Helmet Detection and Recognition using YOLO

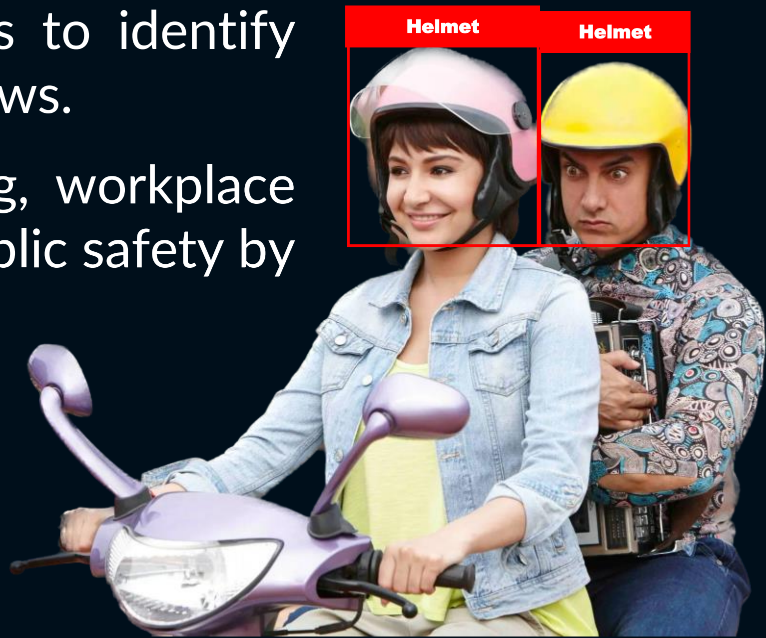
Monika Agarwal
Prateek Agrawal
Rishi Shrivastava
Smita Damani
Snehal Bailmare

Introduction

- Over **1.2 million people die** each year on the world's roads, and between **20 and 50 million suffer non-fatal injuries**.
- Motorcycles are a predominant method of transport in many countries, thanks to their low price and operation cost compared to other vehicles.
- Despite the benefits of motorcycles, safety is often neglected, especially by motorcyclists who are caught riding without a helmet.
- Approximately **41% of motorcycle drivers who die in accidents are not wearing a helmet**.

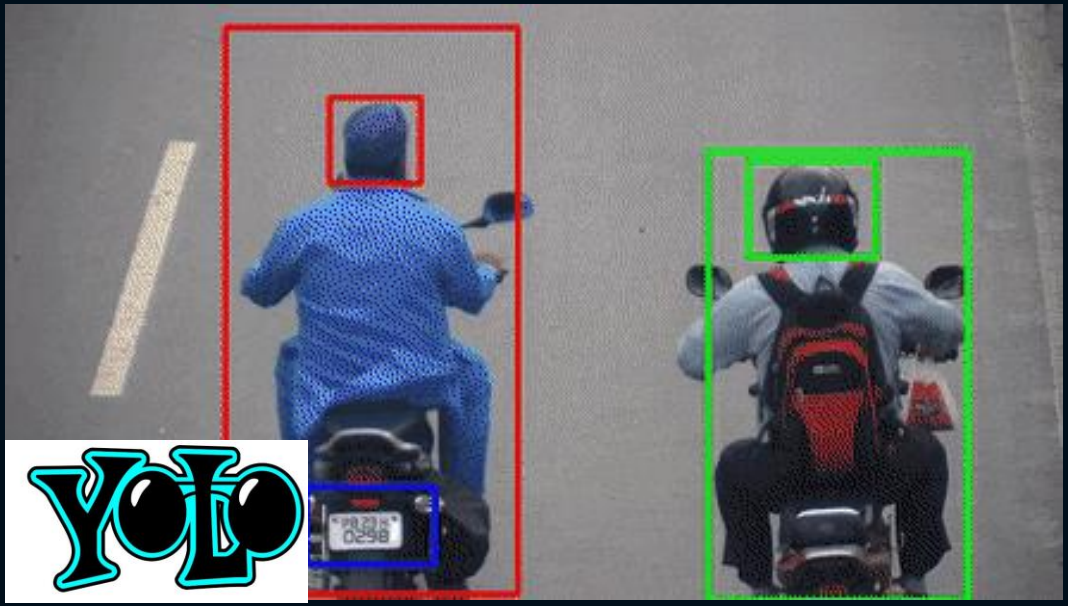
Business Problem

- Motorcyclists, cyclists, and pedestrians are among the most vulnerable groups, bearing a disproportionate burden of road traffic incidents.
- In India alone, **53,019 people died in road accidents** involving two-wheelers in 2019, with **71% of those fatalities being riders and pillion passengers**.
- Of these fatalities, approximately **4 motorcyclist riders died every hour** due to not wearing helmets.
- To tackle this problem, the helmet detection project aims to identify motorcyclists and help government bodies enforce stricter laws.
- The project has potential applications in traffic monitoring, workplace safety, and sporting events, and can significantly improve public safety by promoting the use of helmets among motorcyclists.



Solution

- Our project aims to address the issue of motorcyclists riding without helmets by implementing a methodology for automatic helmet detection.
- The approach utilizes images and videos of traffic on public roads as input data for the detection process.
- We have used YOLOV5 algorithm, a deep learning algorithm, on the transformed images to check if the rider is wearing a helmet or not.
- The algorithm accurately identifies the presence or absence of helmets on the riders, allowing for efficient enforcement of helmet laws and promoting safety on the roads.



Data Sources

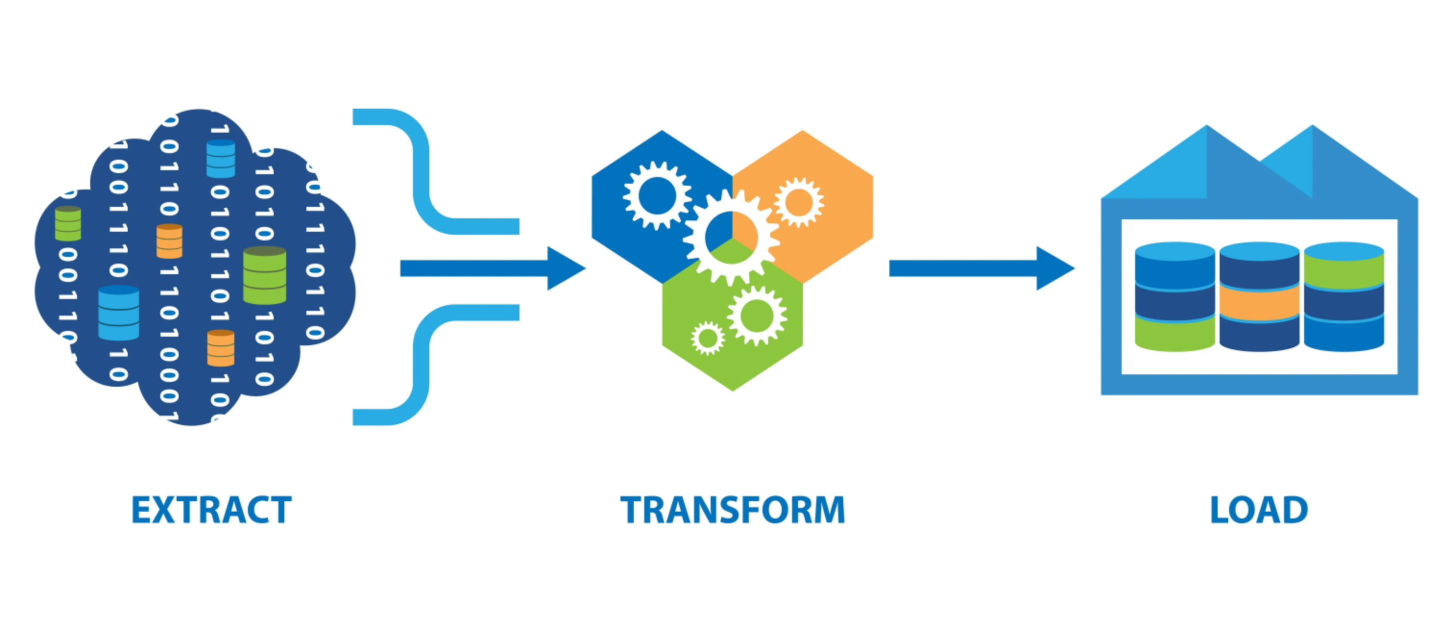
Helmet Dataset from osf.io

Contributors - **Hanhe Lin and Felix Wilhelm Siebert**
Link - **https://osf.io/4pwj8/**
Description – Data consist of **10,000 +** images with approximate size of **25GB.**

Helmet Dataset from Kaggle

Contributors - viklundvisuals
https://www.kaggle.com/datasets/andrewmvd/helmet-detection
Description – This dataset contains 764 images.

Extract, Transform and Load



IMAGES/VIDEOS

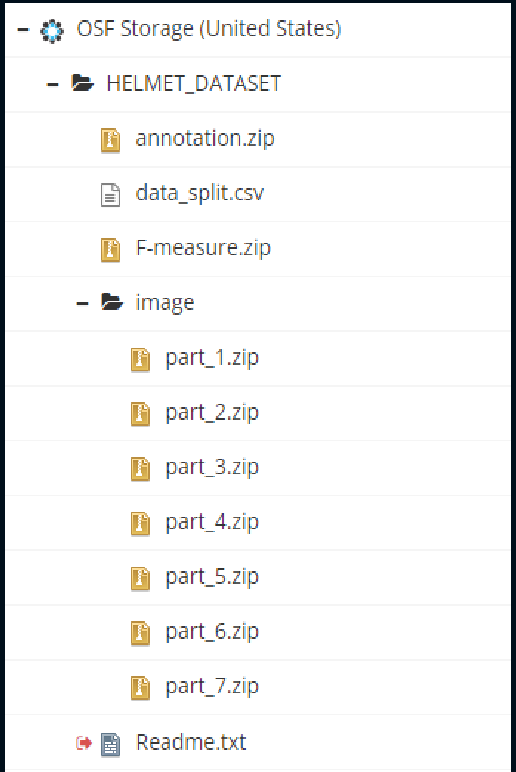
DATA PRE-PROCESSING

AMAZON S3

Data Extraction

- Data extraction for image datasets involves collecting and organizing images from various sources such as websites, social media platforms, or image databases.
- Major Chunk of our image data and annotation files were extracted from the OSF helmet dataset website.

- The Dataset was divided Into 7 parts, which was further subdivided into 100 subfolders and the annotations folder consisted of .csv files having the labels for each image.



Data Transformation

Data transformation is the process of converting raw data into a format that can be easily analysed and used for various purposes.

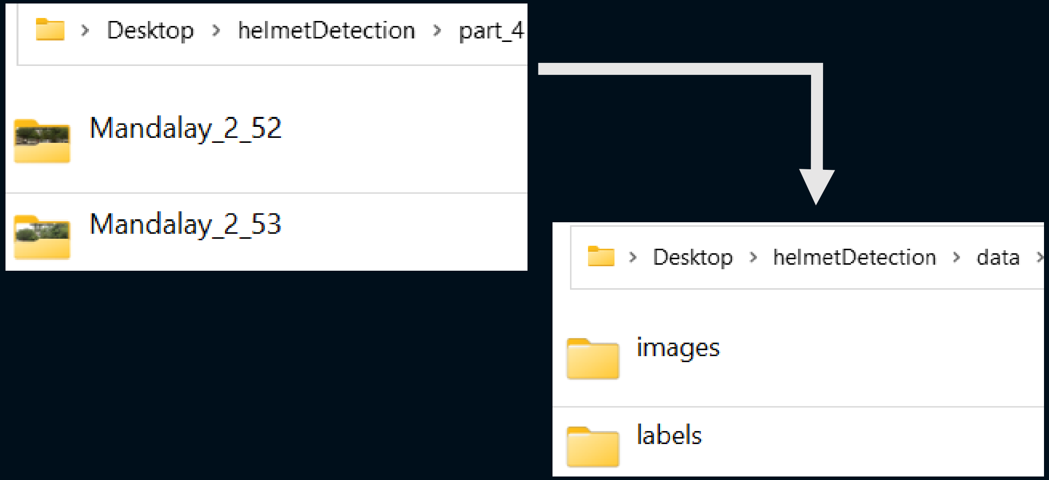


IMAGE TRANSFORMATION

Multiple Image subfolders → Rename images and Merge data → Split into Training and Validation Data

ANNOTATION FILE TRANSFORMATION

.csv Annotation File → Rename Frame_ids and merge .csv filles → Convert .csv to labels (.txt) file

track_id	frame_id	x	y	w	h	label
_84whllmr	1	61	616	94	146	DHelmet
_cp61cy0n	1	247	607	133	158	DHelmetP1Helmet
_kia64fsup	1	642	597	147	171	DHelmet

rs › sdama › Desktop › helmetDetection › data › labels › train › Mandalay_2_531.txt

```
0 0.05625 0.637962962962963 0.048958333333333333 0.13518518518518519
1 0.16302083333333334 0.6351851851851852 0.06927083333333334 0.14629
```

Load Dataset

- Images and labels are uploaded on S3, they can be accessed and processed by various applications and tools, including machine learning algorithms and image recognition systems using Boto3.

python + Boto 3 = Amazon S3

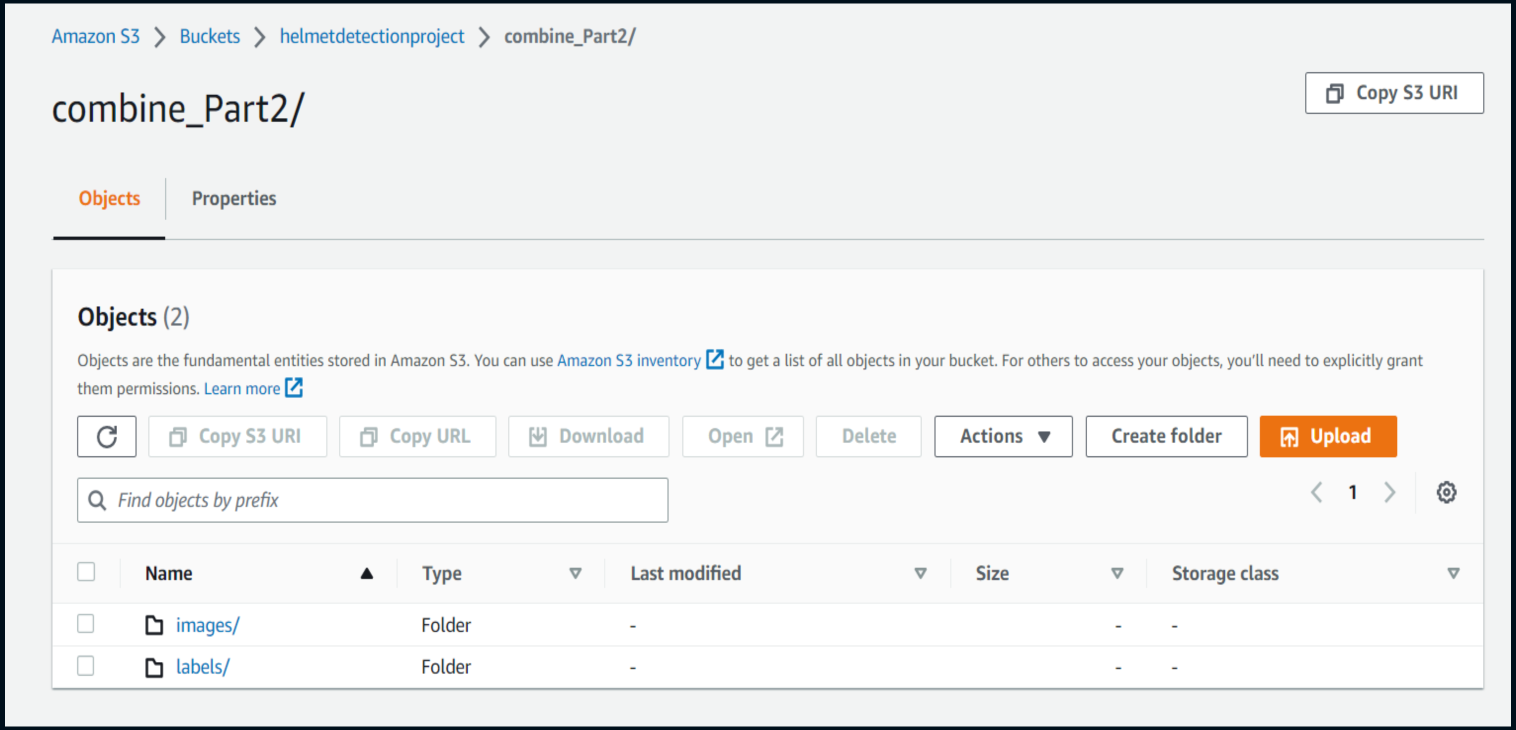
Steps:

✓Bucket Creation

✓Uploading the data on S3 with global permissions.

✓Generated Access keys and secret tokens.

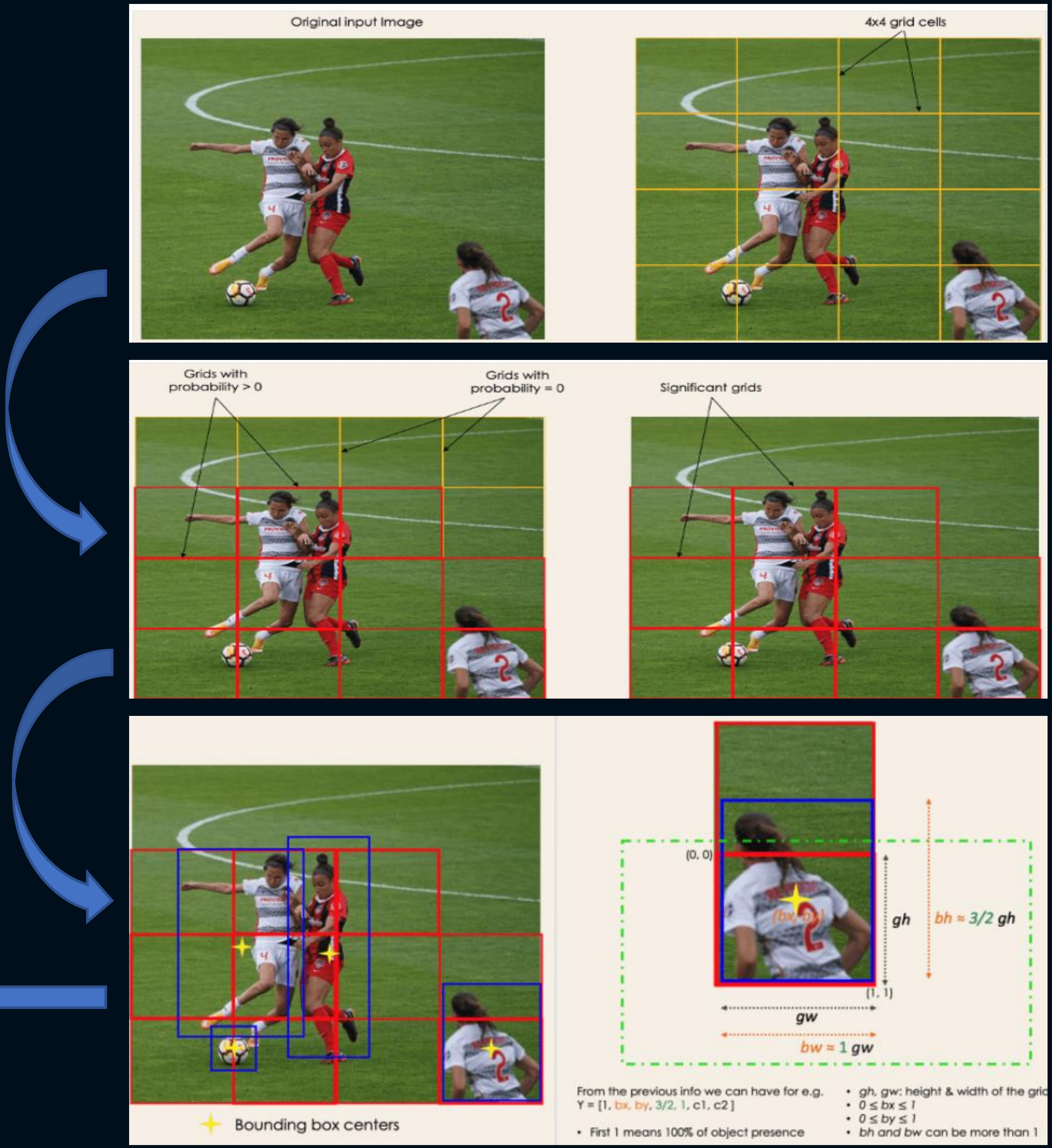
✓Boto3 bridges the access with the application with client access of S3 bucket and resources.

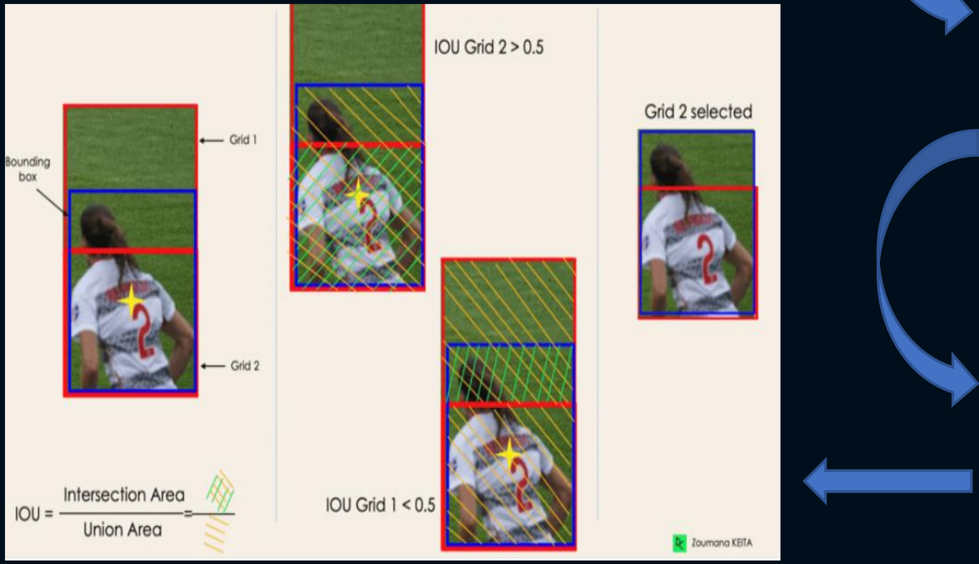


How does YOLO Algorithm work?

Algorithm approaches:

1) Residual blocks
2) Bounding box regression
3) Intersection Over Union (IOU)
4) Non-Maximum Suppression





How did we train our model?

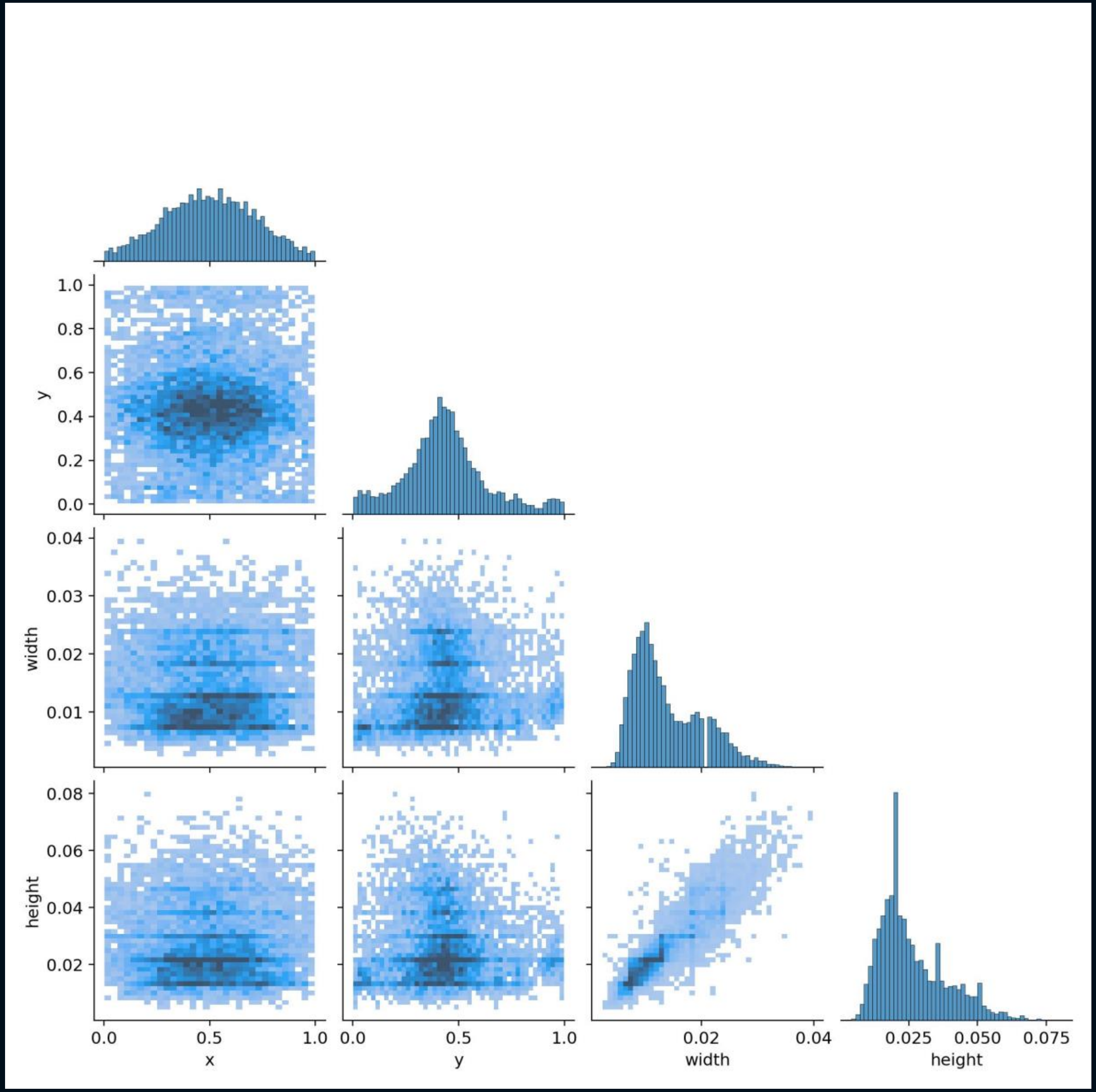
1. **Data Preparation**
2. **Model Configuration** - Configure the YOLOv5 model to detect helmets by modifying the YAML configuration file to set the number and the names of the classes.

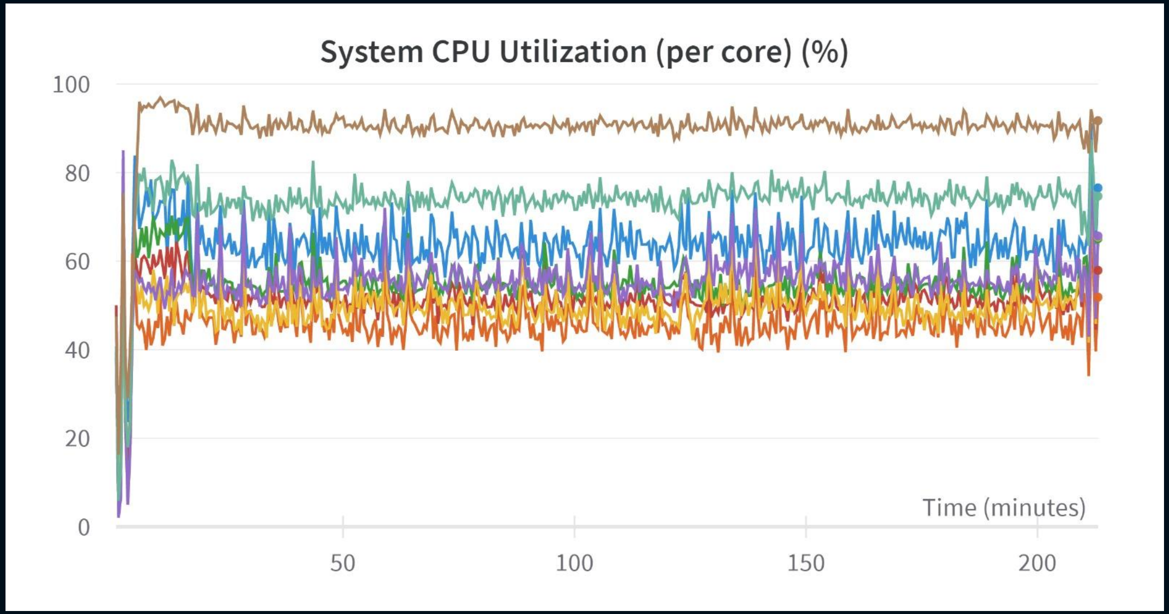
_to_one_place.py | yolov5-helmet-detection-train-and-inference.ipynb | yolov5-helmet-detection-train-and-inference (3).ipynb | final.ipynb

```
{ names: [ Helmet, NoHelmet, NoHelmetP0NoHelmetP1NoHelmetP2NoHelmet, NoHelmetP1NoHelmetP2NoHelmet ],
  nc: 4,
  train: data/images/train,
  val: data/images/val }
```

3. **Model Training:** Train the YOLOv5 model on the prepared dataset using the command "python train.py --img {image_size} \
   --batch {BATCH_SIZE} \
   --epochs {EPOCHS} \
   --data {path_to_data} \
   --cfg yolov5s.yaml \
   --weights yolov5s.pt"

```
YOLOv5s summary: 214 layers, 7033114 parameters, 7033114 gradients, 16.0 GFLOPs
Transferred 342/349 items from yolov5s.pt
optimizer: SGD(lr=0.01) with parameter groups 57 weight(decay=0.0), 60 weight(decay=0.0005), 60 bias
train: Scanning C:\Users\Prateek\Desktop\Project-BigData\final\final\final\yolov5\data\labels\train.cache... 1237 images, 0 backgrounds, 0 corrupt: 100%|██████████| 1237/1237 [00:00<?, ?it/s]
val: Scanning C:\Users\Prateek\Desktop\Project-BigData\final\final\final\yolov5\data\labels\val... 310 images, 0 backgrounds, 0 corrupt: 100%|██████████| 310/310 [00:26<00:00, 11.68it/s]
val: Scanning C:\Users\Prateek\Desktop\Project-BigData\final\final\final\yolov5\data\labels\val...:   0%|          | 0/310 [00:00<?, ?it/s]
val: New cache created: C:\Users\Prateek\Desktop\Project-BigData\final\final\final\yolov5\data\labels\val.cache
       0/49         0G    0.08459    0.05331     0.0455         35        640: 100%|██████████| 78/78 [29:07<00:00, 22.41s/it]
AutoAnchor: 6.32 anchors/target, 1.000 Best Possible Recall (BPR). Current anchors are a good fit to dataset
Plotting labels to yolov5s_custom\exp6\labels.jpg...
Image sizes 640 train, 640 val
Using 8 dataloader workers
Logging results to yolov5s_custom\exp6
Starting training for 50 epochs...
      Epoch    GPU_mem   box_loss   obj_loss   cls_loss  Instances       Size
  0%|          | 0/78 [00:00<?, ?it/s]
                 Class     Images  Instances          P          R      mAP50   mAP50-95:  20%|██        | 2/10 [00:23<01:33, 11.66s/it]WARNING  NMS time limit 2.100s exceeded
                 Class     Images  Instances          P          R      mAP50   mAP50-95:  30%|███       | 3/10 [00:35<01:21, 11.67s/it]WARNING  NMS time limit 2.100s exceeded
                 Class     Images  Instances          P          R      mAP50   mAP50-95:  50%|█████     | 5/10 [00:59<00:59, 11.96s/it]WARNING  NMS time limit 2.100s exceeded
                 Class     Images  Instances          P          R      mAP50   mAP50-95:  70%|███████   | 7/10 [01:22<00:35, 11.84s/it]WARNING  NMS time limit 2.100s exceeded
                 Class     Images  Instances          P          R      mAP50   mAP50-95: 100%|██████████| 10/10 [01:53<00:00, 11.39s/it]
                   all        310       1090      0.523      0.236      0.117     0.0393
      Epoch    GPU_mem   box_loss   obj_loss   cls_loss  Instances       Size
       1/49         0G    0.06792     0.0367    0.04078         33        640: 100%|██████████| 78/78 [27:47<00:00, 21.37s/it]
                 Class     Images  Instances          P          R      mAP50   mAP50-95: 100%|██████████| 10/10 [01:36<00:00,  9.61s/it]
                   all        310       1090      0.542      0.335      0.175     0.0626
      Epoch    GPU_mem   box_loss   obj_loss   cls_loss  Instances       Size
       2/49         0G    0.06403    0.03196    0.03949         20        640: 100%|██████████| 78/78 [27:28<00:00, 21.14s/it]
                 Class     Images  Instances          P          R      mAP50   mAP50-95: 100%|██████████| 10/10 [01:36<00:00,  9.67s/it]
                   all        310       1090      0.498      0.308      0.112     0.0485
      Epoch    GPU_mem   box_loss   obj_loss   cls_loss  Instances       Size
       3/49         0G    0.05395    0.03021    0.03868         33        640: 100%|██████████| 78/78 [27:31<00:00, 21.17s/it]
                 Class     Images  Instances          P          R      mAP50   mAP50-95: 100%|██████████| 10/10 [01:32<00:00,  9.30s/it]
                   all        310       1090      0.569      0.387       0.27       0.16
      Epoch    GPU_mem   box_loss   obj_loss   cls_loss  Instances       Size
       4/49         0G    0.04628     0.0271    0.03781         31        640: 100%|██████████| 78/78 [27:26<00:00, 21.11s/it]
                 Class     Images  Instances          P          R      mAP50   mAP50-95: 100%|██████████| 10/10 [01:32<00:00,  9.26s/it]
```






```
!python train.py --batch {BATCH_SIZE} \
                 --epochs {EPOCHS} \
                 --data data.yaml \
                 --weights yolov5s.pt \
                 --project nfl-extra
```

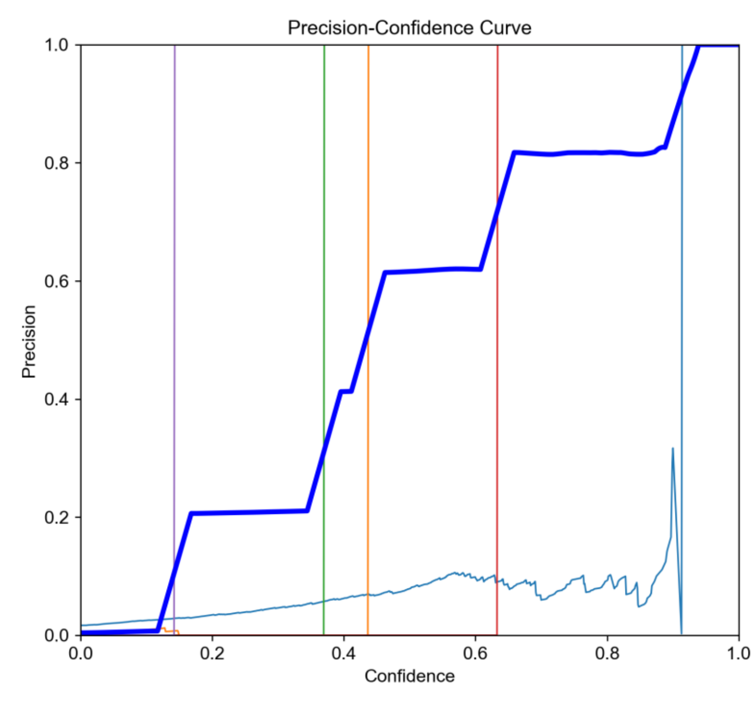
```
train: weights=yolov5s.pt, cfg=, data=data.yaml, hyp=data\hyps\hyp.scratch-low.yaml, epochs=2, batch_size=16, imgsz=640, rect=False, resume=False,
nosave=False, noval=False, noautoanchor=False, noplots=False, evolve=None, bucket=, cache=None, image_weights=False, device=, multi_scale=False,
single_cls=False, optimizer=SGD, sync_bn=False, workers=8, project=nfl-extra, name=exp, exist_ok=False, quad=False, cos_lr=False, label_smoothing=0
.0, patience=100, freeze=[0], save_period=-1, seed=0, local_rank=-1, entity=None, upload_dataset=False, bbox_interval=-1, artifact_alias=latest
github: up to date with https://github.com/ultralytics/yolov5
YOLOv5  v7.0-147-gaa7c45c Python-3.11.2 torch-2.0.0+cpu CPU

hyperparameters: lr0=0.01, lrf=0.01, momentum=0.937, weight_decay=0.0005, warmup_epochs=3.0, warmup_momentum=0.8, warmup_bias_lr=0.1, box=0.05,
cls=0.5, cls_pw=1.0, obj=1.0, obj_pw=1.0, iou_t=0.2, anchor_t=4.0, fl_gamma=0.0, hsv_h=0.015, hsv_s=0.7, hsv_v=0.4, degrees=0.0, translate=0.1,
scale=0.5, shear=0.0, perspective=0.0, flipud=0.0, fliplr=0.5, mosaic=1.0, mixup=0.0, copy_paste=0.0
ClearML: run 'pip install clearml' to automatically track, visualize and remotely train YOLOv5  in ClearML
Comet: run 'pip install comet_ml' to automatically track and visualize YOLOv5  runs in Comet
TensorBoard: Start with 'tensorboard --logdir nfl-extra', view at http://localhost:6006/
```

How did we train our model?

4. **Validation:** Evaluate the performance of the trained model on the validation set by running the command

"python detect.py --weights {path_to_trained_weights}
--img {image_size}
--conf {confidence_threshold}
--source {path_to_validation_images}“

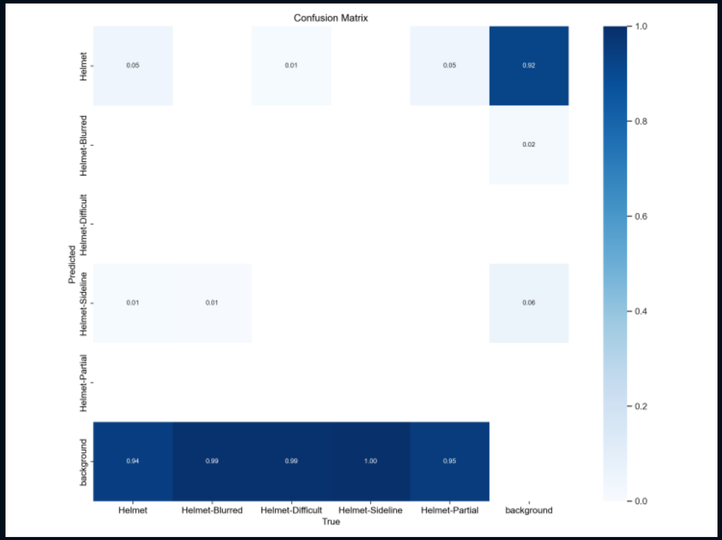


5. **Fine-tuning:** Fine-tune the model if necessary by adjusting the model architecture, hyperparameters, or adding more data to the training set.
6. **Inference:** Test the trained model on new images or videos to detect helmet.

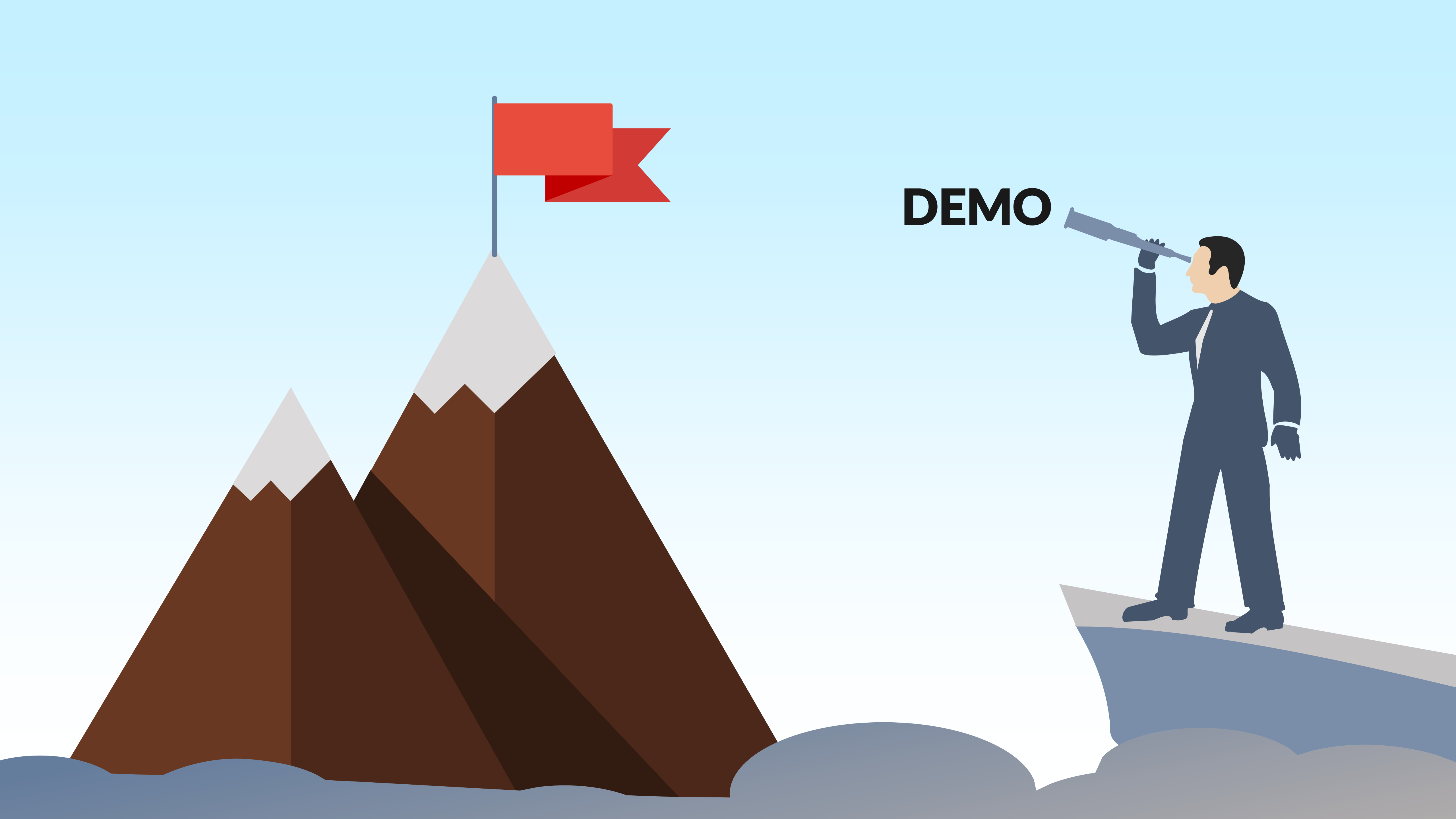
Results

- The table below shows the performance metrics of the model during training and validation:

Epoch	Train Loss	Val Box Loss	Val Obj Loss	Val Class Loss	Precision	Recall	mAP@0.5	mAP@0.5: 0.95
0	0.084	0.068	0.042	0.041	0.236	0.117	0.039	0.067
1	0.068	0.067	0.031	0.04	0.335	0.175	0.063	0.067
2	0.064	0.074	0.031	0.039	0.308	0.112	0.049	0.074
3	0.054	0.042	0.029	0.038	0.387	0.27	0.16	0.042
4	0.046	0.043	0.026	0.037	0.446	0.258	0.122	0.043
5	0.043	0.033	0.024	0.037	0.512	0.332	0.224	0.033
6	0.04	0.037	0.023	0.036	0.541	0.354	0.189	0.037



- The provided results show that the model is improving in terms of its accuracy and ability to identify objects and their corresponding classes. The decreasing object and class loss, along with increasing precision, recall, and mAP metrics, indicate that the **model is progressing well**. Additionally, the constant learning rate suggests that the model is not overfitting.


DEMO

Conclusion

- ✓ The integration of helmet detection models with other intelligent systems such as automated emergency response systems and autonomous vehicles can enhance the overall safety of individuals.
- ✓ The development of more efficient algorithms for real-time helmet detection can further improve the accuracy of the models.
- ✓ The application of helmet detection models can be extended to other safety gear such as safety shoes, gloves, and goggles.
- ✓ The use of computer vision and machine learning can be extended to other areas of safety, such as detecting unsafe behaviour in industrial environments.

"Even Yamraj can't keep up with the speed of karma - wear a helmet and stay safe on the road!"