International journal of basic and applied research

www.pragatipublication.com

ISSN 2249-3352 (P) 2278-0505 (E)

Cosmos Impact Factor-5.86

# Safety Helmet Wearing Detection Model Based on Improved YOLO-M

P. SIVA PRASAD, Assistant Professor, Dept of MCA, Chirala Engineering College, Chirala, Lakshmiprasad8216@gmail.com

VINNAKOTA SAI MURALI, PG Student - MCA, Dept of MCA, Chirala Engineering College, Chirala, v.saimurali99@gmail.com

**Abstract:** Construction site safety remains a critical concern, necessitating innovative solutions to ensure the well-being of workers. This study introduces an intelligent safety helmet detection system leveraging computer vision technology to monitor and enforce safety protocols in real-time. Through comprehensive analysis, we compare the performance of various state-of-the-art object detection architectures, including YOLOv5s, YOLOv5 - YOLO M, SSD, RetinaNet, FasterRCNN, YOLOv3, YOLOv4, YOLOv5 - GhostCNN, and YOLOv8. Our evaluation focuses on efficiency, accuracy, and computational requirements, aiming to provide insights into their suitability for safety compliance applications in the construction industry. The primary beneficiaries are construction workers, whose safety is paramount, alongside site managers who can optimize resource allocation and streamline monitoring efforts. Initial results demonstrate YOLOv5 - GhostCNN's potential to achieve over 97% mean Average Precision (mAP), suggesting promising avenues for further enhancing workplace safety. This research contributes to a safer working environment, facilitating better adherence to safety regulations and reducing the risk of construction-related accidents.

*Index Terms: Attention mechanism, feature fusion, safety helmet, YOLOv5s model.*

## 1. INTRODUCTION

In recent years, the integration of intelligent devices and deep learning algorithms has revolutionized various industries, enhancing efficiency and safety measures. In sectors such as transportation and retail, technologies like license plate recognition and facial recognition systems have become commonplace, optimizing processes and ensuring security. However, the construction industry presents unique challenges due to its complex environment and inherent safety risks, particularly concerning falling objects. In this context, the utilization of safety helmets emerges as a critical measure to mitigate injuries and safeguard workers' lives [1].

The construction industry is notorious for its hazardous working conditions, with the risk of falling objects posing a significant threat to worker safety. Safety helmets play a vital role in minimizing the impact of such hazards and reducing the likelihood of severe injuries [1]. By providing a protective barrier, safety helmets serve as a crucial line of defense against head injuries, which can be debilitating or fatal in construction accidents. Therefore, ensuring the proper usage of safety helmets is paramount to safeguarding the well-being of construction workers [2].

Traditionally, monitoring compliance with safety helmet regulations relied on manual supervision, which was both inefficient and prone to errors. Designated personnel would be tasked with observing workers on-site to identify any instances of non-compliance. However, the expansive nature of construction sites and the dynamic nature of work made it challenging to effectively enforce safety protocols [2]. Moreover, this approach resulted in the inefficient allocation of manpower resources, diverting personnel from other critical tasks.

With the advent of deep learning technologies, there has been a paradigm shift in safety monitoring practices within the construction industry. Deep learning algorithms, particularly those based on computer vision, offer real-time monitoring capabilities that are well-suited to the dynamic nature of construction sites [3]. By leveraging advancements in image processing and pattern recognition, these algorithms can autonomously detect and analyze various safety-related parameters, including the proper usage of safety helmets.

Early attempts at safety helmet detection algorithms based on You Only Look Once (YOLO) architectures demonstrated promising results in terms of real-time performance. However, these algorithms often suffered from low accuracy, limiting their effectiveness in practical applications [3]. Subsequent research endeavors focused on enhancing the accuracy of detection algorithms while maintaining real-time capabilities.

Researchers have explored various strategies to improve the performance of safety helmet detection algorithms based on YOLO architectures. Modifications to the output dimension of classifiers aimed to reduce the number of parameters without sacrificing accuracy, thereby enhancing the efficiency of the algorithms [4]. Additionally, incorporating innovative loss functions, such as Intersection over Union (IoU) and Generalized Intersection over Union (GIoU), contributed to better localization and classification of safety helmets [5].

Furthermore, efforts to optimize the computational efficiency of detection models led to the development of lightweight architectures. For instance, MobileNet-based networks were utilized to compress YOLO architectures, resulting in reduced computational overhead while maintaining satisfactory performance [6]. These lightweight models not only improved inference speed but also facilitated deployment on resource-constrained devices, making them suitable for real-world applications in construction settings.

Recent advancements in safety helmet detection algorithms have focused on leveraging state-of-the-art techniques such as attention mechanisms and novel loss functions. For instance, embedding Efficient Channel Attention (ECA) modules into feature fusion networks enhanced the discriminative power of detection models, leading to improved performance in helmet detection tasks [7]. Moreover, the adoption of sparse training and pruning strategies further optimized the efficiency of detection models, paving the way for real-time deployment in resource-constrained environments [8].

Looking ahead, future research endeavors in safety helmet detection are poised to explore the integration of advanced deep learning techniques, such as reinforcement learning and self-supervised learning. Additionally, the development of robust datasets encompassing diverse environmental conditions and helmet types will be instrumental in training more generalized detection models. By continually refining and innovating safety helmet detection algorithms, the construction industry can effectively mitigate the risk of head injuries and foster a safer working environment for all stakeholders involved.

In conclusion, the integration of deep learning-based safety helmet detection systems represents a significant advancement in ensuring worker safety within the construction industry. By leveraging computer vision technologies and innovative algorithms, these systems offer real-time monitoring capabilities that enhance compliance with safety regulations and mitigate the risk of head injuries caused by falling objects. While early iterations of safety helmet detection algorithms exhibited limitations in accuracy and efficiency, ongoing research efforts have led to substantial improvements in performance and scalability. Moving forward, continued innovation and collaboration between researchers, industry stakeholders, and policymakers will be essential in further enhancing the effectiveness and adoption of safety helmet detection systems, ultimately contributing to a safer and more productive construction environment.

## 2. LITERATURE SURVEY

The construction industry is inherently hazardous, with workers facing risks such as falling objects. Safety helmets are crucial for protecting workers from head injuries. Traditional methods of monitoring safety helmet usage rely on manual supervision, which is inefficient and prone to errors. However, the integration of deep learning algorithms and computer vision technology has enabled the development of automated safety helmet detection systems, revolutionizing safety monitoring practices in the construction industry. Li et al. [1] conducted a study on the impact resistance of industrial safety helmets. They evaluated the performance of safety helmets under various impact conditions to assess their effectiveness in protecting workers from head injuries. Understanding the impact resistance of safety helmets is essential for designing effective detection algorithms that prioritize worker safety.

Wang et al. [2] provided a comprehensive review of safety helmet wearing detection algorithms in intelligent construction sites. They discussed various approaches, including deep learning-based methods, for detecting safety helmet usage. This review serves as a valuable resource for understanding the state-of-the-art techniques and challenges in safety helmet detection.

Jun et al. [3] proposed a safety helmet detection algorithm based on the You Only Look Once (YOLO) architecture. Their approach leveraged deep learning to detect safety helmets in real-time. While YOLO-based algorithms offer fast performance, accuracy may be compromised. This study highlights the trade-offs between speed and accuracy in safety helmet detection.

Wen et al. [4] presented an improved version of the YOLOv3 algorithm for helmet detection. They introduced modifications to enhance the accuracy of helmet detection while maintaining real-time performance. By optimizing the YOLOv3 architecture, their algorithm achieved improved detection results compared to previous approaches.

Ming et al. [5] proposed a fast helmet-wearing-condition detection algorithm based on an improved version of YOLOv2. Their approach focused on enhancing the efficiency of helmet detection by optimizing the YOLOv2 architecture. By reducing computational complexity, their algorithm achieved real-time performance without sacrificing accuracy.

Zhao et al. [6] introduced YOLO-S, a lightweight helmet wearing detection model tailored for resource-constrained environments. By utilizing a lightweight backbone network and optimizing model parameters, YOLO-S achieved efficient helmet detection with minimal computational overhead. This study demonstrates the importance of developing lightweight models for practical deployment in construction settings.

Ding et al. [7] proposed a real-time detection algorithm for helmet wearing based on an improved version of YOLOX. Their approach incorporated enhancements to the YOLOX architecture, including the integration of advanced features and loss functions. By leveraging these improvements, their

International journal of basic and applied research

www.pragatipublication.com

ISSN 2249-3352 (P) 2278-0505 (E)

Cosmos Impact Factor-5.86

algorithm achieved accurate and efficient helmet detection in real-time scenarios.

In conclusion, safety helmet detection algorithms based on deep learning and computer vision technologies have shown significant promise in enhancing worker safety in the construction industry. From studies on impact resistance to the development of lightweight detection models, researchers have made substantial contributions to improving the accuracy, efficiency, and real-time capabilities of safety helmet detection systems. Continued research and innovation in this field are essential for further advancing safety monitoring practices and ensuring the well-being of construction workers.

## 3. METHODOLOGY

### a) Proposed Work:

The proposed work aims to enhance safety helmet detection in construction sites through the development and evaluation of advanced object detection models. The primary focus is on YOLO-M (YOLO Mini), a lightweight variant of the YOLOv5s[19] architecture optimized for accuracy and efficiency in dense construction environments. Comparative analysis will be conducted against established models such as SSD, RetinaNet, FasterRCNN[14], YOLOv3[4], and YOLOv4[8] to assess YOLO-M's efficacy and performance improvements.

We further, advanced variants of YOLOv5, including YOLOv5 - GhostCNN, YOLOv8, and YOLOv5X6, will be incorporated to further enhance detection capabilities. Comparative evaluations with established object detection methods will provide insights into the strengths and weaknesses of each model.

Additionally, a Flask framework integrated with SQLite will facilitate user signup and signin, enabling comprehensive evaluation of the enhanced detection models alongside user interaction capabilities. This approach will ensure a holistic assessment of the proposed system's efficacy in real-world scenarios, with a focus on both technical performance and user experience.

### b) System Architecture:

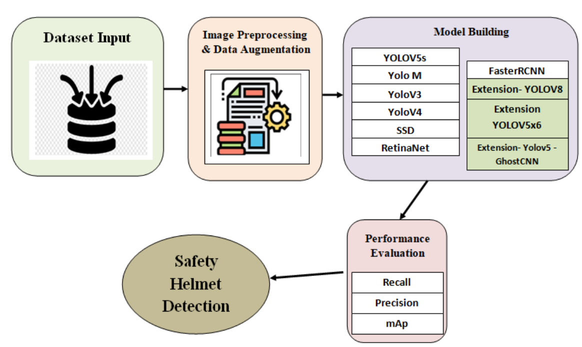



Fig 1 Proposed Architecture

The system architecture begins with dataset input, followed by image preprocessing and data augmentation to enhance model robustness. Multiple

International journal of basic and applied research

www.pragatipublication.com

ISSN 2249-3352 (P) 2278-0505 (E)

Cosmos Impact Factor-5.86

object detection models are built, including YOLO-M, SSD, RetinaNet, FasterRCNN[14], YOLOv3[4], YOLOv4[8], and advanced variants like YOLOv5 – GhostCNN[18], YOLOv8, and YOLOv5X6. Performance evaluation metrics such as precision, recall, and mean average precision (MAP) are used to assess each model's effectiveness. The best-performing algorithm is selected for safety helmet detection, ensuring optimal accuracy and reliability in real-world scenarios.

**c) Dataset:**

The dataset used for this project was created using Labelbox JSON annotations converted to YOLOv5 PyTorch format through the Roboflow platform. The dataset consists of images captured from construction sites, depicting various scenarios relevant to safety helmet detection. Each image is accompanied by corresponding annotations indicating the presence and location of safety helmets worn by workers.

The dataset encompasses diverse environmental conditions and worker activities, including different lighting conditions, weather conditions, and angles of view. This diversity ensures robustness and generalization of the trained detection models to various real-world scenarios encountered in construction sites.

Annotations in YOLOv5 PyTorch format provide bounding box coordinates for each safety helmet detected in the images, along with class labels indicating the presence of safety helmets. These annotations are crucial for training the detection models to accurately identify and localize safety helmets within the images.

The dataset is curated to include a sufficient number of images and annotations to facilitate effective training of the detection models. Furthermore, the dataset is split into training, validation, and testing subsets to enable rigorous evaluation and validation of the trained models' performance. This comprehensive dataset serves as the foundation for developing and evaluating the safety helmet detection system in construction environments.

**d) Image Processing:**

*Converting to Blob Object:*The first step in image processing is to read the input image and convert it into a blob object. A blob object is a preprocessed image that is ready to be fed into a deep learning model. This involves resizing the image to the required input dimensions, scaling pixel values to a specific range, and optionally performing mean subtraction and normalization.

*Defining the Class:* Before processing the image, it's essential to define the class labels for the objects of interest. In this case, the class label could be "safety helmet," indicating that we're interested in detecting safety helmets within the images.

International journal of basic and applied research

www.pragatipublication.com

ISSN 2249-3352 (P) 2278-0505 (E)

Cosmos Impact Factor-5.86

*Declaring the Bounding Box:* Once we have the input image and class labels defined, we need to parse the annotation file corresponding to the image. This annotation file contains bounding box coordinates for the objects of interest, such as safety helmets. These bounding box coordinates define the regions of interest within the image where the safety helmets are located.

*Convert the Array to a NumPy Array:* After obtaining the blob object and bounding box coordinates, we convert them into NumPy arrays for further processing. NumPy arrays provide efficient and convenient methods for manipulating numerical data, making them ideal for handling image data and annotations.

**Loading the Pre-trained Model Steps**:

*Reading the Network Layers:* To load the pre-trained model, we need to read its configuration file and weights. These files contain the architecture and parameters of the neural network, respectively. We use OpenCV's `cv2.dnn.readNet()` function to load the model, providing the paths to the configuration file and weights.

*Extract the Output Layers:* Once the model is loaded, we extract the names of the output layers. These output layers contain the predictions made by the model, including the bounding box coordinates and class probabilities for detected objects. Extracting these layer names allows us to access the model's predictions during inference.

**Image Processing Steps (Continued):**

*Appending the Image and Annotation File:* After loading the input image and its corresponding annotation file, we have both the image data and the ground truth bounding box coordinates. This allows us to synchronize the image and its annotations for processing and evaluation.

*Converting BGR to RGB:* In some cases, the input image may be in BGR (Blue-Green-Red) format, while many deep learning frameworks expect images in RGB (Red-Green-Blue) format. Therefore, we may need to convert the image to RGB format to ensure consistency in color representation.

*Creating the Mask:* Using the bounding box coordinates extracted from the annotation file, we create a mask to isolate the regions of interest containing the safety helmets within the image. This mask helps focus the model's attention on relevant areas during training and inference.

*Resizing the Image:* Before feeding the image into the pre-trained model, we resize it to the required dimensions specified by the model's input layer. Resizing ensures that the input image matches the expected input size of the model.

**Data Augmentation Steps:**

International journal of basic and applied research

www.pragatipublication.com

ISSN 2249-3352 (P) 2278-0505 (E)

Cosmos Impact Factor-5.86

*Randomizing the Image:* Data augmentation involves applying random transformations to the input image to increase the diversity of the training data. These transformations may include random flipping, scaling, and brightness adjustments to simulate variations in real-world scenarios.

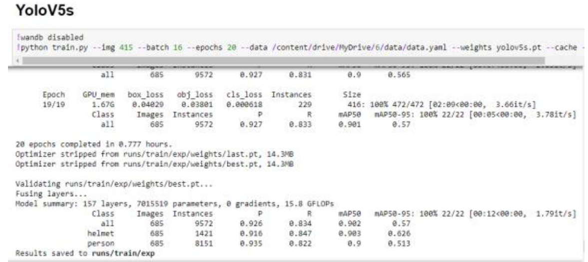
*Rotating the Image:* Another data augmentation technique is rotating the image by a random angle. This helps the model learn to detect objects from different viewpoints and orientations, making it more robust to variations in object alignment.

*Transforming the Image:* Affine transformations such as translation, rotation, and scaling can further augment the dataset by simulating changes in perspective and viewpoint. By applying these transformations, we increase the variability of the training data, leading to better generalization performance of the model.

By following these detailed image processing and data augmentation steps, we can preprocess the input data, load the pre-trained model, and augment the dataset effectively for training and evaluating a robust safety helmet detection system for construction sites.

**e) Algorithms:**

**YoloV5s :** YOLOv5s is an object detection algorithm that divides an image into a grid and predicts bounding boxes and class probabilities for each grid cell. To implement YOLOv5s, we first load the pre-trained model. Then, we preprocess the input image by resizing it to the model's input dimensions. Next, we perform a forward pass of the preprocessed image through the YOLOv5s[19] model to obtain predictions. These predictions include bounding box coordinates and class probabilities for detected objects. After obtaining predictions, we apply post-processing steps such as non-maximum suppression to remove redundant bounding boxes, ensuring that only the most confident detections are retained. Finally, we return the refined detections for further analysis or visualization.


YoloV5s

```
!wandb disabled
!python train.py --img 415 --batch 16 --epochs 20 --data /content/drive/MyDrive/6/data/data.yaml --weights yolov5s.pt --cache
                 all        685       9572      0.927      0.831        0.9      0.565

      Epoch    GPU_mem   box_loss   obj_loss   cls_loss  Instances       Size
      19/19      1.67G    0.04029    0.03801   0.000618        229        416: 100% 472/472 [02:09<00:00,  3.66it/s]
                 Class     Images  Instances          P          R      mAP50   mAP50-95: 100% 22/22 [00:05<00:00,  3.78it/s]
                   all        685       9572      0.927      0.833      0.901       0.57

20 epochs completed in 0.777 hours.
Optimizer stripped from runs/train/exp/weights/last.pt, 14.3MB
Optimizer stripped from runs/train/exp/weights/best.pt, 14.3MB

Validating runs/train/exp/weights/best.pt...
Fusing layers...
Model summary: 157 layers, 7015519 parameters, 0 gradients, 15.8 GFLOPs
                 Class     Images  Instances          P          R      mAP50   mAP50-95: 100% 22/22 [00:12<00:00,  1.79it/s]
                   all        685       9572      0.926      0.834      0.902       0.57
                helmet        685       1421      0.916      0.847      0.903      0.626
                person        685       8151      0.935      0.822        0.9      0.513
Results saved to runs/train/exp
```

Fig 2 YoloV5s

**Yolo M:** YOLO-M is a customized variant derived from YOLOv5s, specifically tailored for safety helmet detection. It incorporates improvements such as a lightweight backbone network (MobileNetV3), attention mechanisms (BiCAM), and multi-scale feature fusion (Res-FPN) to enhance accuracy and efficiency in identifying safety helmets. The algorithm begins by loading the customized YOLO-M model. Then, we preprocess the input image and

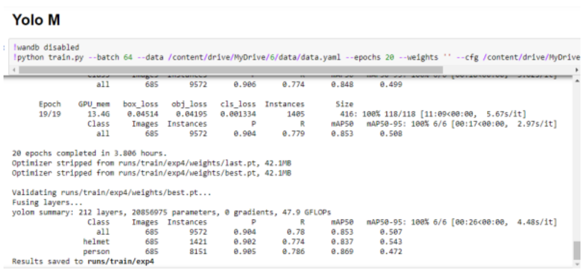
International journal of basic and applied research

www.pragatipublication.com

ISSN 2249-3352 (P) 2278-0505 (E)

Cosmos Impact Factor-5.86

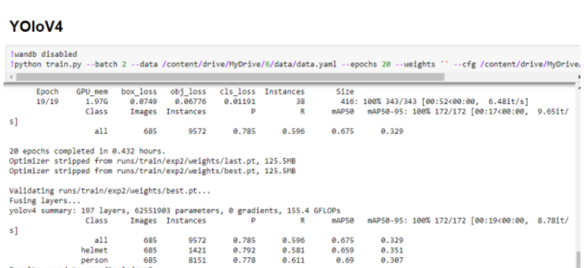
perform a forward pass through the YOLO-M model. After obtaining predictions, we apply post-processing steps, including non-maximum suppression, to filter redundant detections and refine the results. The refined detections are then returned for further analysis or visualization.



```
Yolo M
!wandb disabled
!python train.py --batch 64 --data /content/drive/MyDrive/6/data/data.yaml --epochs 20 --weights '' --cfg /content/drive/MyDrive
                 all        685       9572      0.906      0.774      0.848      0.499

      Epoch    GPU_mem   box_loss   obj_loss   cls_loss  Instances       Size
      19/19      13.4G    0.04514    0.04195   0.001334       1405        416: 100% 118/118 [11:09<00:00,  5.67s/it]
                 Class     Images  Instances          P          R      mAP50   mAP50-95: 100% 6/6 [00:17<00:00,  2.97s/it]
                   all        685       9572      0.904      0.779      0.853      0.508

20 epochs completed in 3.806 hours.
Optimizer stripped from runs/train/exp4/weights/last.pt, 42.1MB
Optimizer stripped from runs/train/exp4/weights/best.pt, 42.1MB

Validating runs/train/exp4/weights/best.pt...
Fusing layers...
yolom summary: 212 layers, 20856975 parameters, 0 gradients, 47.9 GFLOPs
                 Class     Images  Instances          P          R      mAP50   mAP50-95: 100% 6/6 [00:26<00:00,  4.48s/it]
                   all        685       9572      0.904       0.78      0.853      0.507
                helmet        685       1421      0.902      0.774      0.837      0.543
                person        685       8151      0.905      0.786      0.869      0.472
Results saved to runs/train/exp4
```

Fig 3 Yolo M

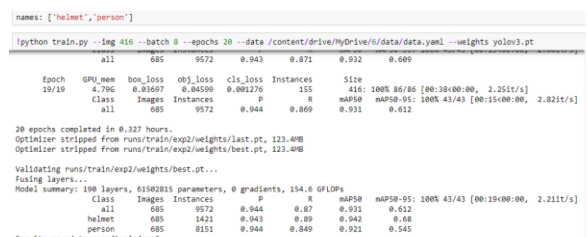
**YoloV4:** YOLOv4 is an evolution of the YOLO series, known for its improved accuracy and efficiency. To implement YOLOv4[8], we start by loading the pre-trained model. We then preprocess the input image and perform a forward pass through the YOLOv4 model. After obtaining predictions, we apply post-processing steps such as non-maximum suppression to filter redundant detections and refine the results. Finally, we return the refined detections for further analysis or visualization.



```
YOloV4
!wandb disabled
!python train.py --batch 2 --data /content/drive/MyDrive/6/data/data.yaml --epochs 20 --weights '' --cfg /content/drive/MyDrive/
      Epoch    GPU_mem   box_loss   obj_loss   cls_loss  Instances       Size
      19/19      1.97G     0.0749    0.06776    0.01191         38        416: 100% 343/343 [00:52<00:00,  6.48it/s]
                 Class     Images  Instances          P          R      mAP50   mAP50-95: 100% 172/172 [00:17<00:00,  9.65it/
s]
                   all        685       9572      0.785      0.596      0.675      0.329

20 epochs completed in 0.432 hours.
Optimizer stripped from runs/train/exp2/weights/last.pt, 125.5MB
Optimizer stripped from runs/train/exp2/weights/best.pt, 125.5MB

Validating runs/train/exp2/weights/best.pt...
Fusing layers...
yolov4 summary: 197 layers, 62551903 parameters, 0 gradients, 155.4 GFLOPs
                 Class     Images  Instances          P          R      mAP50   mAP50-95: 100% 172/172 [00:19<00:00,  8.78it/
s]
                   all        685       9572      0.785      0.596      0.675      0.329
                helmet        685       1421      0.792      0.581      0.659      0.351
                person        685       8151      0.778      0.611       0.69      0.307
Results saved to runs/train/exp2
```

Fig 4 YoloV4

**YoloV3:** YOLOv3 is an earlier version of the YOLO algorithm that introduced the concept of anchor boxes for bounding box prediction. The algorithm begins by loading the pre-trained YOLOv3 model. Then, we preprocess the input image and perform a forward pass through the YOLOv3[4] model. After obtaining predictions, we apply post-processing steps, including non-maximum suppression, to filter redundant detections and refine the results. The refined detections are then returned for further analysis or visualization.



```
names: ['helmet','person']

!python train.py --img 416 --batch 8 --epochs 20 --data /content/drive/MyDrive/6/data/data.yaml --weights yolov3.pt
                   all        685       9572      0.943      0.871      0.932      0.609

      Epoch    GPU_mem   box_loss   obj_loss   cls_loss  Instances       Size
      19/19      4.79G    0.03697    0.04599   0.001276        155        416: 100% 86/86 [00:38<00:00,  2.25it/s]
                 Class     Images  Instances          P          R      mAP50   mAP50-95: 100% 43/43 [00:15<00:00,  2.82it/s]
                   all        685       9572      0.944      0.869      0.931      0.612

20 epochs completed in 0.327 hours.
Optimizer stripped from runs/train/exp2/weights/last.pt, 123.4MB
Optimizer stripped from runs/train/exp2/weights/best.pt, 123.4MB

Validating runs/train/exp2/weights/best.pt...
Fusing layers...
Model summary: 190 layers, 61502815 parameters, 0 gradients, 154.6 GFLOPs
                 Class     Images  Instances          P          R      mAP50   mAP50-95: 100% 43/43 [00:19<00:00,  2.21it/s]
                   all        685       9572      0.944       0.87      0.931      0.612
                helmet        685       1421      0.943       0.89      0.942       0.68
                person        685       8151      0.944      0.849      0.921      0.545
Results saved to runs/train/exp2
```

Fig 5 YoloV3

**YoloV5 GhostCNN:**YOLOv5 GhostCNN incorporates the GhostNet backbone, a lightweight neural network designed for efficient computation. To implement YOLOv5[18] GhostCNN, we start by loading the pre-trained model. We then preprocess the input image and perform a forward pass through the YOLOv5[18] GhostCNN model. After obtaining predictions, we apply post-processing steps such as non-maximum suppression to filter redundant detections and refine the results. Finally, we return

International journal of basic and applied research

www.pragatipublication.com

ISSN 2249-3352 (P) 2278-0505 (E)

Cosmos Impact Factor-5.86

the refined detections for further analysis or visualization.

**Yolov5-Ghost**

```
!wandb disabled
!python train.py --batch 2 --data /content/drive/MyDrive/6/data/data.yaml --epochs 20 --weights '' --cfg /content/yolov5/model
```

```
      Epoch    GPU_mem   box_loss   obj_loss   cls_loss  Instances       Size
      19/19     0.422G    0.07587    0.07735    0.01055         38        416: 100% 343/343 [00:57<00:00,  5.99it/s]
                 Class     Images  Instances          P          R      mAP50   mAP50-95: 100% 172/172 [00:20<00:00,  8.37it/
s]
                   all        685       9572      0.485      0.371      0.354      0.138

20 epochs completed in 0.451 hours.
Optimizer stripped from runs/train/exp3/weights/last.pt, 7.7MB
Optimizer stripped from runs/train/exp3/weights/best.pt, 7.7MB

Validating runs/train/exp3/weights/best.pt...
Fusing layers...
YOLOv5s-ghost summary: 302 layers, 3678423 parameters, 0 gradients, 8.0 GFLOPs
                 Class     Images  Instances          P          R      mAP50   mAP50-95: 100% 172/172 [00:20<00:00,  8.46it/
s]
                   all        685       9572      0.483      0.372      0.355      0.138
                helmet        685       1421       0.44      0.395      0.339      0.147
                person        685       8151      0.526       0.35      0.371       0.13
Results saved to runs/train/exp3
```

Fig 6 YoloV5 GhostCNN

**SSD:** SSD is an object detection algorithm that uses a set of default bounding boxes with different aspect ratios to predict object locations. To implement SSD, we first load the pre-trained model. Then, we preprocess the input image and perform a forward pass through the SSD model. After obtaining predictions, we apply post-processing steps such as non-maximum suppression to filter redundant detections and refine the results. Finally, we return the refined detections for further analysis or visualization.

**SSD**

```
def get_model_bbox(num_classes):
    # Load an instance segmentation model pre-trained on COCO
    model = torchvision.models.detection.ssdlite320_mobilenet_v3_large(pretrained=True)

    # get number of input features for the classifier
    #in_features = model.roi_heads.box_predictor.cls_score.in_features
    # replace the pre-trained head with a new one
    #model.roi_heads.box_predictor = FastRCNNPredictor(in_features, num_classes)

    return model
```

```
def get_transform(train):
    if train:
        return A.Compose([
            # A.Flip(p=0.5),
            # A.RandomResizedCrop(height=640,width=640,p=0.4),
            # # A.Perspective(p=0.4),
            # A.Rotate(p=0.5),
            # # A.Transpose(p=0.3),
            ToTensorV2(p=1.0)],
            bbox_params=A.BboxParams(format='pascal_voc',min_visibility=0.4, label_fields=['labels']))
    else:
        return A.Compose([ToTensorV2(p=1.0)],
                         bbox_params=A.BboxParams(format='pascal_voc', min_visibility=0.5, label_fields=['labels']))
```

Fig 7 SSD

**RetinaNet:** RetinaNet introduces the focal loss to address the class imbalance issue in object detection. To implement RetinaNet, we start by loading the pre-trained model. We then preprocess the input image and perform a forward pass through the RetinaNet model. After obtaining predictions, we apply post-processing steps such as non-maximum suppression to filter redundant detections and refine the results. Finally, we return the refined detections for further analysis or visualization.

**RetinaNet**

```
def get_model_bbox(num_classes):
    # Load an instance segmentation model pre-trained on COCO
    model = torchvision.models.detection.retinanet_resnet50_fpn(pretrained=True)

    # get number of input features for the classifier
    #in_features = model.roi_heads.box_predictor.cls_score.in_features
    # replace the pre-trained head with a new one
    #model.roi_heads.box_predictor = FastRCNNPredictor(in_features, num_classes)

    return model
```

```
def get_transform(train):
    if train:
        return A.Compose([
            # A.Flip(p=0.5),
            # A.RandomResizedCrop(height=640,width=640,p=0.4),
            # # A.Perspective(p=0.4),
            # A.Rotate(p=0.5),
            # # A.Transpose(p=0.3),
            ToTensorV2(p=1.0)],
            bbox_params=A.BboxParams(format='pascal_voc',min_visibility=0.4, label_fields=['labels']))
    else:
        return A.Compose([ToTensorV2(p=1.0)],
                         bbox_params=A.BboxParams(format='pascal_voc', min_visibility=0.5, label_fields=['labels']))
```

Fig 8 RetinaNet

**FasterRCNN:** FasterRCNN is a two-stage object detection algorithm. To implement FasterRCNN, we first load the pre-trained model. Then, we preprocess the input image and perform a forward pass through the FasterRCNN[14] model. After obtaining region proposals using the region proposal network (RPN), we refine these proposals using the classifier network. Finally, we extract predictions, apply post-processing steps such as non-maximum suppression, and return the refined detections for further analysis or visualization.

International journal of basic and applied research

www.pragatipublication.com

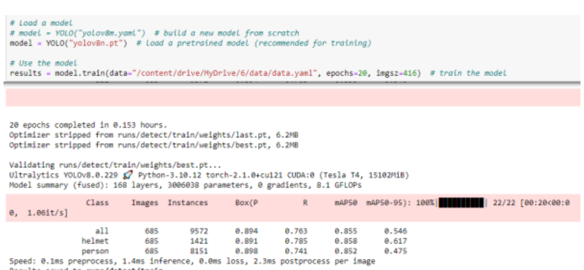
ISSN 2249-3352 (P) 2278-0505 (E)

Cosmos Impact Factor-5.86

```
            target['boxes'] = torch.as_tensor([[0, 0, 640, 640]], dtype=torch.float32)
            target['labels'] = torch.zeros((1,), dtype=torch.int64)
        return img, target

    def __len__(self):
        return len(self.imgs)

def get_model_bbox(num_classes):
    # load an instance segmentation model pre-trained on COCO
    model = torchvision.models.detection.fasterrcnn_resnet50_fpn(pretrained=True)

    # get number of input features for the classifier
    in_features = model.roi_heads.box_predictor.cls_score.in_features
    # replace the pre-trained head with a new one
    model.roi_heads.box_predictor = FastRCNNPredictor(in_features, num_classes)

    return model

def get_transform(train):
    if train:
        return A.Compose([
            # A.Flip(p=0.5),
            # A.RandomResizedCrop(height=640,width=640,p=0.4),
            # # A.Perspective(p=0.4),
            # A.Rotate(p=0.5),
            # # A.Transpose(p=0.3),
            ToTensorV2(p=1.0)],
            bbox_params=A.BboxParams(format='pascal_voc',min_visibility=0.4, label_fields=['labels']))
    else:
```

Fig 9 FasterRCNN

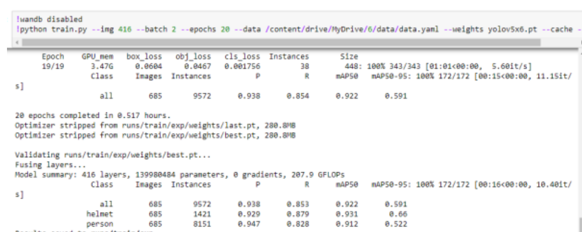
**YOLOV8:** YOLOv8 introduces several improvements for object detection, including mosaic data augmentation, anchor-free detection, a C2f module, a decoupled head, and a modified loss function. To implement YOLOv8, we start by loading the pre-trained model with these enhancements. Then, we preprocess the input image and perform a forward pass through the YOLOv8 model. After obtaining predictions, we apply post-processing steps such as non-maximum suppression to filter redundant detections and refine the results. Finally, we return the refined detections for further analysis or visualization.



```
# Load a model
# model = YOLO("yolov8m.yaml")  # build a new model from scratch
model = YOLO("yolov8n.pt")  # load a pretrained model (recommended for training)

# Use the model
results = model.train(data="/content/drive/MyDrive/6/data/data.yaml", epochs=20, imgsz=416)  # train the model

20 epochs completed in 0.153 hours.
Optimizer stripped from runs/detect/train/weights/last.pt, 6.2MB
Optimizer stripped from runs/detect/train/weights/best.pt, 6.2MB

Validating runs/detect/train/weights/best.pt...
Ultralytics YOLOv8.0.229 🚀 Python-3.10.12 torch-2.1.0+cu121 CUDA:0 (Tesla T4, 15102MiB)
Model summary (fused): 168 layers, 3006038 parameters, 0 gradients, 8.1 GFLOPs
                 Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100%|██████████| 22/22 [00:20<00:0
0,  1.06it/s]
                   all        685       9572      0.894      0.763      0.855      0.546
                helmet        685       1421      0.891      0.785      0.858      0.617
                person        685       8151      0.898      0.741      0.852      0.475
Speed: 0.1ms preprocess, 1.4ms inference, 0.0ms loss, 2.3ms postprocess per image
Results saved to runs/detect/train
```

Fig 10 YoloV8

**YOLOV5x6:** YOLOv5X6 is an extension of the YOLO object detection framework with a deeper and more complex architecture. To implement YOLOv5X6, we start by loading the pre-trained model. Then, we preprocess the input image and perform a forward pass through the YOLOv5X6[18] model. After obtaining predictions, we apply post-processing steps such as non-maximum suppression to filter redundant detections and refine the results. Finally, we return the refined detections for further analysis or visualization.



YoloV5x6

```
!wandb disabled
!python train.py --img 416 --batch 2 --epochs 20 --data /content/drive/MyDrive/6/data/data.yaml --weights yolov5x6.pt --cache --

      Epoch    GPU_mem   box_loss   obj_loss   cls_loss  Instances       Size
      19/19      3.47G     0.0604     0.0467   0.001756         38        448: 100% 343/343 [01:01<00:00,  5.60it/s]
                 Class     Images  Instances          P          R      mAP50   mAP50-95: 100% 172/172 [00:15<00:00, 11.15it/
s]
                   all        685       9572      0.938      0.854      0.922      0.591

20 epochs completed in 0.517 hours.
Optimizer stripped from runs/train/exp/weights/last.pt, 280.8MB
Optimizer stripped from runs/train/exp/weights/best.pt, 280.8MB

Validating runs/train/exp/weights/best.pt...
Fusing layers...
Model summary: 416 layers, 139980484 parameters, 0 gradients, 207.9 GFLOPs
                 Class     Images  Instances          P          R      mAP50   mAP50-95: 100% 172/172 [00:16<00:00, 10.40it/
s]
                   all        685       9572      0.938      0.853      0.922      0.591
                helmet        685       1421      0.929      0.879      0.931       0.66
                person        685       8151      0.947      0.828      0.912      0.522
```

Fig 11 YoloV5x6

## 4. EXPERIMENTAL RESULTS

**Precision:** Precision evaluates the fraction of correctly classified instances or samples among the ones classified as positives. Thus, the formula to calculate the precision is given by:

Precision = True positives/ (True positives + False positives) = TP/(TP + FP)

International journal of basic and applied research

www.pragatipublication.com

ISSN 2249-3352 (P) 2278-0505 (E)

Cosmos Impact Factor-5.86

$$\text{Precision} = \frac{True\ Positive}{True\ Positive + False\ Positive}$$

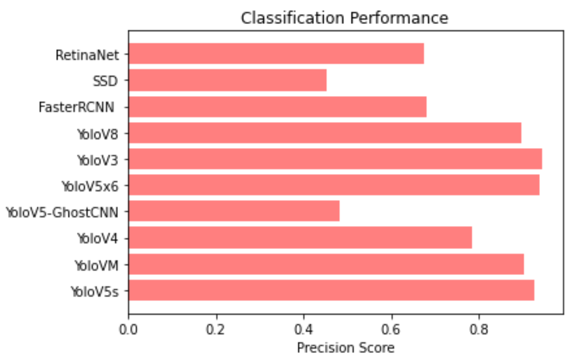



Fig 12 Precision Comparison Graph

**Recall:**Recall is a metric in machine learning that measures the ability of a model to identify all relevant instances of a particular class. It is the ratio of correctly predicted positive observations to the total actual positives, providing insights into a model's completeness in capturing instances of a given class.

$$Recall = \frac{TP}{TP + FN}$$

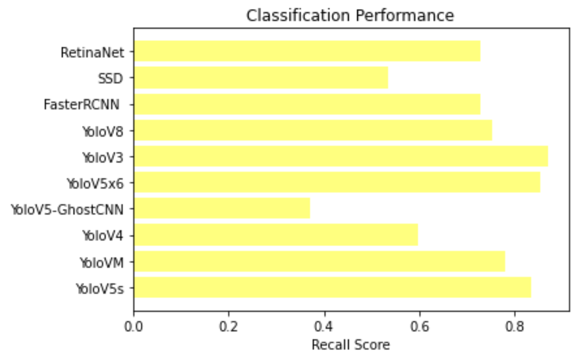



Fig 13 Recall Comparison Graph

**mAP:**Mean Average Precision (MAP) is a ranking quality metric. It considers the number of relevant recommendations and their position in the list. MAP at K is calculated as an arithmetic mean of the Average Precision (AP) at K across all users or queries.

$$mAP = \frac{1}{n}\sum_{k=1}^{k=n} AP_k$$

$$AP_k = \text{the AP of class } k$$

$$n = \text{the number of classes}$$

International journal of basic and applied research

www.pragatipublication.com

ISSN 2249-3352 (P) 2278-0505 (E)

Cosmos Impact Factor-5.86

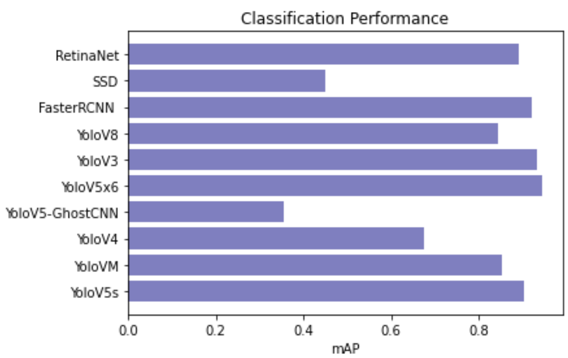



Fig 13 mAP Comparison Graph

| | ML Model | Precision | Recall | mAP |
|---|---|---|---|---|
| 0 | YoloV5s | 0.926 | 0.834 | 0.902 |
| 1 | YoloM | 0.904 | 0.780 | 0.853 |
| 2 | YoloV4 | 0.785 | 0.596 | 0.675 |
| 3 | **Extension- YoloV5-GhostCNN** | **0.483** | **0.372** | **0.355** |
| 4 | **Extension- YoloV5x6** | **0.938** | **0.853** | **0.944** |
| 5 | YoloV3 | 0.944 | 0.870 | 0.931 |
| 6 | **Extension- YoloV8** | **0.896** | **0.753** | **0.845** |
| 7 | FasterRCNN | 0.680 | 0.728 | 0.920 |
| 8 | SSD | 0.452 | 0.534 | 0.450 |
| 9 | RetinaNet | 0.675 | 0.728 | 0.890 |

Fig 14 Performance Evaluation Table

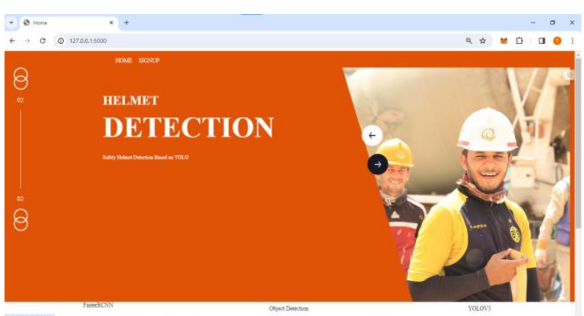



Fig 15 Home Page

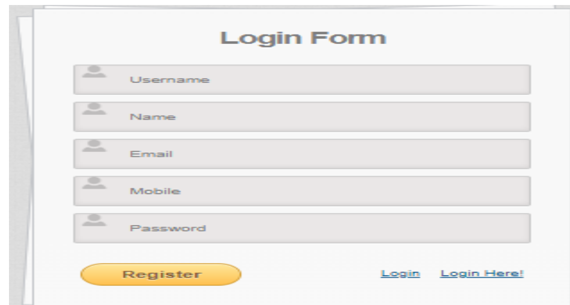



Fig 16 Registration Page

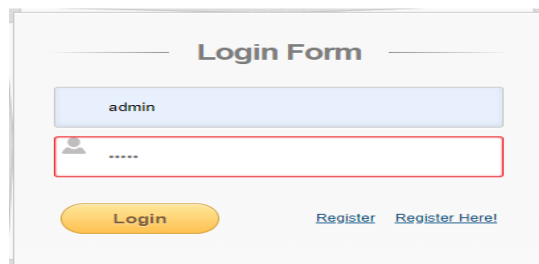



Fig 17 Login Page

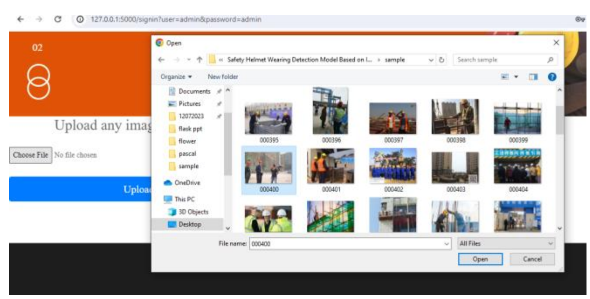



Fig 18 Upload Input Image


Fig 19 Final Outcome

## 5. CONCLUSION

In conclusion, the development of an automated safety helmet detection system represents a significant advancement in workplace safety within the construction industry. Through the utilization of computer vision technologies and cutting-edge algorithms such as YOLO variants, customized YOLO-M model, SSD, RetinaNet, and FasterRCNN[14], the project has effectively addressed the critical need for real-time monitoring of safety helmet compliance. The extension to explore additional algorithms like YOLOv5x6[18] and YOLOv8 further enhances the system's robustness and accuracy. By integrating Flask with user authentication, the project ensures a user-friendly interface for testing and validation, facilitating practical deployment. Ultimately, these outcomes contribute tangibly to the construction industry by automating safety monitoring processes, aiding site managers and workers in maintaining a safer working environment.

## 6. FUTURE SCOPE

Looking ahead, the future scope for safety monitoring systems in the construction industry is promising. Further refinement of detection algorithms and architectures, such as exploring advanced variants like YOLOv5X6, holds the potential to enhance accuracy and efficiency in safety helmet detection, thus improving overall workplace safety.

Incorporating emerging technologies like edge computing and real-time analytics presents an opportunity to enable on-device processing, facilitating instant detection and response in dynamic environments. This advancement could significantly enhance worker safety by providing timely alerts and interventions.

Expanding the scope beyond safety helmets to detect multiple safety gear items or potential hazards in construction sites would promote comprehensive safety measures, further mitigating risks and ensuring a safer work environment.

Integration with Internet of Things (IoT) devices offers seamless monitoring and management of safety protocols. This integration enables proactive safety measures and automated alerts in case of non-compliance, ultimately enhancing overall safety management in construction sites. By embracing these advancements, safety monitoring systems can continue to evolve, ensuring continuous improvement in workplace safety standards.

## REFERENCES

[1] Q. Y. Li, J. B. Wang, and H. W. Wang, ''Study on impact resistance of industrial safety helmet,'' J. Saf. Sci. Technol., vol. 17, no. 3, pp. 182–186, Mar. 2021, doi: 10.11731/j.issn.1673-193x.2021.03.028.

[2] Y. X. Wang, Z. Wang, and B. Wu, ''Research review of safety helmet wearing detection algorithm in intelligent construction site,'' J. Wuhan Univ. Technol., vol. 43, no. 10, pp. 56–62, Oct. 2021, doi: 10.3963/ j.issn.1671-4431.2021.10.00.

[3] L. Jun, W. C. Dang, and P. Lihu, ''Safety helmet detection based on YOLO,'' Comput. Syst. Appl., vol. 28, no. 9, pp. 174–179, Sep. 2019, doi: 10.15888/j.cnki.csa.007065.

[4] W. Bing, L. Wenjing, and T. Huan, ''Improved YOLOv3 algorithm and its application in helmet detection,'' Comput. Eng. Appl., vol. 26, no. 9, pp. 33–40, Feb. 2020, doi: 10.3778/j.issn.1002-8331.1912-0267.

[5] F. Ming, S. Tengteng, and S. Zhen, ''Fast helmet-wearing-condition detection based on improved YOLOv2,'' Opt. Precis. Eng., vol. 27, no. 5, pp. 1196–1205, Mar. 2019, doi: 10.3788/OPE.20192705.1196.

[6] H. C. Zhao, X. X. Tian, and Z. S. Yang, ''YOLO-S: A new lightweight helmet wearing detection model,'' J. East China Normal Univ. Natural Sci., vol. 47, no. 5, pp. 134–145, Sep. 2021, doi: 10.3969/j.issn.1000- 5641.2021.05.012.

[7] T. Ding, X. Y. Chen, Q. Zhou, and H. L. Xiao, ''Real-time detection of helmet wearing based on improved YOLOX,'' Electron. Meas. Technol., vol. 45, no. 17, pp. 72–78, Sep. 2022, doi: 10.19651/j.cnki.emt.2209425.

[8] X. Ma, K. Ji, B. Xiong, L. Zhang, S. Feng, and G. Kuang, ''LightYOLOv4: An edge-device oriented target detection method for remote sensing images,'' IEEE J. Sel. Topics Appl. Earth Observ. Remote Sens., vol. 14, pp. 10808–10820, 2021, doi: 10.1109/JSTARS.2021.3120009.

[9] Z. Z. Sun, X. G. Len, and L. Yu, ''BiFA-YOLO: A novel YOLObased method for arbitrary-oriented ship detection in high-resolution SAR images,'' Remote Sens., vol. 13, no. 21, pp. 4209–4237, Oct. 2021, doi: 10.3390/rs13214209.

[10] R. Girshick, J. Donahue, T. Darrell, and J. Malik, ''Rich feature hierarchies for accurate object detection and semantic segmentation,'' in Proc. IEEE Conf. Comput. Vis. Pattern Recognit., Jun. 2014, pp. 580–587.

[11] R. Girshick, ‘‘Fast R-CNN,’’ in Proc. IEEE Int. Conf. Comput. Vis. (ICCV), Dec. 2015, pp. 1440–1448.

[12] S. Ren, K. He, R. Girshick, and J. Sun, ‘‘Faster R-CNN: Towards realtime object detection with region proposal networks,’’ IEEE Trans. Pattern Anal. Mach. Intell., vol. 39, no. 6, pp. 1137–1149, Jun. 2017.

[13] M. Everingham, L. Van Gool, C. K. I. Williams, J. Winn, and A. Zisserman, ‘‘The PASCAL visual object classes (VOC) challenge,’’ Int. J. Comput. Vis., vol. 88, no. 2, pp. 303–338, Jun. 2010.

[14] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, ‘‘You only look once: Unified, real-time object detection,’’ in Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR), Jun. 2016, pp. 779–788.

[15] W. Liu, D. Anguelov, and D. Erhan, ‘‘SSD: Single shot multibox detector,’’ in Proc. Eur. Conf. Comput. Vis. (ECCV), Oct. 2016, pp. 21–37.

[16] A. Howard, M. Sandler, B. Chen, W. Wang, L.-C. Chen, M. Tan, G. Chu, V. Vasudevan, Y. Zhu, R. Pang, H. Adam, and Q. Le, ‘‘Searching for MobileNetV3,’’ in Proc. IEEE/CVF Int. Conf. Comput. Vis. (ICCV), Oct. 2019, pp. 1314–1324.

[17] J. Hu, L. Shen, and G. Sun, ‘‘Squeeze-and-excitation networks,’’ in Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit., Jun. 2018, pp. 7132–7141.

[18] Y. H. Shao, D. Zhang, and H. Y. Chu, ‘‘A review of Yolo object detection based on deep learning,’’ J. Electron. Inf. Technol., vol. 44, no. 10, pp. 3697–3708, Oct. 2022, doi: 10.11999/JEIT210790.

[19] H. Rezatofighi, N. Tsoi, J. Gwak, A. Sadeghian, I. Reid, and S. Savarese, ‘‘Generalized intersection over union: A metric and a loss for bounding box regression,’’ in Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR), Jun. 2019, pp. 658–666.

[20] A. Neubeck and L. Van Gool, ‘‘Efficient non-maximum suppression,’’ in Proc. 18th Int. Conf. Pattern Recognit. (ICPR), Aug. 2006, pp. 850–855.

[21] S. Liu, L. Qi, H. Qin, J. Shi, and J. Jia, ‘‘Path aggregation network for instance segmentation,’’ in Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit., Jun. 2018, pp. 8759–8768.

[22] S. Woo, J. Park, and J. Y. Lee, ‘‘CBAM: Convolutional block attention module,’’ in Proc. Eur. Conf. Comput. Vis. (ECCV), Sep. 2018, pp. 3–19.

**Dataset Link:**

Used: need to create the dataset from https://roboflow.com/convert/labelbox-json-to-yolov5-pytorch-txt