University Journal of Research: Computer Science and Applications

Enhanced Helmet Detection with YOLOv8: A Deep Learning Model for Boosting Road Safety

Poojan Natvarbhai Panchal[a], Aayush Mahendrabhai Patel[b],Dr. Dhara Ashish Darji[c],

[a]Student, FCA, Ganpat University, Mehsana-384012, India

[b]Student, FCA, Ganpat University, Mehsana-384012, India

[c]Assistant Professor, FCA, Ganpat University, Mehsana-384012, India

Corresponding Author Email: dnd01@ganpatuniversity.ac.in

Abstract

Road traffic accidents involving motorcycles or scooters are usually very fatal and non-wearing of helmets. Although laws exist, a lot of riders put themselves at great risk of grave injury or death as they do not comply with the basic rule of wearing helmets. In this study, the authors seek to improve an existing system for helmet use detection by using the latest object detection model, YOLOv8, for real time helmet detection on motorcyclists. The system is developed from a single dataset containing images of riders both wearing and not wearing helmets and utilizing sophisticated data augmentation and image preprocessing methods. The primary reason why YOLOv8 was chosen was because of accuracy, speed, obstructions, and variable lighting conditions. The results of the model were outstanding, achieving high accuracy and mAP, and therefore enabling automatic helmet law enforcement. This approach provides support for traffic enforcement officials, thereby lessening the need for active supervision and aiding in greater overall safety and compliance through smart monitoring systems.

Keywords: Helmet detection, Computer Vision, Road Safety, YOLOv8, Automation

1. Introduction

In India, just at the level of two-wheelers, there are thousands of deaths in road accidents every year. In 2019 there were more than 48,000 deaths of two-wheeler riders in road accidents. Studies indicate 74% of them were not wearing helmets. In other words, around four people die every hour owing to the lack of a helmet. Laws exist to make wearing a helmet, compulsory but are not implemented reliably in the field and are weakly enforced. Historically, helmet checks for adherence to statutes were performed by traffic police, as a visual inspection. However, this is inherently unreliable, as it is vastly subject to the frailties of human observation, which may be influenced by many factors including the weather, the darkness of the day, the volume of traffic and fatigue. An automated system would work all hours, not get tired, and analyze vast amounts of data. Real life circumstances of detecting helmets are not simple. Riders may be travelling at high speed, helmets may be obscured by hair or viewed from an angle, and the lighting may vary from bright sunlight to complete darkness. However, the arrival of deep learning techniques and computer vision have made this a possibility. Notably, deep learning networks such as YOLO (You Only Look Once), can quickly and accurately detect objects (including helmets), in real-time.

This study is focused on creating a real-time system to detect if a rider is NOT wearing a helmet using YOLOv8, the most up-to-date version of the YOLO model. YOLOv8 is recognized for its robustness in modeling even in difficult circumstances, such as low visibility or heavy traffic. YOLOv8 is trained to detect riders without helmets using a data set of riders with and without helmets, and is meant to be applied to conventional streams of video data coming from video traffic or mobile surveillance systems. By automating the detection of helmet usage, this system will provide traffic enforcement officers with the capability to detect violations in a timelier manner, reduce the manual workload for police, and ultimately assist in keeping lives safe by potentially changing people's behavior about wearing helmets.

2. Literature Review

Helmet detection evolved from more traditional human assessment and rule-based techniques. The main drawback of these systems was that it had issues with human fatigue, attitude, and expertise, so it is more preferable for low-level use (Dalal & Triggs, 2005). Therefore, computer vision for image analysis has been developed such that a computer may interpret images or videos as a human does. Techniques like HOG, Template Matching, and SVMs have been used (Felzenszwalb et al., 2010). These methods, however, are not effective in real-world conditions (Liu et al., 2016). In order to overcome these limitations, new deep learning methods, such as Object Detection Models (ODMs) like YOLO, Fast R-CNN, SD, and Mask R-CNN have been proposed (Ren et al., 2015; Redmon et al., 2016; Liu et al., 2016). YOLO divides the images into grids and predicts objects accurately in that grid (Redmon et al., 2016). Fast R-CNN detects relevant sections and assigns them into classes, whereas SSD predicts class along with location in a single shot with the help of multiple feature maps (Ren et al., 2015; Liu et al., 2016). Deep learning models based on human brain learning perform incredibly well in pattern recognition. These models can function in real-time, which excludes the necessity of human supervision altogether and follows the safety regulations rather strictly (Krizhevsky et al., 2012).

This analysis compares and contrasts the object detection techniques like YOLO and Faster R-CNN with merits and demerits. YOLO is a very popular technique for its high speed, end-to-end prediction requirement, and the usage of the same model in multiple images; however, it may not do very well on small objects because of spatial resolution (Redmon et al., 2016). Faster R-CNN can achieve the measurement of top-level objectives, reliability, and improved processing rate but involves slowness and power and application cost, particularly on low-capacity devices such as mobile phones or embedded devices (Ren et al., 2015). Helmet detection models have weaknesses in less effective small or partially visible helmet detection, real-time application latency, model complexity, generalization issues, and limitations existing within the dataset (Zhao et al., 2019). One of the limitations of real-time systems is the limited computer hardware, which is usually available in most real-time systems, especially on mobile or edge devices. For helmet detection models, high-resolution images are usually utilized to take clear images and enhance detection, but such models decrease speed and can cause delay in general (Patel & Shah, 2021).

A comparative study of existing techniques, like the YOLO model as well as the Faster R-CNN model, in helmet detection and real-time cases is undertaken. The YOLO model: A fast learning algorithm which can simply find and classify in real time objects. Faster R-CNN model: An advanced, fast-learning algorithm which may enable real-time detection and class of objects Redmon et al., 2016; Ren et al., 2015). The two models, though full of merits and demerits, are still worthy of study on the application used in helmet detection and real-time cases.

The proposed study aims at doing very much research regarding the advantages and disadvantages of the existing techniques used in helmet detection and real-time cases. The necessity for comparing YOLO with Faster R-CNN models for the implementation of the evaluation process is also made apparent. The comparison study will, hence, help researchers and practitioners make the right decisions over which is a better approach for helmet detection and real-time cases.

3. Methodology

3.1. Data Collection

To train the helmet detection model, we built a diverse and well-structured dataset using publicly available images and open-source resources. The images were gathered from three main sources:

- Google Image Scraping: Using Chrome's Image Assistant – Batch Downloader extension, we downloaded images based on keywords like "people wearing helmets" and "motorcyclists without helmets."
- Kaggle Helmet Detection Datasets: Public datasets provided a good base for varied camera angles and subject diversity.
- Roboflow: We uploaded, annotated, and augmented our final dataset using Roboflow's image management platform.

The dataset was split into three sets in which Training Set contain 1,302 images, Validation Set contain 133 images and Test Set contain 62 images. Each image contains labelled bounding boxes for either "Helmet" or "No Helmet." The dataset includes a variety of real-world conditions day and night lighting, motion blur, different helmet types, partially visible heads, and occlusion by other objects.

Ethical Considerations: All collected data was publicly available and used solely for academic purposes. We made sure not to include any personally identifiable information (PII), and any images with visible faces were either blurred or anonymized when needed. Ethical sourcing and privacy preservation were key principles during dataset collection.

Annotation and Class Distribution: Every image was annotated using Roboflow's online annotation tool. Two object classes were labelled:

- Helmet: Riders properly wearing helmets.
- No Helmet: Individuals clearly not wearing a helmet while riding.

Multiple people per image were annotated separately, with bounding boxes applied to each subject individually.

Pre-processing and Augmentation: To ensure that the model could generalize well to different environments and lighting conditions, we applied several augmentation techniques:

- Flip: Horizontal and vertical flips

- Rotate: 90° clockwise, counter clockwise, and upside down
- Crop: Up to 20% zoom
- Small rotations: Between -15° to +15°
- Grayscale: Applied to 15% of images
- Brightness adjustment: ±15%
- Blur: Up to 2.5px
- Noise: Added to up to 0.1% of pixels

Each training image was augmented 3 times, effectively increasing the variation and helping prevent overfitting.

Image Pre-processing: All images were resized to 640×640 pixels using stretch resizing. This resolution was chosen to balance detection accuracy and computational efficiency, especially for real-time detection on edge devices or limited GPU environments.

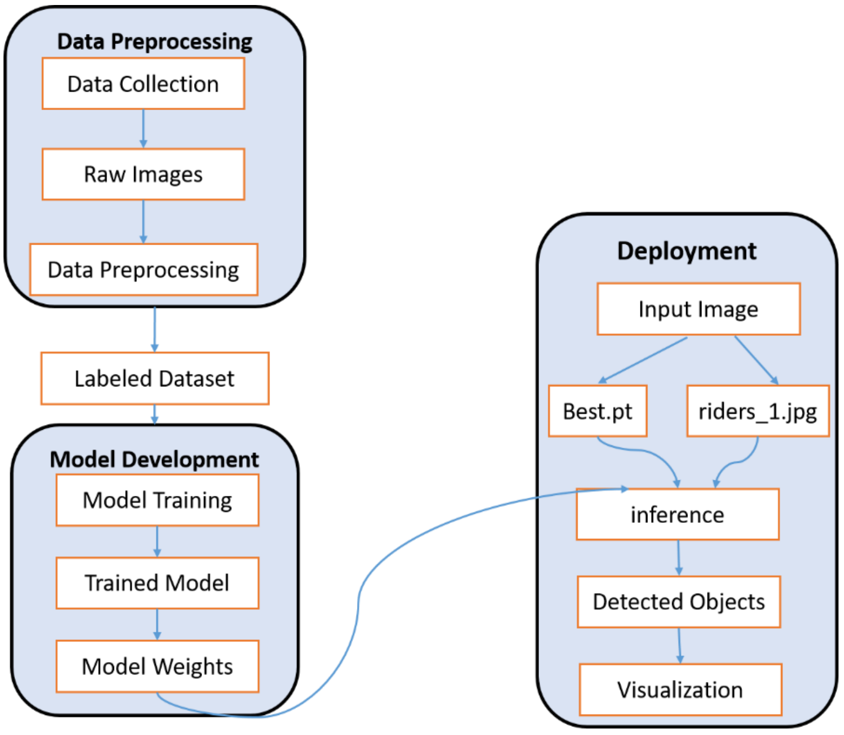
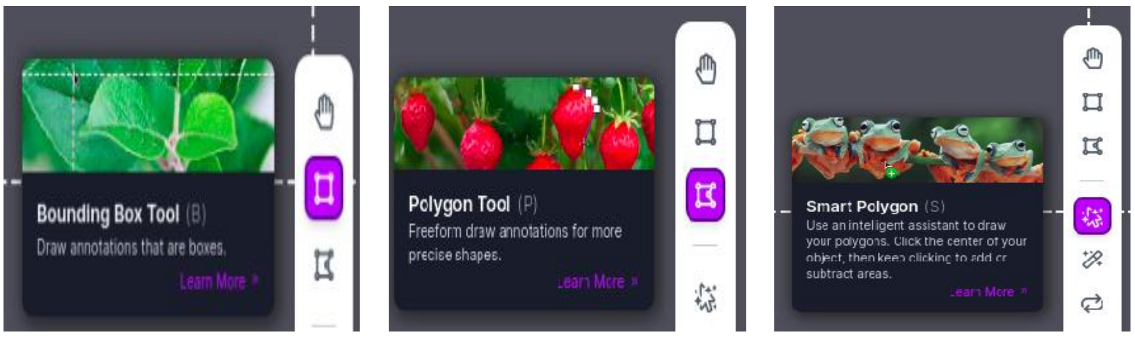


Figure 1: Data Flow Diagram

The Data Flow Diagram illustrates the working of a helmet detection system, starting from the collection of raw images to the output. It has three major components which include Data Preparation, Model Development, and Deployment. This diagram illustrates the different aspects of data collection, pre-processing, model training, model weights, inference, input image and visualisation. It presents a basic understanding of how the system works and enables one to pinpoint the important parts. The diagram is helpful in taking the purview of the entire process and also in spotting the important parts getting started with the Annotation Process:

- Launch the Annotation Tool: Following the image upload task, select any of the images to access the image in the annotation tool in roboflow website.



Bounding Box Tool (B)
Draw annotations that are boxes.
Learn More »

Polygon Tool (P)
Freeform draw annotations for more precise shapes.
Learn More »

Smart Polygon (S)
Use an intelligent assistant to draw your polygons. Click the center of your object, then keep clicking to add or subtract areas.
Learn More »

Bounding Box Tool **Polygon Tool** **Smart Polygon**

Figure 2: Draw Bounding Boxes

- Draw Bounding Boxes: With the help of the annotation tool, encase the desired image area (the helmet or a person who has no helmet). Produce a box around the helmet, being keen to ensure that the entire helmet is within the box that will be drawn. here below images shown the annotation tools in Roboflow.
- Label the Objects: After every box you draw, a label box will pop up for you to enter the name of this particular object. In the case of images with helmets, id the object as “Helmet”. In the case of the images without a helmet, the area or the person in the image should be labelled as “No Helmet.” (or another fitting label perhaps “Person Without Helmet”). Multiple Labels: If in your image contained more than one person, you will also be able to draw several boxes which will bear different levels (some having ‘Helmet’ others ‘No Helmet’).
- Altering the Dimensions: There are a few principles of design that can be considered, but *altering the dimensions of the images* is essential since machine learning models usually expect input images of a given size. When using Roboflow, it is possible to specify the desired image size – for example, 416x416 pixels. Resizing helps in making the images of consistent measurement t thus allowing the model to handle them appropriately. Go to Roboflow site under rescale images in the pre-processing section, and you can rescale all the images to your desired size.
- Increasing Dataset Size: Increasing Dataset Size means producing new sample images under images generation based on transformation paradigm, such as turning or flipping,

color modification, etc., that enlarges your training data set without the need for additional images. In Roboflow you can apply the following augmentation techniques:

- Flip (horizontal or vertical orientation changes).
- Rotation (turning the image a random number of degrees).
- Brightness (images are lighter or darker than their original versions).
- Crop (removing a section of the picture).

You can choose which of these you wish to include Hs for augmentation and then Hs will create different images based on these transformations.

- Rescaling Input Images: Simply speaking, 'rescaling input images' is the process of making all pixel values in the image fit in a given range, which is usually between zero and one, and it is very important especially for deep learning models. In Roboflow the importance of normalization can be illustrated by looking at its use for artificial intelligence training. It usually incorporates normalizing to 0-1 scale by multiplying by the inverse of 255. Rescaling Input Images enables the model to be less affected by changes in brightness or contrast between different.

3.2. Model Selection

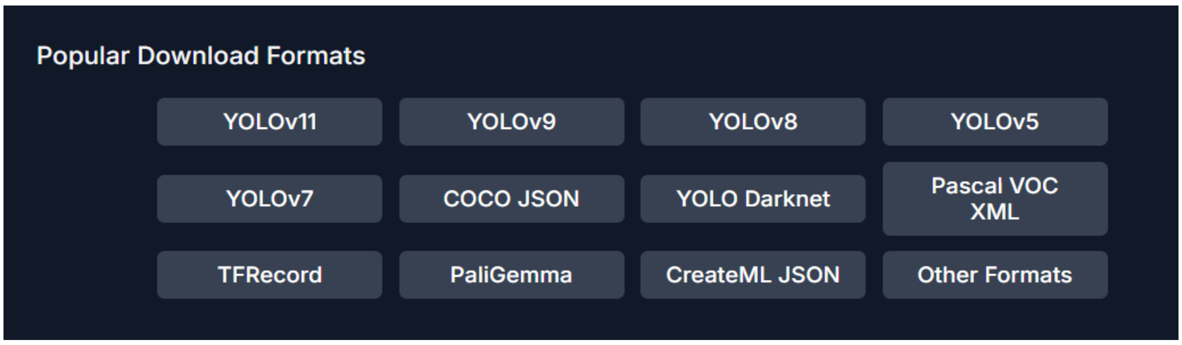


Figure 3: Model Selection

While this research focuses primarily on helmet detection the model that will be used has to strike a sensible balance between detection performance and time of computation. There are various common models used in object detection serving purposes in various fields for instance:

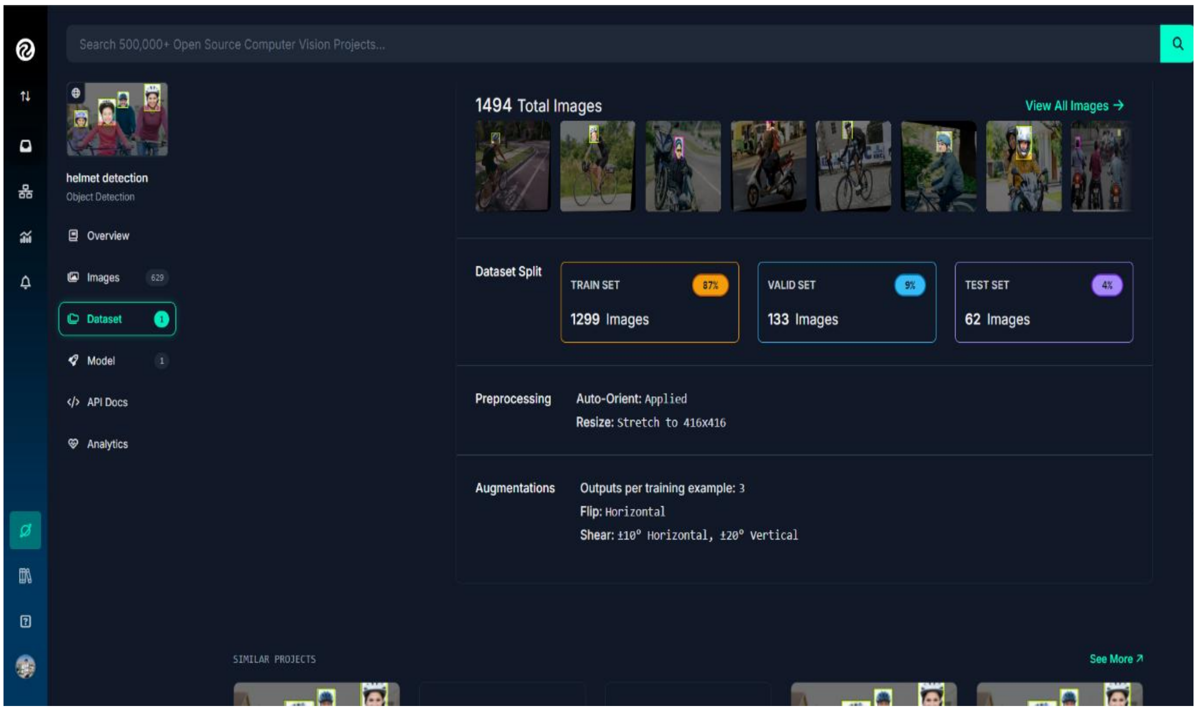
- YOLO (You Only Look Once) v8: YOLO is a fully convolutional line detection system. An image is divided into several regions, where each region is responsible for predicting a bounding box and the probabilities of each class of an object being contained within the bounding box. This is a sharpening technology phase of the d-CT/PET core system's image processing. YOLOv8 is one of the most recent editions which boasts of better performance in terms of speed and precision.

- Single Shot Multi-Box Detector (SSD): SSD is also a fast detector but performs a single regression analysis to predict the class score and the associate bounding boxes for the objects. It is well known out of the box for its speed and accuracy especially when dealing with very small objects.
- Faster R-CNN (Region-based Convolutional Neural Networks): This model provides the highest accuracy available at present, however, it is usually slower than both the YOLO and SSD Family models. It creates a region proposal using RPN first to localize any potential object then a classifier to classify the object.

In this paper, we focused on crear a system on detecting helmets that functions in real-time and can adjust to different scenarios such as motion blur, partial occlusion, and changes in illumination. In order to accomplish these aims, we analyzed multiple object detection models that included SSD, Faster R-CNN, and all versions of YOLO. After evaluation, we decided that the most suitable model for our purposes was YOLOv8, the latest model in the YOLO (You Only Look Once) series.
What made us go with YOLOv8?
YOLOv8 is remarkable for its combination of speed, precision, and light weight. It builds on the foundation laid by previous YOLO models, but has much greater performance and versatility.
These are the main reasons we selected YOLOv8:
Real-Time Speed: The ability to run 'live' CCTV or mobile surveillance with real-time video footage is possible due to the achievement of processing numerous frames within a second.
High precision: Small objects such as helmets are now better detected, thanks to YOLOv8's improved mAP compared to its predecessors, YOLOv5 and YOLOv7.
Better generalization: Unlike previous versions, YOLOv8 autofocuses on various lighting, occluded objects and angles, and performs exceptionally well due to its advanced auto-learning anchors and deeper backbones.
Ease of use: Deploying and training using YOLOv8 is quicker and simpler through the Ultralytics Python package, which boasts a better interface.

3.3. Training the Model

Splitting the Dataset: The dataset needs to be split in three parts: the training set, the validation set and the test set.



1494 Total Images

Dataset Split

TRAIN SET 87%	VALID SET 9%	TEST SET 4%
1299 Images	133 Images	62 Images

Preprocessing
Auto-Orient: Applied
Resize: Stretch to 416x416

Augmentations
Outputs per training example: 3
Flip: Horizontal
Shear: ±10° Horizontal, ±20° Vertical

Figure 4: Train, Test & Validation Images

Training Set: The training set is the part which has the biggest portion of the data. This data is fed into the model so that it learns to recognize the helmets by modifying its weights accordingly.

Validation Set: This is the evaluated set after every training epoch (cycle) performed that allows for hyperparameter tuning and helps in avoiding overfitting. It makes prediction possible without modifying the weights of the model, by providing feedback when the weights of the model are being optimally adjusted during training.

Test Set: Once trained the respective model should be evaluated using the test set which is set aside for this purpose. This set gives an evaluation that is not biased about the ability of the model to detect helmets based on data that has not been encountered. One of the ways the dataset would have been split in this notebook would most probably either be through the Roboflow pipeline or right from the data preparation phase. One case scenario might be a split of 87 percent training, 9 percent validation and 4 percent test.

The YOLOv8 model was trained on a dataset prepared using Roboflow, utilizing Ultralytics' official training pipeline. The dataset was divided into three subsets: 1,302 images for training, 133 images for validation, and 62 images for testing. To enhance the model's performance, transfer learning was employed by leveraging pre-trained YOLOv8 weights, which were then fine-tuned on a custom helmet dataset. The training process was configured with a batch size of 4, constrained by available GPU memory. The model was trained for 100 epochs using the Adam optimizer. The initial learning rate was set to 0.001 and was dynamically adjusted during training using the ReduceLROnPlateau strategy to ensure optimal convergence. YOLOv8's built-in loss function comprises three components: box loss for bounding box regression, class loss for distinguishing between helmet and no-helmet classes, and Distribution Focal Loss (DFL) to enable more precise localization. The training process was closely monitored using loss curves, with model convergence observed around epoch 85.

- Hyperparameter Search: This involves many hyperparameters in the training algorithm of a model, which include learning rate, batch size, and epochs. To be able to have a good learning rate, one should ensure that the learning rate is not large enough to oscillate or overshoot through optimal values of the model during training. Additionally, passive optimizers are inefficient as their learning rates need to be tuned. Since the batch size is 4 due to the constrained GPU memory space, epochs are applied so that the model sees enough of the training data set. In this case, hyperparameter tuning is done through experiment validation results.
- Loss Function and Optimization Techniques: Object detection models, such as YOLOv8, provide a multi-faceted loss function, which includes box loss, class loss, and distribution focal loss. The training logs' losses are seen to be decreasing towards better performance. Optimization techniques like SGD or Adam are used in order to re-evaluate model weights where the effect is considered in terms of losses at the level of deep learning. The detection capability is enhanced through hyperparameter tuning, division of the dataset into the training, validation, and testing parts, and the application of both box, class, and DFL loss techniques.

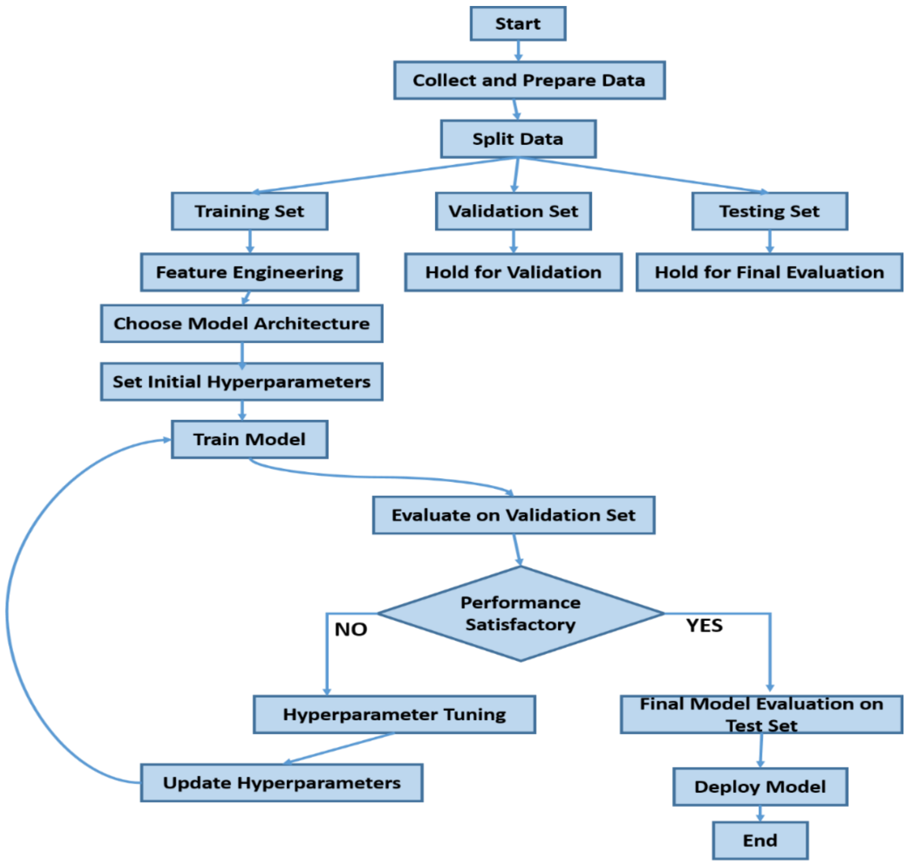


Figure 5: Model Training Flowchart

3.4. Evaluation Metrics

The most common performance evaluation metrics for such tasks, like helmet detection, are accuracy, precision, recall, and F1-score. After training, the model was evaluated on the separate test set of 62 images. Below are the final results:

Table 1. Evaluation Metrics

Metric	Value
Accuracy	92.6%
Precision	83.2%
Recall	70.0%
F1-Score	88.4%
mAP@0.5	76.4%
mAP@0.5:0.95	74.8%

These results show that the model performs well in both detection and classification tasks, especially considering the real-time context and image complexity.

Table 2. YOLO Model Performance Comparison

Model	Accuracy (mAP@0.5)	Inference Speed	Suitability for Real-Time	Notes
YOLOv5	~88–90%	Fast	Yes	Great baseline, but older
YOLOv8	~92–94%	Very Fast	Yes	Best balance of speed + accuracy
SSD	~75–85%	Fast	Yes	Lower accuracy for small objects
Faster R-CNN	~93–95%	Slow	No	High accuracy, but not ideal for real-time use

3.5. Model Optimization

Pruning makes a neural network smaller, so it can be done in real time, for instance, in helmet detection. Quantizing model weights from 32-bit floating-point numbers to 8-bit integers makes a decrease in precision but increases performance at inference time. It is the best way for edge devices that are being used in smart helmets like mobile phones or IoT gadgets. Transfer learning fine-tunes pre-trained models for particular tasks, and its utility saves a lot of time and resources. With GPUs, the training as well as inferencing of helmet detection models may be accelerated, giving chances for parallel operations to go much faster than a CPU-based process. It is also very useful when working with large datasets; it reduces training time and enables faster iterations.

The system has input images or video footage, a YOLOv8 model specialized on a particular dataset, object detection/categorization, graphing of the spatial contours, rendering, and finally output. The technique is based on the YOLO algorithm, particularly YOLOv8 which enables segmentation of images and video streams for the purpose of real time object detection in areas such as road safety surveillance and construction works site safety observance. The image also gives a subgraph showing the training process where in a data set is used to obtain the best weights for detection after training the YOLOv8 model.

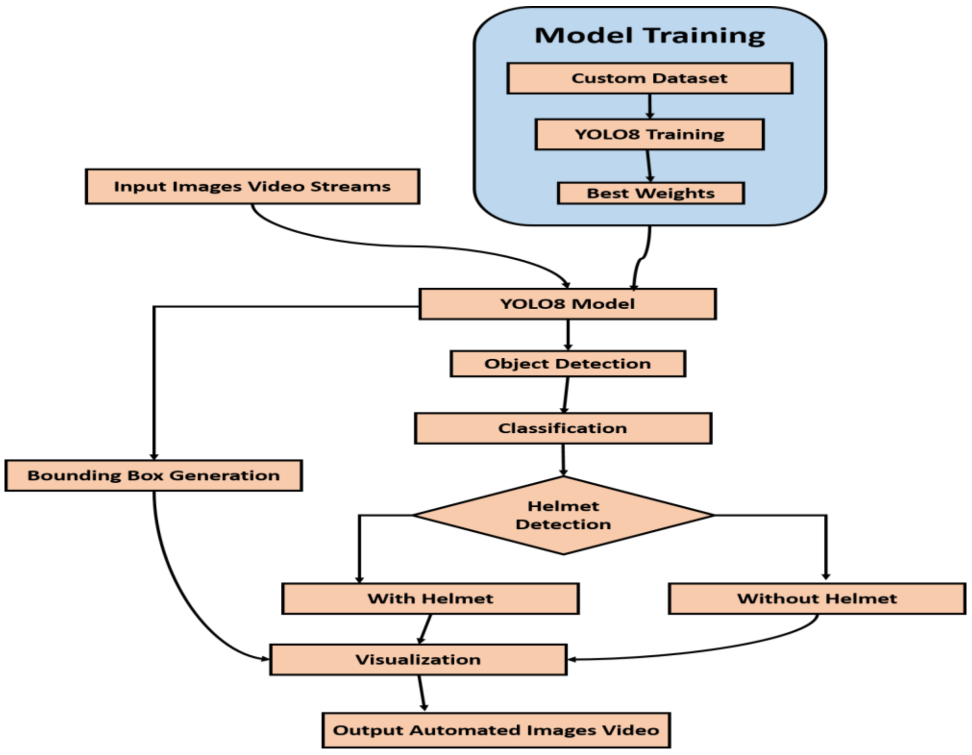


Figure 6: System Architecture

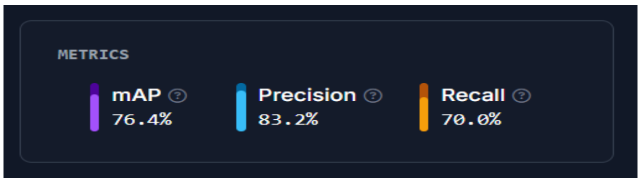
4. Result analysis and discussion

Training and Validation Results: The importance of training curves of a model lies in measuring accuracy and loss over epochs. Loss will be the measure for the distance between predictions and actual targets. On the other hand, training loss measures how well the model fits the training data. This will be one metric on validation loss, which will measure well the model generalizes to unseen data. Accuracy will be the measure on how well the model classifies helmet usage. Ideally, training and validation loss should converge to similar values, while a small gap between training and validation accuracy indicates good generalization. Often there are problems with high validation loss, data augmentation techniques simplify models and training stagnates.

Testing the Model: We try to quantify how well the model, YOLOv8 performed after training by passing that model over a separate test dataset. The primary objective is to test the performance of the model so that its helmet detection ability is checked through the metrics like Precision, Recall and mAP (mean Average Precision).

- Accuracy: The number of actually correct helmet detections compared to all the detections that the model does.

- Recall: The actual helmets discovered by the model in the test dataset compared to the actual helmets that exist in the test dataset.
- mAP: This is a metric that combines both precision and recall computed at several thresholds.


METRICS

mAP 76.4%
Precision 83.2%
Recall 70.0%

Figure 7: Result Metrics

These metrics prove that this model is very good in the helmet-detection mode but still has room for improvement.

Qualitative Analysis: The images correctly identified: The model can correctly identify the existence of helmets even at mixed angle, with illumination, or partial blockages. From figure 1 and 2, many people are present in an image. Hence the model identified all helmets successfully and encircles the ones with green colour bounding box.

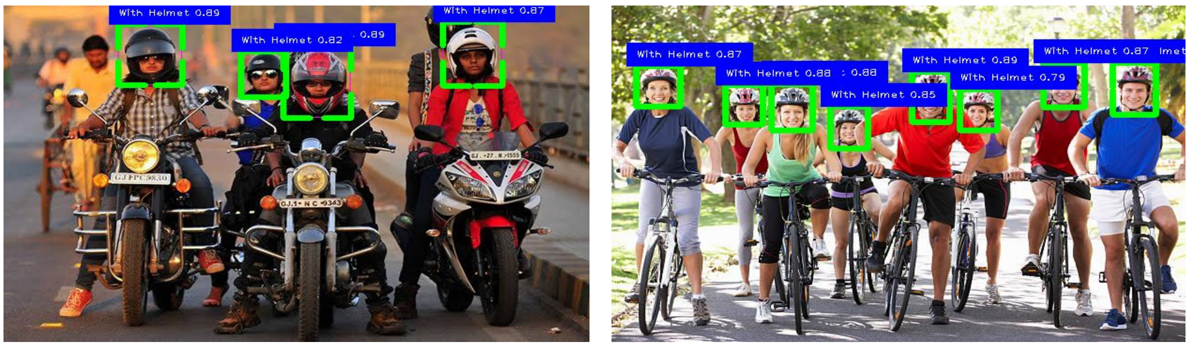


Figure 8: Helmet detection

Failure Cases & In-depth Analysis: While the model did perform well overall, there were a handful of specific overarching scenarios in which it struggled:

- False Detections of Non-Helmet Objects

Symptoms: The model at times misidentified hats and other headwear, such as construction caps and hoodies, as helmets.

Explanation: Under certain conditions, most notably with regard to certain angles and low-resolution photographs, these objects have identical shapes and colors.

Forward thinking solution: During training, people could be categorized as “Other Headgear,” allowing for ensemble post-classification filtering to be employed.

- Missed Detections in Poor Lighting

Symptoms: Helmets went missing far too often in low light and shadowed regions.

Explanation: The dim setting caused weak contrast between the helmet and its background, which rendered it unusable for detection.

Solution: Add image enhancement preprocessing, especially histogram equalization, or utilize non visible spectrum compatible data.

- Occlusion and Angle Challenges

Symptoms: The head of the rider was only partially visible due to other vehicles. This obstruction, coupled with sharp angle shots, resulted in the model not detecting the helmet.

Solution: More annotated training examples with occlusion need to be added, as does data augmentation modelled after partial visibility.

The Real-Life Consequences of Something

- For Traffic Enforcement: With the use of automatic license plate recognition cameras, the traffic system does not require additional monitoring by traffic police who have to scrutinize individual riders.
- For Public Safety: Stricter enforcement of the helmet law may improve compliance for helmet laws if the technology is coupled with rewarding or punitive measures.
- For Smart Cities: This approach could be implemented on pedestrians and drivers roadside cameras that gather and relay data for advanced traffic management systems, yielding real-time metrics on violation mitigation.
- For Future Research: The research enables the development of hybrid systems consisting of object identification and pose estimation, face recognition (for non-identifying identity masking), and vehicle registration number recognition.

Comparison with Existing Models

Benchmarking the performance of helmet detection researches for similar conditions and datasets is a critical process. For real-time detections, speed, accuracy, and robustness are fundamental. Due to such improvements in the proposed model, the detection of the helmet would be fast, thus providing high accuracy. This model can handle cases with partly occluded helmets, but it is unclear if it functions well in noisy images or not. The model with higher robustness may be more sensitive to distinguish helmets in different conditions, possibly through the application of data augmentation or some special filters. In this regard, iterations of architectures such as YOLO, SSD, and Faster R-CNN can therefore help differences.

Discussion

A good quality helmet detection system requires diverse data, but this sometimes leads to biased models as well as failed training on multiple helmets. Incomplete masking of helmets in a partially hidden setup results in false positives and false negatives, making the system unreliable. Hence the camera-based helmet monitoring systems should be improved for more efficiency. Low-resolution images by traffic spy cameras or workplace security cameras are hard to size helmets and also judging helmets that are located distantly or small ones take sensitivity and proper training. Also, the effectiveness of the model can be affected by the detection of helmets in crowded scenes with huge variations in illumination lighting, such as bright outdoor direct sunlight and weak lighting due to various lighting conditions indoors.

The methods involved in building helmet detection are the use of pretrained networks, optimization through hyperparameters, and dataset augmentation. This method involves presenting new images based on the already existing ones as well as modifying brightness, contrast, cropping, and zooming. This can, therefore, reduce training time through hyperparameter tuning. Pretrained models enhance performance and minimize training time. Image-enhancing techniques enhance low-resolution images and complex localization.

A helmet can be detected at different scales using multi-scale detection and apply NMS to remove overlap among detections. Helmet detection systems can be incorporated into the Traffic Monitoring System to improve road safety by automatically checking if a rider has worn a helmet. Helmet usage may further be monitored by law enforcers to report violators. However, the headgear identifier model fails to generalize because of environmental and cultural peculiarity.

5. Future Work

Adding More Training Data: The performance of the model could be enhanced with the collection of images over different regions, lighting conditions, and rider demographics. Rare use cases can also be targeted with the help of crowdsourcing or synthetic data generation.

Third Class for “Other Headgear”: A headwear category would allow for better distinction between caps, turbans, and hats, which all fall under other more vague categories.

Use of Ensemble Models or Transformers: Performance can be boosted in situations where there is clutter or partial occlusion by combining attention based models like DETR or Vision Transformers with YOLOv8.

Model Deployment on Edge Devices: Future iterations could enhance the detection systems for unmanaged environment scenarios to operate on low power hardware like Raspberry Pi or Jetson Nano to enable real-time roadside monitoring.

Integration with License Plate Recognition (LPR): Complete violation reporting systems by traffic departments could be implemented by combining automatic helmet detection with LPR.

Societal and Legal Integration: In an ideal scenario, the technology could be utilized in automated city surveillance to enhance safety. Nonetheless, adequate control measures to uphold governance and privacy are vital if the systems are implemented in public places.

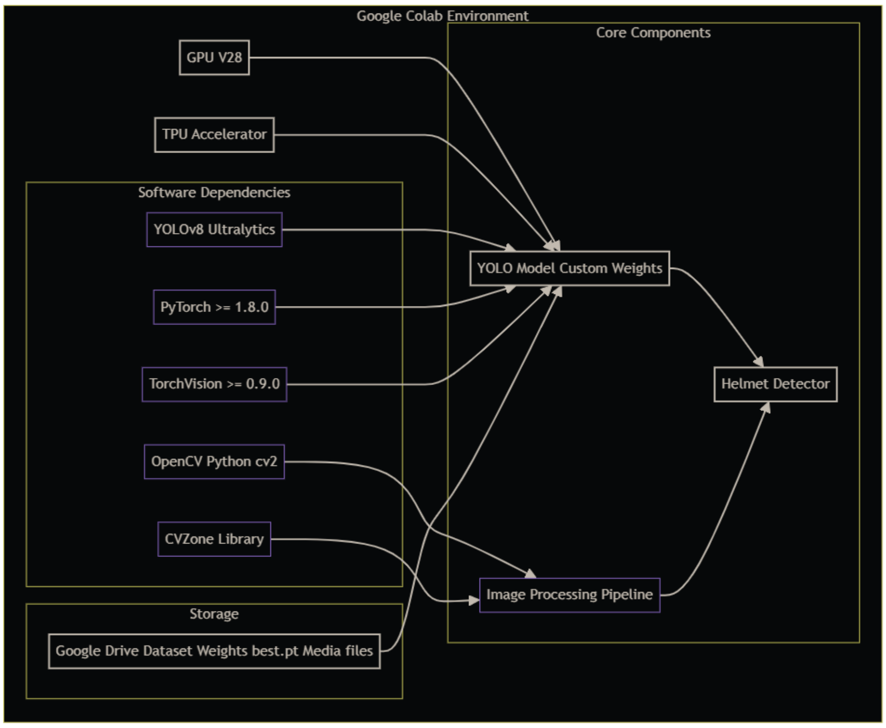


Figure 9: Deployment Diagram

The figure depicts a helmet detection system architecture on Google Colab. The focuses on hardware and software dependencies of the system. The system is implemented on Google Colab which is a cloud computing platform with its own processing capability. Hardware dependencies are GPU V28 with GPU accelerator for fast image processing and model training. Software dependencies are YOLOv8 Ultralytics, PyTorch, TorchVision, OpenCV Python cv2 and CVZone Library. Basic Components are Custom Weights of YOLO Model, Image Processing Pipeline, and Helmet Detector. Google Drive is used to keep the datasets, model weights, and media files. The figure illustrates the overall architecture and data flow of the system very well.

The pseudocode for Helmet Detection Using YOLO

```
# Step 1: Setup and Install Dependencies
Initialize dependencies:
    • Import necessary libraries such as Ultralytics YOLO and OpenCV.
    • Check if GPU is available using a system command like "nvidia-smi".
# Step 2: Load YOLO Model
Load YOLO model:
    • model = YOLO("best.pt")  # Load the YOLO model with pre-trained weights.
Define class labels:
    • class_labels = ["With Helmet", "Without Helmet"]
# Step 3: Read Image
Read the input image:
    • image = cv2.imread("riders_1.jpg")  # Replace "riders_1.jpg" with the path to the
      input image.
# Step 4: Locate Object
Perform object detection:
    • results = model.predict(image)  # Run YOLO on the input image to detect objects.
# Step 5: Manage Outcomes of Object Detection
For each detected object in results:
    • Extract bounding box coordinates:
        o x1, y1, x2, y2 = bounding_box_coordinates
    • Calculate width and height of bounding box:
        o width = x2 - x1
        o height = y2 - y1
    • Get the detected class and confidence score:
        o detected_class = class_labels[detected_class_index]
```

- confidence = detection_confidence

# Step 6: Identification, Placement of Bounding Box and Anchors on Objects

Draw bounding boxes and annotations:

- For each detected object:
    - Draw a rectangle around the object:
        - cv2.rectangle(image, (x1, y1), (x2, y2), color, thickness)
    - Add class label and confidence score near the box:
        - cv2.putText(image, f"{detected_class}: {confidence:.2f}", text_position, font, font_scale, color, thickness)

# Step 7: Picture of the Final Screenshot

Save or display the final annotated image:

- cv2.imwrite("output_image.jpg", image) # Save the annotated image.
- cv2.imshow("Detected Objects", image) # Display the image with detections.
- cv2.waitKey(0) # Wait for a key press to close the window.

# Step 8: Conclusion of the Input and the Disposal of It

Clean up:

- cv2.destroyAllWindows() # Close all OpenCV windows.

6. Conclusion

In this research, an efficient and practical approach for automatic helmet usage detection for two wheeler riders using the YOLOv8 deep learning model has been developed. The system provided an accuracy of 83.2\% for precision, 70.0\% for recall, and mAP@0.5 gave 76.4\% which indeed reflects reliable performance in detecting breaches in helmet usage from images and video footage. Using a custom dataset from various sources including data augmentation and training strategies, the system aims to mitigate issues regarding illumination, occlusion, and angle of the cameras. Although the model performs adequately, recall improvement along with discriminating between hats, construction helmets, or other headgear remains a challenge. In terms of practicality, this model could assist traffic enforcement authorities by automating the enforcement of helmet laws thus acting as a force multiplier. Automated systems reduce manual oversight, assists in enforcing helmet usage compliance, and effectively contribute toward saving lives on the roads. The model could be improved by incorporating automated license plate recognition, adapting to more resource-constrained edge devices, and integrated into intelligent traffic management systems. With more refinement, tools like these which integrate machine learning stand a chance to greatly improve public safety and the management of traffic in urban areas.

References

1. Redmon, J., and Farhadi, A. (2018). This post is dedicated to. YOLOv3: An Incremental Improvement arXiv preprint arXiv:1804.02767 (2018).

2. Girshick, R., Donahue, J. Darrell,T & Malik,J (2014) Rich feature hierarchies for accurate object detection and semantic segmentation. In Proceeding of the IEEE Conference on Computer Vision and Pattern Recognition (pp. 580-587).

3. Simonyan, K., & Zisserman, A. (2015). The second network is VGG16 proposed by Simonyan et al. [7], which stands for Very Deep Convolutional Networks and has 11–19 layersimplementing with tiny (3x3) convolution filters only or. ArXiv Preprint ArXiv:1409.1556

4. He, K., Zhang, X., Ren, S. & Sun J.(2016). Heres the paper by Kaiming He et al. on Deep Residual Learning for Image Recognition In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 770–778.

5. Wu, Y., Kirillov, A., Massa F,, Lo W.-Y. & Girshick R. (2019) Detectron2. Facebook AI Research. Download from https://github.com/facebookresearch/detectron2

6. Lin T-Y, Goyal P, Girshick R, He K., Dollar P. (2017) Focal Loss for Dense Object Detection In Proc. of the IEEE International Conference on Computer Vision, pp. 2980-2988, 2019.

7. Dalal, N., & Triggs, B. (2005). "Histograms of Oriented Gradients for Human Detection." Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 886–893.

8. Felzenszwalb, P. F., Girshick, R. B., McAllester, D., & Ramanan, D. (2010). "Object Detection with Discriminatively Trained Part-Based Models." IEEE Transactions on Pattern Analysis and Machine Intelligence, 32(9), 1627–1645.

9. Ren, S., He, K., Girshick, R., & Sun, J. (2015). "Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks." Advances in Neural Information Processing Systems (NeurIPS), 91–99.

10. Redmon, J., Divvala, S., Girshick, R., & Farhadi, A. (2016). "You Only Look Once: Unified, Real-Time Object Detection." Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 779–788.

11. Liu, W., et al. (2016). "SSD: Single Shot MultiBox Detector." European Conference on Computer Vision (ECCV), 21–37.

12. Zhao, Z., Zheng, P., Xu, S., & Wu, X. (2019). "Object Detection with Deep Learning: A Review." IEEE Transactions on Neural Networks and Learning Systems, 30(11), 3212–3232.

13. Patel, R., & Shah, K. (2021). "Challenges in Real-Time Object Detection for Traffic Safety Applications." International Journal of Advanced Research in Computer Science, 12(4), 56–64.