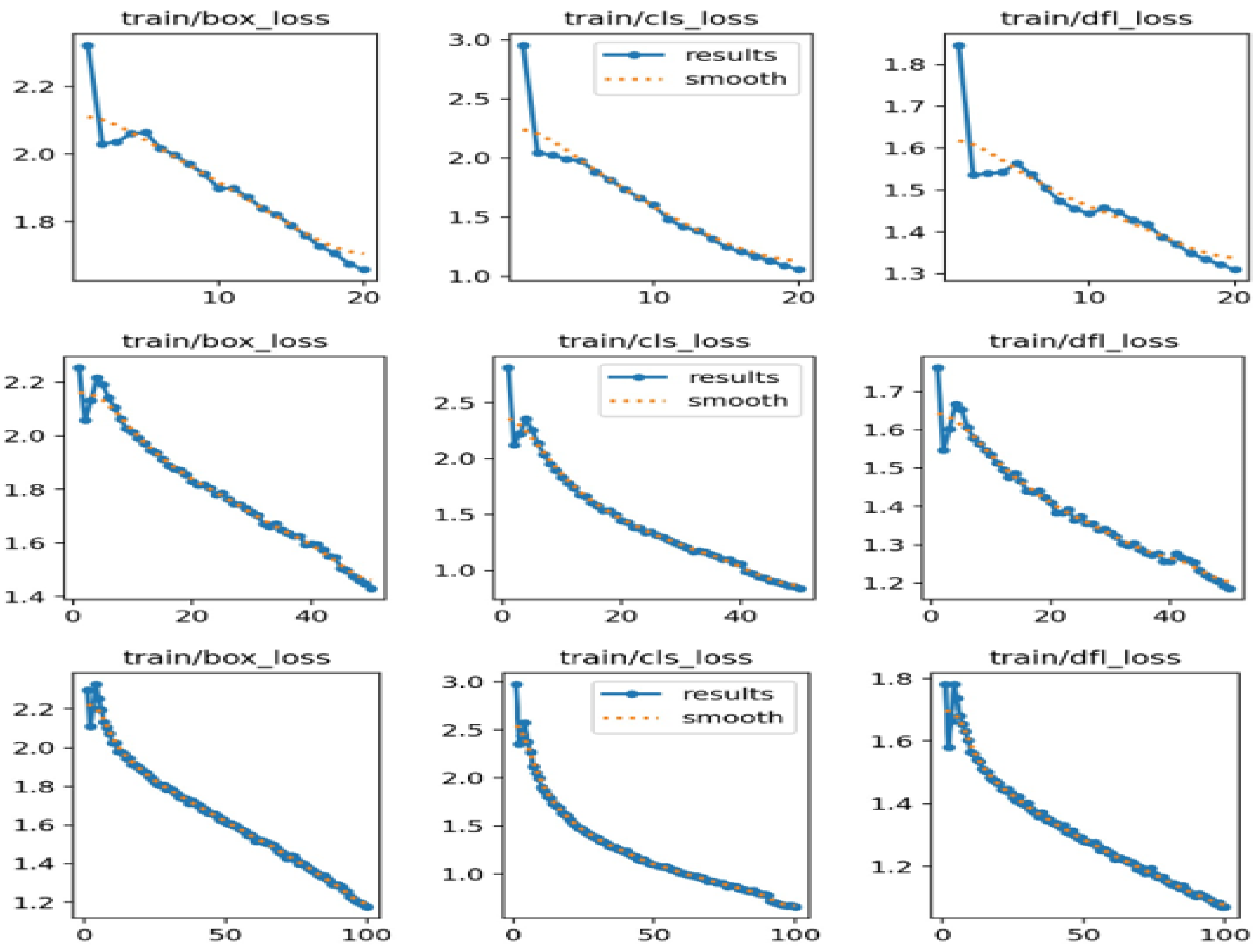
```
%cd {HOME}
!yolo task=detect mode=predict model={HOME}/runs/detect/train6/weights/best.pt conf=0.25 source={dataset.location}/test/images save=True
```



```
from IPython.display import Image, display

for image_path in glob.glob(f'{HOME}/runs/detect/predict2/*.jpg')[:]:
      display(Image(filename=image_path, width=600))
      print("\n")
```

RESULTS

We used the validation set to study and evaluate the performance of the model with the different hyperparameters. The loss visualizations we have attained shows an overall smooth descent with noticeable fluctuations at the start of the first ten epochs. As the losses increase, the model's performance continues to decline, indicating that it improves over time during training. The study used and evaluated epochs 20, 50, and 100, but the loss illustrations show that the graph continues to descend. This means that the model improvements can continue even after the 100th epoch.

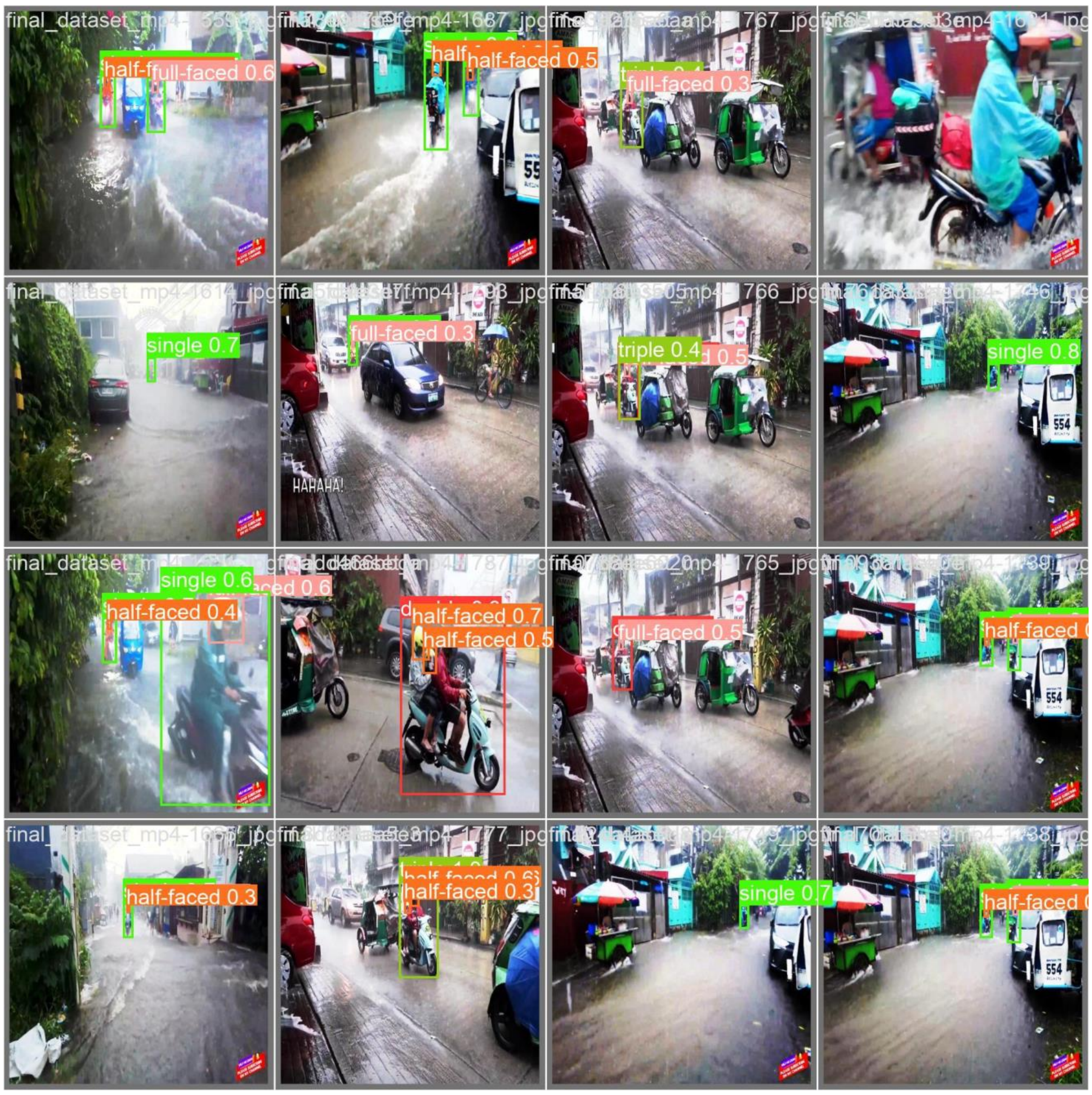


We obtained the performance metric values of the models. We used mAP@50, precision, and recall as the metrics for our study. Observing the mAP@50 values, the model's performance improves as the batch size, epoch, and learning rate increase. The combination 3 model had the highest metric scores: 57.6% for mAP@50, 58.5% for precision, and 62.9% for recall. The model with hyperparameter combination 3 outperformed the combination 1 and 2 models in terms of mAP@50 by 2.5% and 3.5%, respectively. The combination 3 model improved precision scores by 2.8% and 5% over the combination 1 and combination 2 models, respectively. Recall scores were also the highest for the combination 3 model, up 7.9% from the combination 1 model and 4.3% from the combination 2 model. Meanwhile, the models' performance in classifying helmet types and motorcycle riders varies. In terms of helmet types, the models are fairly effective at detecting full-faced helmets, but they could be better at detecting half-faced, invalid, and no-helmet classes. For riders, the models have medium to good efficacy in detecting single and double riders, but less so for triple riders. Overall, the validation results show that the model with the hyperparameter combination of a batch size of 32, an epoch of 100, and a learning rate of 0.03 outperformed all three specified combinations for the object detection task. However, the validation scores indicate that there is still room for improvement, as the mAP@50 scores only reached 60%. Since the loss graphs show that the model's losses continue to decline after epoch 100, increasing the value for epoch during training can improve the model's performance by giving it more time to learn from the input data. Along with the epoch, modifying and increasing the value of the learning rate may improve model performance by speeding up convergence and generating more significant updates to the model's weights.

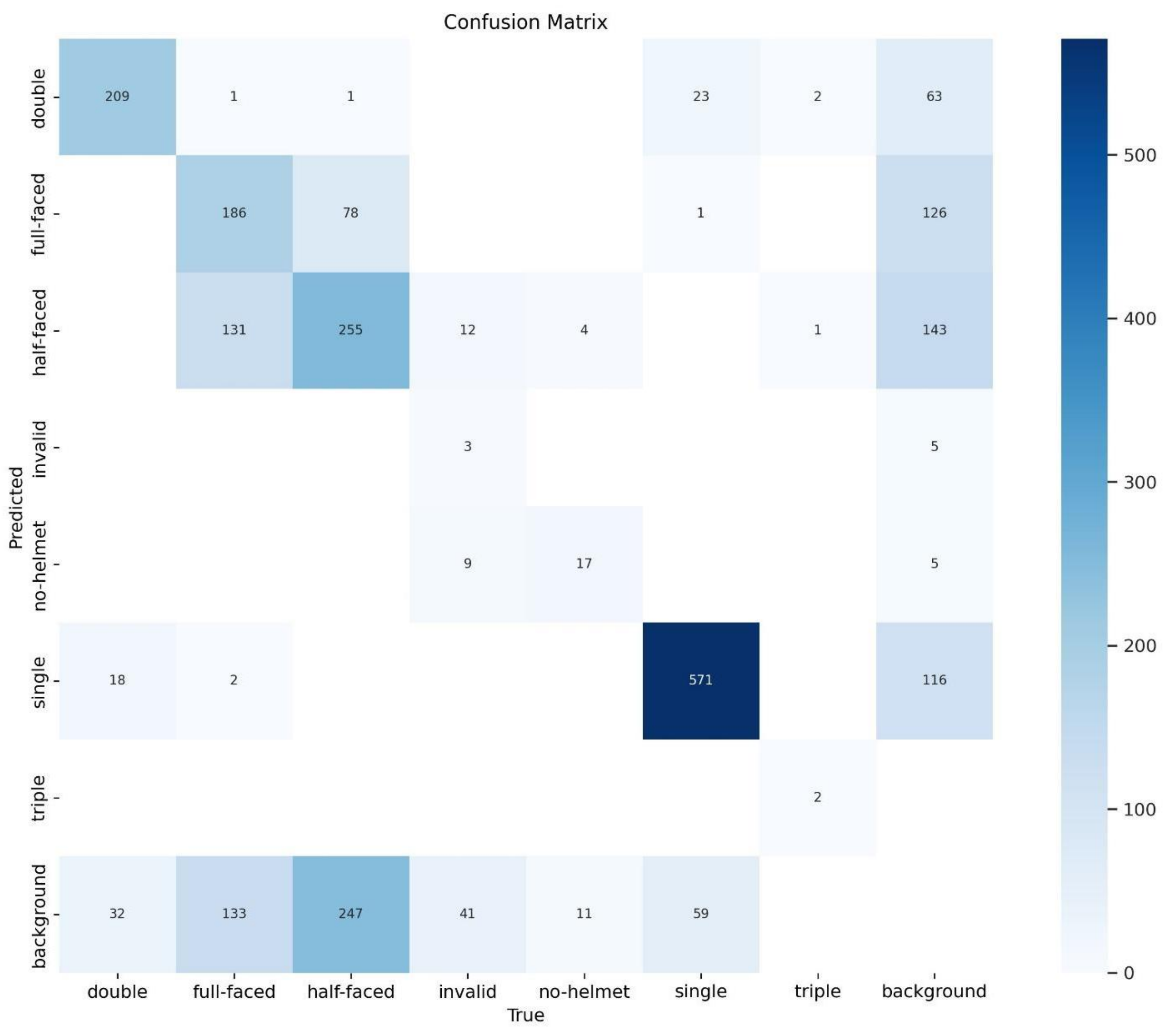
Combination 1	Batch Size (8)	Epoch (20)	Learning Rate (0.01)
Classes	mAP@50	Precision	Recall
All	0.551	0.557	0.55
Full-faced	0.527	0.51	0.532
Half-faced	0.441	0.496	0.547
Invalid	0.209	0.313	0.0615
No-helmet	0.429	0.524	0.562
Single	0.903	0.789	0.891
Double	0.804	0.69	0.853
Triple	0.544	0.576	0.4
Combination 2	Batch Size (16)	Epoch (50)	Learning Rate (0.02)
Classes	mAP@50	Precision	Recall
All	0.541	0.535	0.586
Full-faced	0.511	0.52	0.562
Half-faced	0.487	0.506	0.587
Invalid	0.288	0.397	0.323
No-helmet	0.421	0.416	0.688
Single	0.904	0.812	0.888
Double	0.813	0.697	0.853
Triple	0.359	0.396	0.2
Combination 3	Batch Size (32)	Epoch (100)	Learning Rate (0.03)
Classes	mAP@50	Precision	Recall
All	0.576	0.585	0.629
Full-faced	0.515	0.506	0.594
Half-faced	0.448	0.469	0.542
Invalid	0.217	0.289	0.246
No-helmet	0.379	0.362	0.562
Single	0.886	0.762	0.876
Double	0.792	0.71	0.826
Triple	0.799	1	0.759

We also have the confusion matrix for each model combination. It shows the performance of the model for classifying the detected objects. The detection model correctly classified a limited number of instances of the specific helmet type. Although many full-faced and half-faced helmets were correctly predicted, others remained to be identified. Many full-faced helmets were mistaken for half-faced helmets, and vice versa. This can be attributed to the helmets' similar appearance, with the only difference being the chin protection. Some helmet images are small and unclear, leading to the model's misclassification. Some images show the helmet in both the front and side views, which reduces visibility of the chin area and may lead to misclassification. Instances of the invalid class were also miscategorized. When viewed through images, invalid helmets resemble full-faced and half-faced helmets. Meanwhile, for the classes that specify the number of riders in a motorcycle, the model was able to classify the images correctly, with fewer misclassifications than for the helmet type. The model accurately classified triple riders. Also, single and double riders had a significantly higher rate of correctly classified cases. However, there were some cases where single riders were classified as doubles. Some instances in the double class were classified as single, possibly due to limited visibility. In this case, some double riders resembled single riders because the second rider was obscured by the first rider, revealing only small portions of the head or body. This could be attributed to instances where two separate single-rider motorcycles were too close to each other, resulting in misclassification. However, it is important to note that the model incorrectly classified a large number of object instances as background. This means that the model will occasionally fail to detect the presence of an object in an image. This could be due to the limited visibility of the objects in the images. This can be seen in traffic images, where motorcycle riders appear small and far away from the camera's point of view. As a result, some object instances went undetected despite their true presence. Overall, the model shows promising but limited results in correctly detecting objects within each class. There is still room to improve the model's performance capacity.

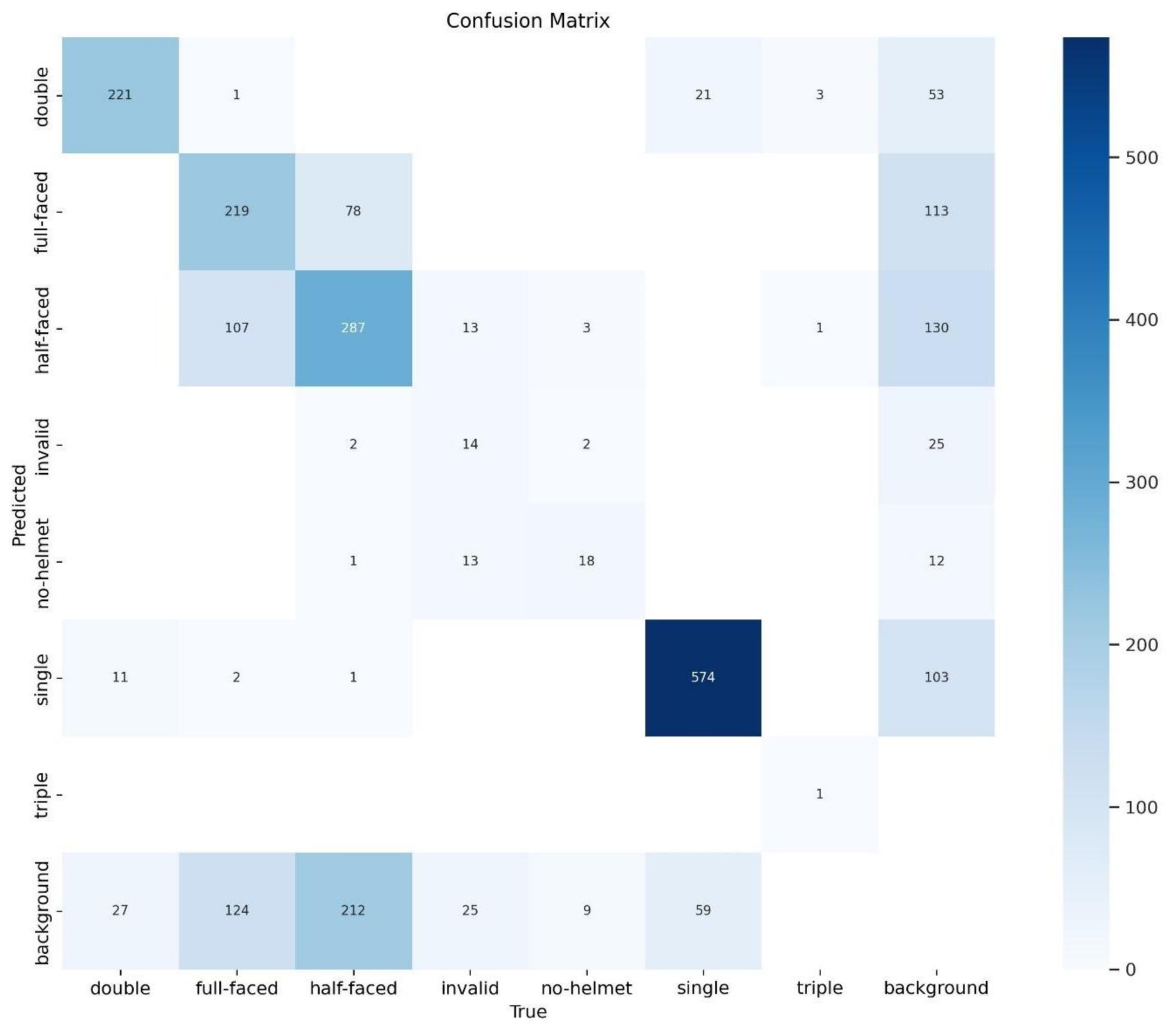
We have an example of the predicted output of the model. We can see in one image that the motorcycle rider and helmet can be clearly seen but is not detected by the model indicating that the model still struggles in the detection task.



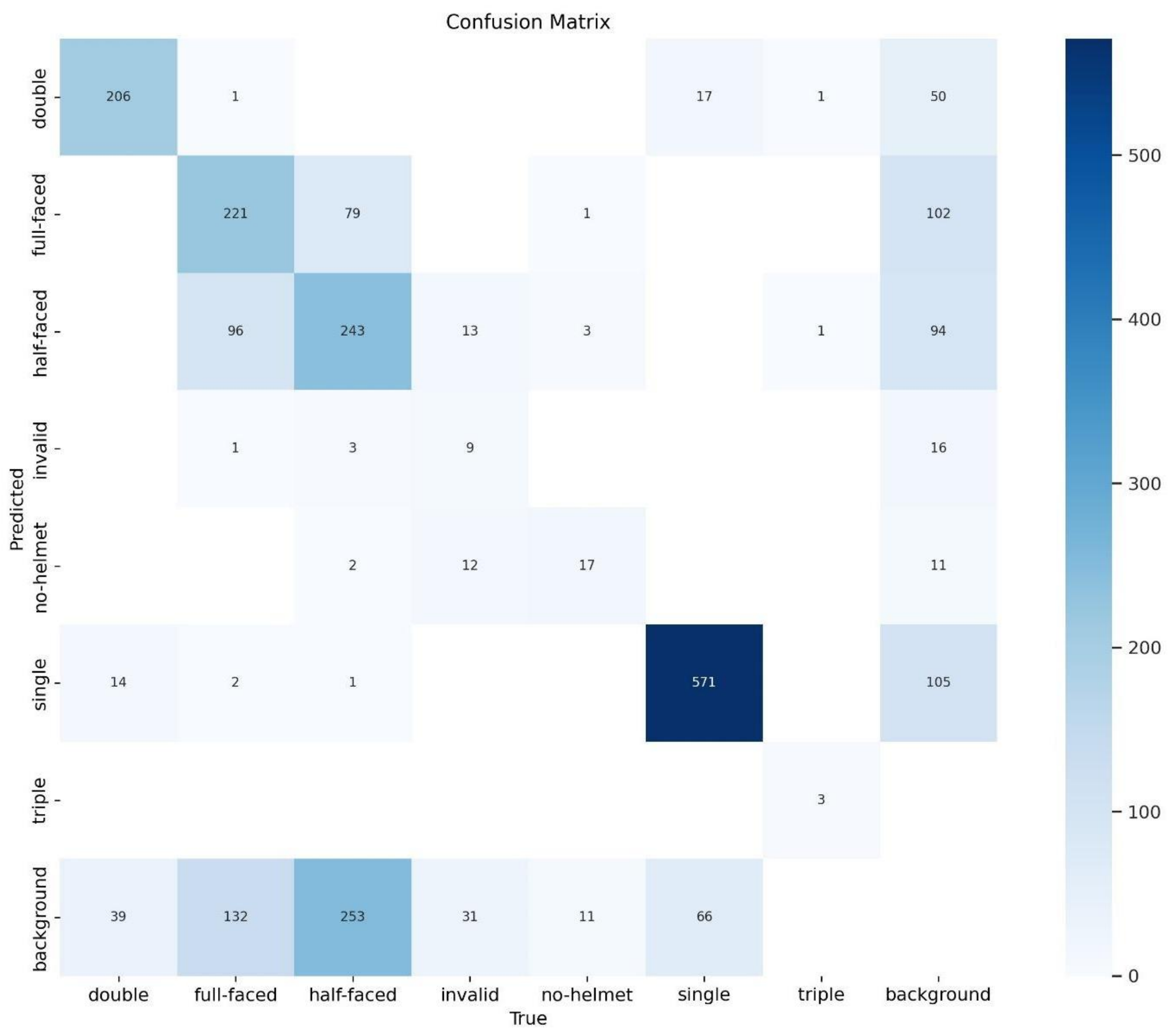
Combination 1 confusion matrix



Combination 2 confusion matrix

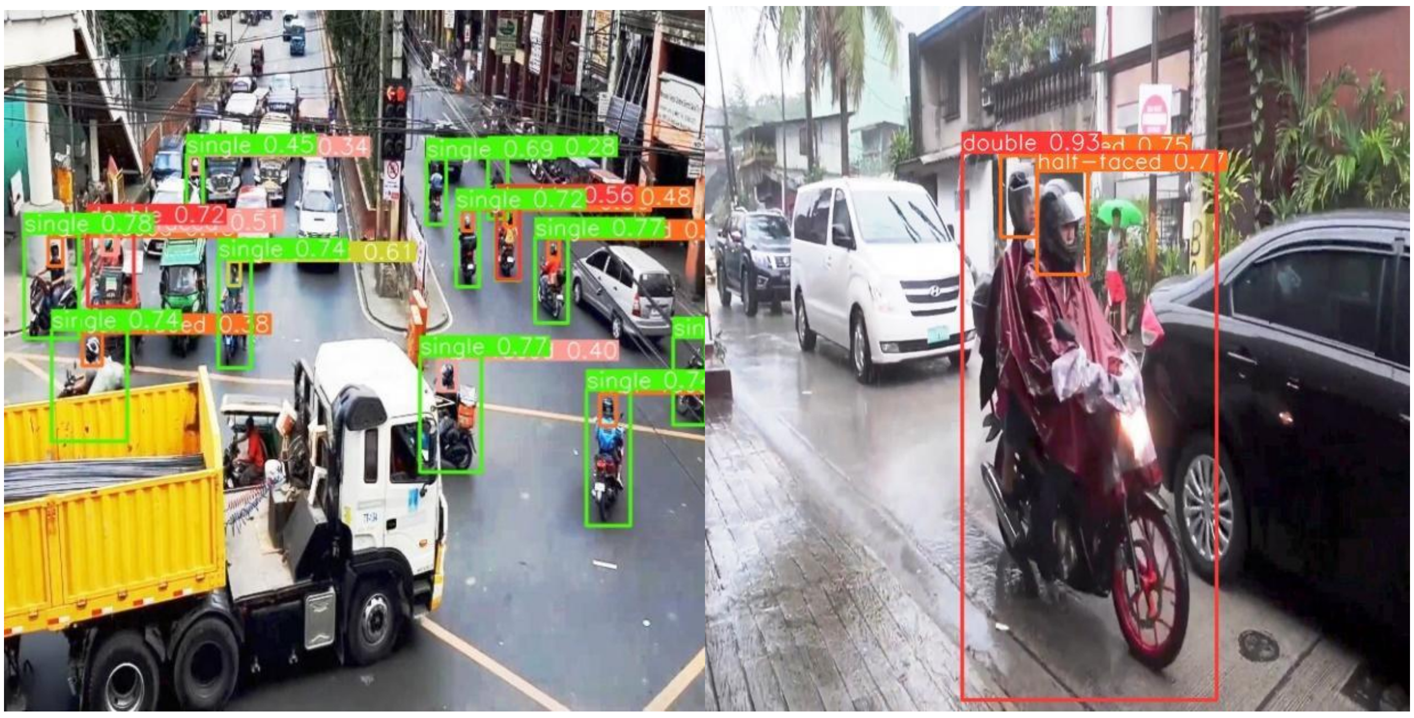


Combination 3 confusion matrix



Using our validation results, we know that the model with combination 3 where batch size is 32, epoch is 100, and learning rate is 0.03 has the best performance. So we used this model for testing. With the test images, the model performed at an average level. The model had a mAP@50 of 57.7%, a precision score of 49.3%, and a recall of 59.6%. The obtained metric scores show that the model struggles with the detection task but has the ability to do so. In terms of metrics per class, the model performs well in detecting full-faced helmet types, with a mAP@50 of 81.6%, precision of 77.2%, and recall of 78.5%. Meanwhile, the model performs well in detecting double riders, with a mAP@50 of 86.6%, precision of 80.6%, and recall of 87.9%. This demonstrates the model's potential for the object detection task. However, the model still needs to improve its performance in observing metric scores to detect the other classes. The lower scores indicate that the model's performance can be improved further. This can be accomplished by adjusting the hyperparameters and assessing the effects on the model. Furthermore, feeding more high-quality data and instances of each class can improve the model's performance. Overall, the model delivered impressive results, but there is still room for improvement.

Class	Images	Instances	mAP@50	Precision	Recall
All	159	1007	0.577	0.493	0.596
Full-faced	159	130	0.816	0.772	0.785
Half-faced	159	200	0.473	0.533	0.525
Invalid	159	316	0.462	0.548	0.456
No-helmet	159	26	0.229	0.345	0.192
Single	159	12	0.693	0.593	0.833
Double	159	321	0.866	0.806	0.879
Triple	159	2	0.499	0.533	0.533



Although the dataset contained a wide range of conditions due to the images' different angles, weather, and lighting, as well as the different sizes of the object instances, it was imbalanced, with some classes having more total instances than others. We can address this by improving the dataset. We need to collect data with equal and sufficient representation for all classes to reduce the impact of an imbalanced dataset on model performance. This could be a way to improve the performance of the model in the future.

REFERENCES

Walking Echo (2023). *When Rain Gets Wild: Jaw-Dropping Scenes During Super Heavy Rainfall | Philippines | [4K]*. YouTube. https://www.youtube.com/watch?v=UQWsOPj623s

PHiLiPPiNE SCENES (2018). *RAiNY METRO MANiLA STREETS AS SUPER TYPHOON OMPOMG (MANGKHUT) PASSES*. YouTube. https://www.youtube.com/watch?v=YIWSIU4bl70

RAILWAY UNIVERSE | EISENBAHNUNIVERSUM (2020). Crazy Traffic Metro Manila. YouTube. https://www.youtube.com/watch?v=ydky3fx9ftk

KW Photography (2020). Crazy Traffic in Manila Philippines. YouTube. https://www.youtube.com/watch?v=AESq7KMvd9Q

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

Zagami, D. (2022). *hvas-face-person-recognition*. GitHub. https://github.com/davidezagami/hvas-face-person-recognition

Moin, M. (2022). *What is DeepSORT and how to implement YOLOv7 Object Tracking using DeepSORT*. Medium. https://medium.com/@m.moinfaisal/what-is-object-tracking-and-why-deepsort-and-how-to-implement-yolov7-object-tracking-using-deepsort-f0c952f89b06

Li, X., Li. X, Han, B., Wang, S., and Chen, K. (2023). *Application of EfficientNet and YOLOv5 Model in Submarine Pipeline Inspection and a New Decision-Making System*. Water. 2023; 15(19):3386. https://doi.org/10.3390/w15193386