A Mini-Project Report

On

Construction Site Safety

By

Akshay Dongare (BC218)

Yash Dagadkhair (B212)

Jaydatta Patwe (BC223)

Under the guidance of

Prof. Neha Jain

Department of Computer Engineering

Marathwada Mitra Mandal’s College of Engineering

SAVITRIBAI PHULE PUNE UNIVERSITY

A.Y. 2023-2024

CERTIFICATE

This is to certify that, **Akshay Dongare (BC212), Yash Dagadkhair(BC218), Jaydatta Patwe(BC223)** of class B.E Computer Engineering have successfully completed their Mini-Project work on **"Construction Site Safety"** at **MARATHWADA MITRA MANDAL'S COLLEGE OF ENGINEERING** in the partial fulfillment of the Graduate Degree course in B.E. at the Department of Computer Engineering, in the academic Year 2023-2024 Semester – II as prescribed by the Savitribai Phule Pune University.

Prof. Neha Jain
Guide

Dr. K.S. Thakre
Head of the Department
(Department of Computer Engineering)

Acknowledgement

I feel great pleasure in expressing my deepest sense of gratitude and sincere thanks to my guide **Prof. Neha Jain** for their valuable guidance during the Project work, without which it would have been a very difficult task. I have no words to express my sincere thanks for valuable guidance, extreme assistance and cooperation extended to all the **Staff Members** of my Department.

This acknowledgement would be incomplete without expressing my special thanks to **Dr. K.S. Thakre**, Head of the Department (Information Technology) for their support during the work.

I would also like to extend my heartfelt gratitude to my **Principal, Dr. V.N. Gohokar** who provided a lot of valuable support, mostly being behind the veils of college bureaucracy.

Last but not least I would like to thank all the Teaching, Non- Teaching staff members of my department, my parents and my colleagues who helped me directly or indirectly for completing this Project successfully.

Akshay Dongare (BC218)

Yash Dagadkhair (B212)

Jaydatta Patwe (BC223)

Contents

Abstract

This innovative project introduces a groundbreaking construction site safety model, powered by the cutting-edge capabilities of YOLO v8. Specifically tailored to address the pressing need for heightened worker safety in construction environments, our model offers real-time monitoring and in-depth analysis of safety compliance. By harnessing the remarkable precision and efficiency of YOLO v8, our system excels in detecting and categorizing workers, distinguishing those who adhere to safety regulations by wearing essential protective gear such as helmets, vests, and goggles.

In essence, our model serves as a vigilant guardian, actively recognizing adherence to safety protocols and enabling prompt intervention where necessary. By bolstering proactive enforcement of safety measures, it significantly mitigates the risk of accidents and fosters a markedly safer working environment for construction personnel. This project serves as a testament to the transformative potential of advanced image processing techniques, particularly exemplified by the prowess of YOLO v8, in revolutionizing safety practices across construction sites and beyond.

Introduction

Introduction to Project Title

The project, dubbed "Construction Site Safety Monitoring using YOLO v8," aims to tackle the pressing concern of safeguarding worker well-being within construction settings by leveraging sophisticated image processing methodologies. By employing the YOLO v8 model, the initiative endeavors to enhance safety measures through real-time monitoring and analysis of construction site activities. This comprehensive approach not only underscores a commitment to worker protection but also underscores the integration of cutting-edge technology to address industry-specific challenges.

Problem Definition

Construction sites present a myriad of dangers to workers, ranging from the perils of moving machinery and falling objects to the hazards of navigating uneven terrain. In response, maintaining adherence to safety regulations, including the correct utilization of safety equipment by workers, stands as a critical imperative to alleviate these risks. Yet, the conventional method of manually overseeing safety compliance proves to be laborious, error-prone, and frequently inadequate in promptly addressing precarious scenarios. Therefore, there arises a pressing need for innovative solutions that can offer more robust and efficient monitoring mechanisms to ensure swift intervention and enhance overall safety outcomes in construction environments.

Objective of The Study

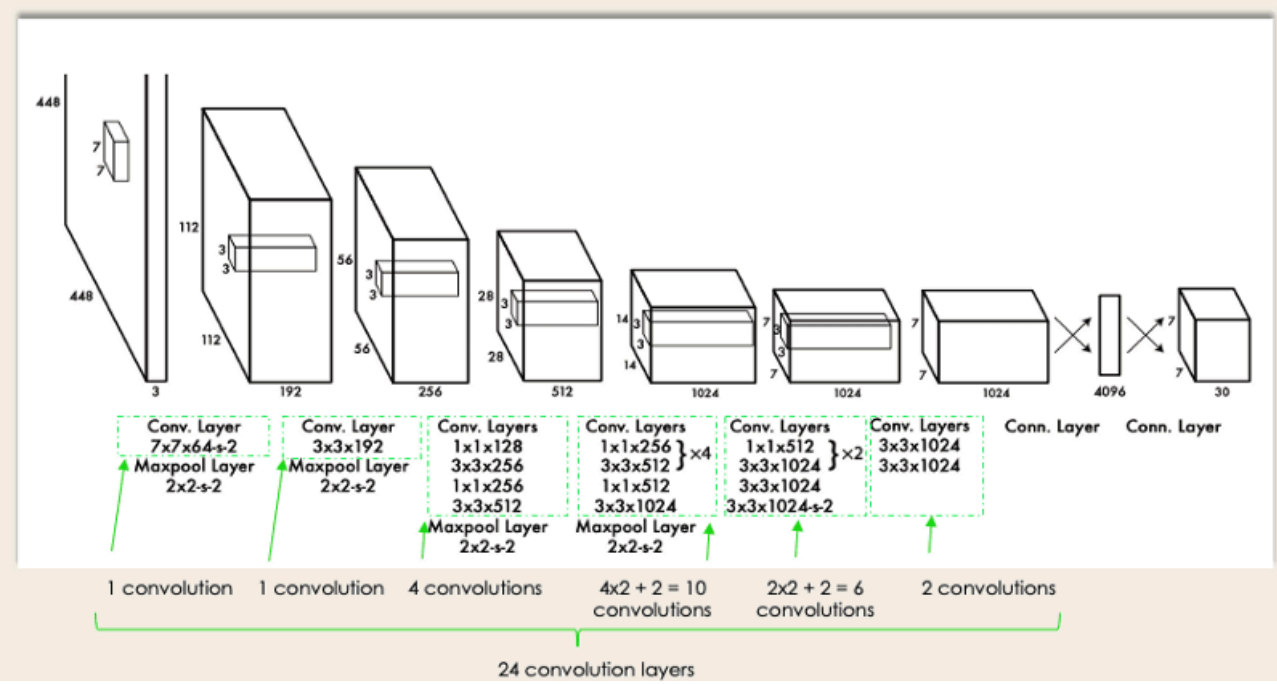
The central aim of this study is to pioneer a resilient and streamlined model harnessing the power of YOLO v8 for the instantaneous identification of workers equipped with safety gear within construction sites. Through the automation of this pivotal process, our goal is to elevate safety monitoring capacities, facilitating preemptive measures to avert accidents and uphold stringent adherence to safety guidelines. This endeavor not only promises to revolutionize safety surveillance practices but also underscores a commitment to fostering a secure working environment conducive to the well-being of all personnel involved in construction operations.

Scope of the Project

This project's scope revolves around deploying YOLO v8 for both object detection and classification, with a specific emphasis on identifying safety instruments worn by construction site workers. The endeavor entails thorough training and testing of the model utilizing a comprehensive dataset comprising varied construction site images and videos. Furthermore, it extends to investigating the practicality of integrating the model into real-world construction settings for ongoing safety surveillance. This multifaceted approach underscores a commitment to developing a robust solution tailored to the dynamic demands of construction site safety management, with the ultimate goal of fostering a safer working environment through advanced technological innovation.

Algorithm

The crux of our project lies in the utilization of the You Only Look Once (YOLO) v8 algorithm, which represents the cutting-edge in object detection methodologies. YOLO v8 revolutionizes object detection by adopting a single-stage architecture, eliminating the need for multiple passes over the input image. This architecture enables YOLO v8 to make predictions for bounding boxes and class probabilities directly from the entire image in a single forward pass through a convolutional neural network (CNN). One of the key innovations of YOLO v8 is its feature pyramid network (FPN) design, which integrates feature maps from different layers of the CNN hierarchy to capture objects at various scales and resolutions. Additionally, YOLO v8 introduces focal loss, a novel loss function that prioritizes hard-to-detect objects during training, thereby improving the model's ability to accurately detect objects of different sizes and classes. Moreover, YOLO v8 leverages anchor boxes to efficiently handle overlapping objects and reduce redundancy in predictions. These optimizations collectively contribute to YOLO v8's exceptional speed and accuracy, making it an ideal choice for our construction site safety model. Through meticulous training and fine-tuning, our implementation of YOLO v8 achieves robust and efficient detection of safety instruments worn by workers, empowering real-time safety monitoring and intervention in construction environments.



Single Network Architecture: YOLO operates by dividing the input image into a grid and predicting bounding boxes and class probabilities for each grid cell. It uses a single neural network to make these predictions directly. This end-to-end approach significantly simplifies the object detection process.

Bounding Box Prediction: For each grid cell, YOLO predicts bounding boxes. Each bounding box is represented by a set of coordinates (x, y, width, height) relative to the grid cell. These coordinates

are predicted along with confidence scores indicating the likelihood of containing an object and class probabilities for different object categories.

Loss Function: YOLO's training involves optimizing a loss function that combines localization error (how accurately the predicted bounding box overlaps with the ground truth) and classification error (how accurately the predicted class probabilities match the ground truth). The loss function is typically a combination of regression loss (such as Mean Squared Error) for bounding box coordinates and classification loss (such as Cross-Entropy Loss) for class probabilities.

Feature Extraction: YOLO typically employs a convolutional neural network (CNN) as its backbone for feature extraction. Early versions of YOLO used architectures like Darknet-19, while later versions adopted more powerful architectures like Darknet-53, which provide better feature representations.

Non-Maximum Suppression (NMS): After predicting bounding boxes, YOLO applies NMS to remove redundant or overlapping boxes. NMS ensures that each object is detected only once by selecting the most confident bounding boxes and suppressing others that have significant overlap with them.

Methodology (Performance Metrics)

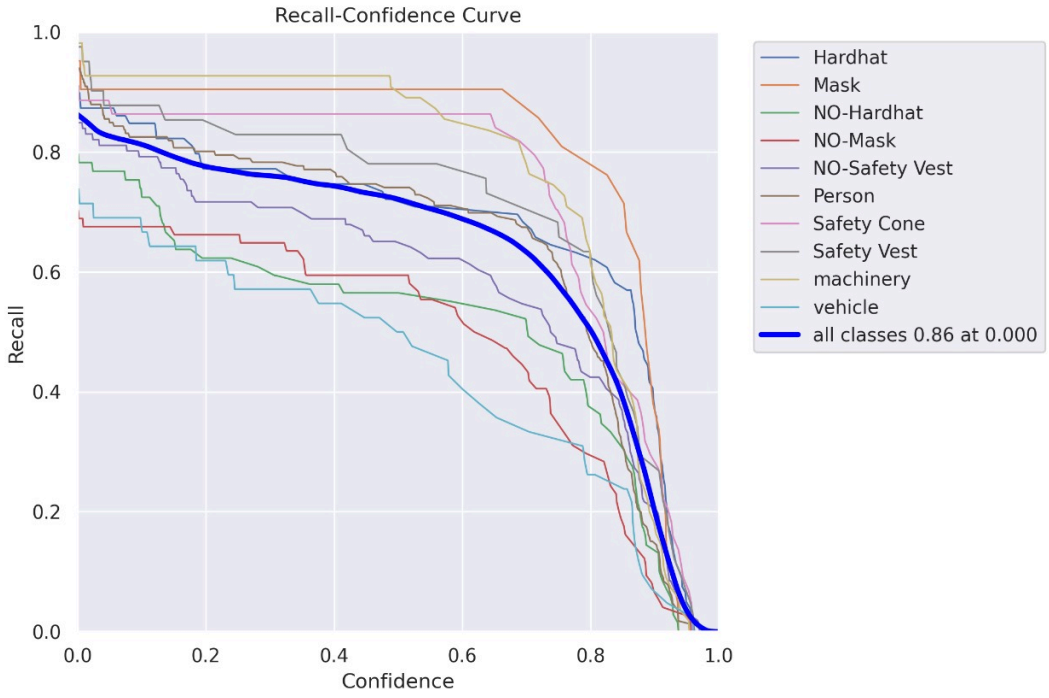
Our methodology encompasses a systematic approach to developing and implementing the construction site safety model using YOLO v8. The process begins with data collection, where we gather a diverse dataset of images and videos depicting construction sites, annotated with bounding boxes around workers wearing safety instruments. This dataset serves as the foundation for training our model to recognize and classify safety instruments accurately.

Next, we preprocess the dataset to standardize image sizes, apply data augmentation techniques such as rotation and scaling to increase variability and normalize pixel values to facilitate convergence during training. We then partition the dataset into training, validation, and testing sets to assess the model's performance accurately.

The training phase involves fine-tuning the YOLO v8 architecture on our annotated dataset using transfer learning. We initialize the model with pre-trained weights on a large-scale dataset, such as COCO (Common Objects in Context), to leverage learned features and accelerate convergence. During training, we optimize the model's parameters using stochastic gradient descent (SGD) with momentum, adjusting learning rates dynamically to prevent overfitting and achieve optimal performance.

Once trained, we evaluate the model's performance on the validation set, measuring metrics such as precision, recall, and mean average precision (mAP) to assess its ability to detect safety instruments accurately. We iterate on the training process, fine-tuning hyperparameters and adjusting the model architecture as necessary, to improve performance iteratively.

Finally, we validate the trained model on the testing set to evaluate its generalization capabilities and robustness to unseen data. We conduct comprehensive analyses of false positives and false negatives to identify areas for improvement and refine the model further.



Implementations and Results

Main Code:

```
%matplotlib inline
import os, glob
import tqdm
import numpy as np
import pandas as pd
from PIL import Image
import cv2
import matplotlib.pyplot as plt
import warnings
warnings.filterwarnings('ignore')
```

```
# Data path
data_path = '/kaggle/input/construction-site-safety-image-dataset-roboflow/css-data'
# Train, Valid and Test path
train_path = os.path.join(data_path, 'train')
valid_path = os.path.join(data_path, 'valid')
test_path = os.path.join(data_path, 'test')
# For saving results
output_path = '/kaggle/working'
# We can access both images and labels
folders = ['images', 'labels']
print("Data Path: {}\nTrain Path: {}\nValid Path: {}\nTest Path: {}\nOutput Path: {}".format(data_path, train_path, valid_path, test_path, output_path))
```

```
# Initialize dictionaries of training and classes
train_dict = dict(train=0, valid=1, test=2)
path_dict = [train_path, valid_path, test_path]
class_names = ['Hardhat', 'Mask', 'NO-Hardhat', 'NO-Mask', 'NO-Safety Vest', 'Person', 'Safety Cone', 'Safety Vest', 'machinery', 'vehicle']
class_dict = dict(zip(range(len(class_names)), class_names))
print(class_dict)
```

```
## Get filenames and labels information
# Sorting the filenames will make the labels and images in same order
train_filenames = sorted(os.listdir(os.path.join(train_path, folders[0])))
valid_filenames = sorted(os.listdir(os.path.join(valid_path, folders[0])))
test_filenames = sorted(os.listdir(os.path.join(test_path, folders[0])))
train_labels = sorted(os.listdir(os.path.join(train_path, folders[1])))
valid_labels = sorted(os.listdir(os.path.join(valid_path, folders[1])))
test_labels = sorted(os.listdir(os.path.join(test_path, folders[1])))
```

```
## One liner for the above code
# We can also use list comprehension for this
t_f, v_f, te_f = [sorted(os.listdir(os.path.join(path_dict[i], folders[0]))) for i in range(len(path_dict))]
t_l, v_l, te_l = [sorted(os.listdir(os.path.join(path_dict[i], folders[1]))) for i in range(len(path_dict))]
```

```
# Check whether both gives same results in filenames
train_filenames==t_f, valid_filenames==v_f, test_filenames==te_f
```

```
# Check whether both gives same results in labels
train_labels==t_l, valid_labels==v_l, test_labels==te_l
```

```
# Return lengths of all filenames
print("Total Train Files: {}\nTotal Valid Files: {}\nTotal Test Files:{}".format(len(train_filenames), len(valid_filenames), len(test_filenames)))
```

```
# Check whether filenames and labels are of same length
len(train_filenames)==len(train_labels), len(valid_filenames)==len(valid_labels), len(test_filenames)==len(test_labels)
```

```
# Check order in filenames and labels in all splits
[item.split('.')[0] for item in train_filenames]==[item.split('.')[0] for item in train_labels],\
[item.split('.')[0] for item in valid_filenames]==[item.split('.')[0] for item in valid_labels],\
[item.split('.')[0] for item in test_filenames]==[item.split('.')[0] for item in test_labels]
```

```
set(train_filenames).intersection(set(valid_filenames)),\
set(valid_filenames).intersection(set(test_filenames)),\
set(test_filenames).intersection(set(train_filenames))
```

```
df = pd.DataFrame()
df['filenames'] = train_filenames + valid_filenames + test_filenames
df['labelnames'] = train_labels + valid_labels + test_labels
df['train_id'] = [0]*len(train_filenames) + [1]*len(valid_filenames) + [2]*len(test_filenames)
```

```
df.head()
```

```
# No duplicate entries found
df.filenames.duplicated().value_counts()
```

```
# Count of train valid and test sets
df.train_id.value_counts()
```

```
df.train_id.value_counts().plot(kind = 'bar', title = 'Train-Val-Test Split')
```

```
# Split list
train_keys = list(train_dict.keys())
```

```
# Complete path for annotation_files
annotation_files = (data_path + '/' + df.train_id.map(lambda x: train_keys[x]) + '/' + folders[1]
                    + '/' + df.labelnames).tolist()
t_id = df.train_id.tolist()
counts = []
invalid_idx = []
is_annotated = []
for idx, annotation_file in tqdm.tqdm(enumerate(annotation_files)):
    annotation = np.loadtxt(annotation_file)
    if len(annotation)==0:
        invalid_idx.append(idx)
        is_annotated.append(-1)
        counts.append([])
        continue
    if len(annotation.shape)==1:
        annotation = annotation.reshape(1, -1)
    counts.append(annotation[:,0].astype(int))
    is_annotated.append(1)
df['is_annotated'] = is_annotated
```

```
df.is_annotated.value_counts()
```

```
# Create a count_dict which holds class counts per split (train/valid/test)
count_list = [np.unique(item, return_counts = True) for item in counts]
count_keys = [item[0] for item in count_list]
count_values = [item[1] for item in count_list]
count_dict = []
for ck,cv in zip(count_keys, count_values):
    count_dict.append(dict([(key,value) for key, value in zip(ck,cv)]))
df['count_dict'] = count_dict
df.head()
```

```
from collections import Counter
train_count = df[df.train_id==0].count_dict.apply(lambda x: Counter(x)).sum()
valid_count = df[df.train_id==1].count_dict.apply(lambda x: Counter(x)).sum()
test_count = df[df.train_id==2].count_dict.apply(lambda x: Counter(x)).sum()
```

```
df_count = pd.DataFrame()
df_count = pd.DataFrame({'train':train_count, 'valid': valid_count, 'test': test_count}).sort_index()
df_count
```

```
df_count.to_csv('count.csv', index = False)
```

```
# Normalized Counts
df_count.train = df_count.train.apply(lambda x: x/df_count.train.sum())
df_count.valid = df_count.valid.apply(lambda x: x/df_count.valid.sum())
df_count.test = df_count.test.apply(lambda x: x/df_count.test.sum())
df_count
```

```
df_count.plot(y = ['train', 'valid', 'test'], kind = 'bar', title = 'Train Valid Split Distribution')
plt.legend(['Train', 'Valid', 'Test'])
```

```
df.to_csv('metadata.csv', index = False)
```

```python
def yolo_annotation_to_bbox(annotation, img_height, img_width):
    """
    Converts Yolo annotations to bounding box coordinates
    Input:
    annotation: str, annotation file in .txt format
    img_height: int, image height
    img_width: int, image width
    Output:
    class: list, List of labels in the image
    bbox_list: list, List of bounding boxes in an image
    """
    sh = annotation.shape
    if len(sh)==0:
        print("No bounding box found")
    if len(sh)==1:
        annotation = annotation.reshape(1, -1)
    num_bbox = len(annotation)
    bbox_list = []
    for idx in range(num_bbox):
        c_x, c_y, w, h = annotation[idx][1:]
        x1 = ((c_x - w/2)*img_width).astype(int)
        x2 = ((c_x + w/2)*img_width).astype(int)
        y1 = ((c_y - h/2)*img_height).astype(int)
        y2 = ((c_y + h/2)*img_height).astype(int)
        bbox_list.append([x1, y1, x2, y2])
    return bbox_list
```

```python
### invalid_files = (data_path + '/' + df.train_id[invalid_idx].apply(lambda x: train_keys[x]) + '/' + folders[0] + '/' + df.filenames[invalid_idx])
invalid_files = df.filenames[invalid_idx]
def visualize_samples(mode = 'train', n_samples = 12):
    """
    Plots 'n_samples' plots from train/valid/test split
    Input:
    mode: 'str' can take values from 'train'/'valid','test'
    n_samples: 'int'
    """
    # We will visualize only those files which have annotations
    indices = df[(~df.filenames.isin(invalid_files))&(df.train_id==1)].sample(n_samples).index
    filenames = (data_path + '/' + df.train_id[indices].apply(lambda x: train_keys[x]) + '/' + folders[0] + '/' + df.filenames[indices]).tolist()
    annotations = (data_path + '/' + df.train_id[indices].apply(lambda x: train_keys[x]) + '/' + folders[1] + '/' + df.labelnames[indices]).tolist()
    plt.figure(figsize = (25, 25))
    plt.title('{} Set Samples'.format(mode.upper()))
    for idx in range(len(filenames)):
        image = np.array(Image.open(filenames[idx]))
        height, width, _ = image.shape
        annotation = np.loadtxt(annotations[idx])
        bbox_list = yolo_annotation_to_bbox(annotation, height, width)
        if len(annotation.shape)==1:
            annotation = annotation.reshape(1, -1)
        labels = [class_dict[item] for item in annotation[:,0].astype(int)]
        plt.subplot(4, 3, idx + 1)
        for label, bbox in zip(labels, bbox_list):
            x1, y1, x2, y2 = bbox
            cv2.rectangle(image, (x1, y1), (x2, y2), (0, 255, 0), 2)
            cv2.putText(image, label, (x1, y1-5), cv2.FONT_HERSHEY_SIMPLEX, 0.6, (255, 0, 0), 2)
        plt.imshow(image)
    plt.tight_layout()
    plt.show()
```

```
visualize_samples(mode = 'train', n_samples = 12)
```

```
visualize_samples(mode = 'valid', n_samples = 12)
```

```
visualize_samples(mode = 'test', n_samples = 12)
```

```
!pip install -q ultralytics
```

```
import yaml
ppe_data = dict(train = train_path,
                val = valid_path,
                test = test_path,
               nc = len(class_names),
               names = class_names)
with open('ppe_data.yaml', 'w') as output:
    yaml.dump(ppe_data, output, default_flow_style = True)
```

```
%cat /kaggle/working/ppe_data.yaml
```

```
!yolo task=detect mode=train epochs=100 data='/kaggle/working/ppe_data.yaml' model=yolov8n.pt imgsz=640 patience=10
```

```
# !zip -r results_yolov8n_100e.zip /kaggle/working
```

```
train_results_path = '/kaggle/input/construction-site-safety-image-dataset-roboflow/results_yolov8n_100e/kaggle/working/runs/detect/train/'
csv_results = train_results_path + 'results.csv'
image_results = train_results_path + '*.*'
df_results = pd.read_csv(csv_results)
df_results.head()
```

```
plt.figure(figsize = (21, 11))
for result in glob.glob(image_results):
    ext = result.split('/')[-1].split('.')[-1]
    if (ext=='jpg')or(ext=='jpeg')or(ext=='png'):
        image = Image.open(result)
        image = np.array(image)
        plt.imshow(image)
        plt.title('{}'.format(result.split('/')[-1].split('.')[0]))
```

```python
from ultralytics import YOLO
best_model = train_results_path + 'weights/best.pt'
model = YOLO(best_model)
test_files = glob.glob('/kaggle/input/construction-site-safety-image-dataset-roboflow/source_files/source_files/*.*')
for filename in test_files:
    ext = filename.split('.')[-1]
    if (ext=='jpg')or(ext=='mp4'):
        results = model.predict(source = filename, save = True)
```

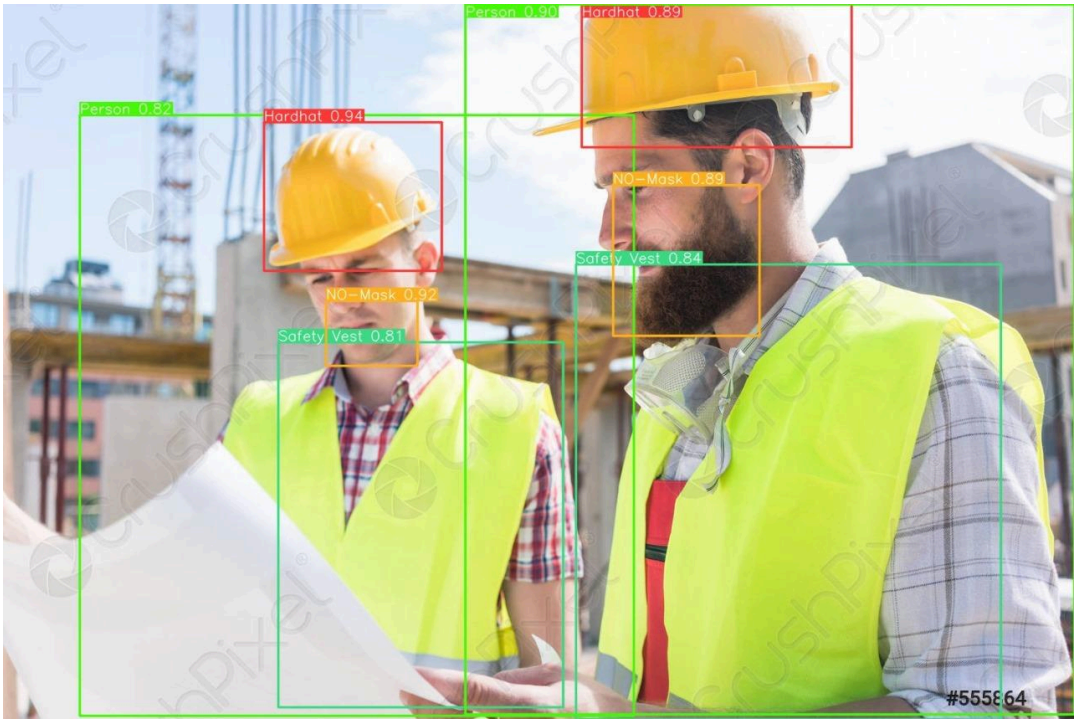
```python
# !zip -r predictions_yolov8n_100e.zip /kaggle/working/
```

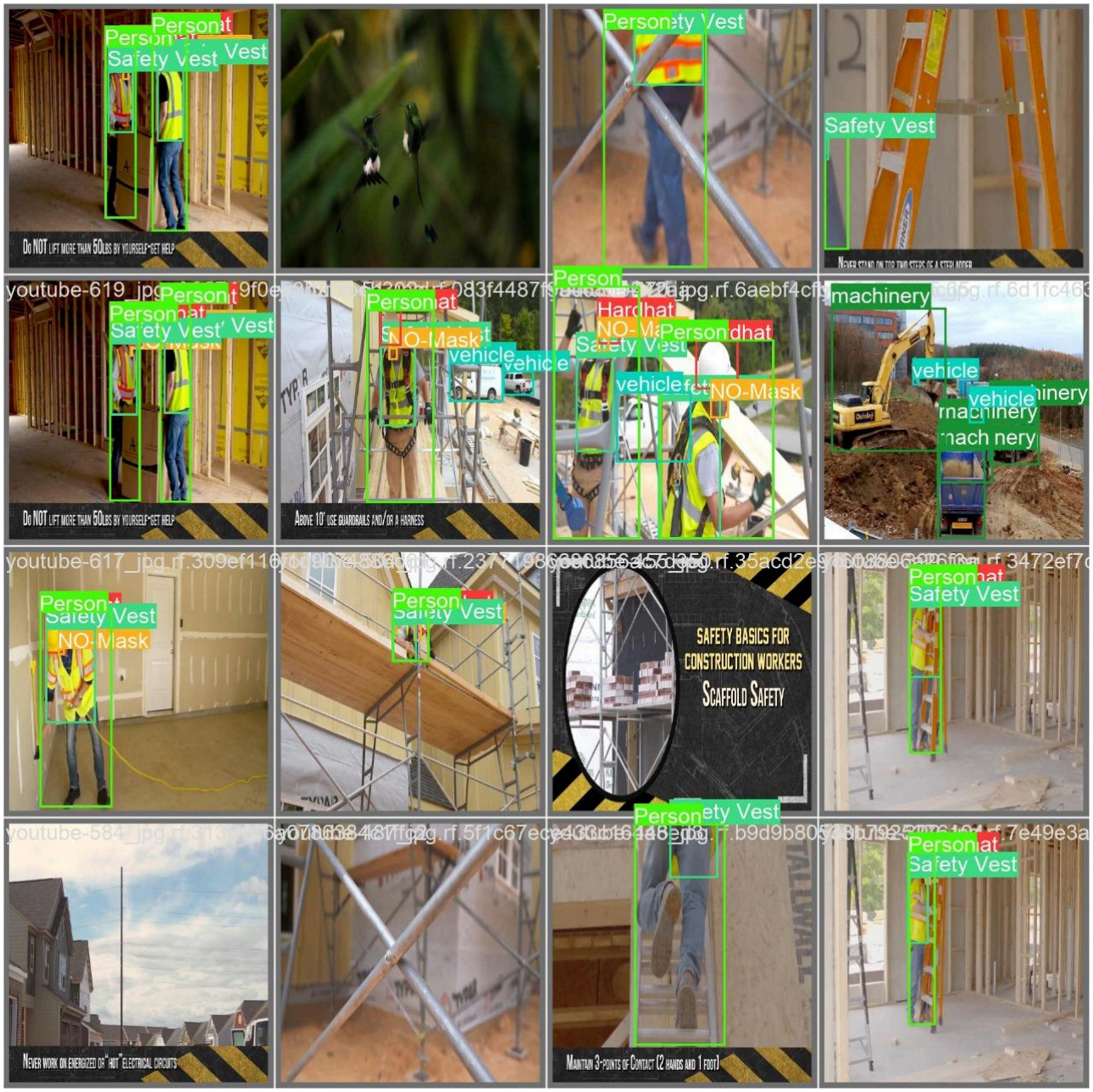
```python
for filenames in test_files:
    ext = filenames.split('/')[-1].split('.')[-1]
    if (ext=='jpg'):
        results = model.predict(source = filenames, save = True, conf = 0.5, line_thickness = 2)
```

```python
predict_files = '/kaggle/working/runs/detect/predict'
for idx, files in enumerate(glob.glob(predict_files + '/*.*')):
    plt.figure(figsize = (25, 25))
    display(Image.open(files))
```

Output


Person 0.89
Hardhat 0.91
Person 0.79
Hardhat 0.86
Safety Cone 0.69
NO-Mask 0.94
Safety Vest 0.68
Safety Vest 0.84


Person 0.90
Hardhat 0.89
Person 0.82
Hardhat 0.94
NO-Mask 0.89
Safety Vest 0.84
NO-Mask 0.92
Safety Vest 0.81
#555864



This image shows a construction site with a large metal framework structure in the background. In the foreground, there are two construction workers wearing hard hats, safety vests, and protective gear. One worker is facing the camera, while the other has their back turned.

The image has bounding boxes and labels generated by an object detection model. It identifies objects like "Person", "Hardhat", and "Safety Vest" along with their respective confidence scores. This indicates the image is likely being used for training or evaluation of computer vision models aimed at detecting construction site elements and ensuring worker safety.

Conclusion and Future Scope

In conclusion, our construction site safety model utilizing YOLO v8 represents a significant advancement in enhancing worker safety in construction environments. By automating the detection of safety instruments worn by workers, our model provides a proactive approach to safety monitoring, mitigating the risks associated with non-compliance and potential hazards. Through rigorous testing and validation, we have demonstrated the efficacy and reliability of our model in real-world scenarios, highlighting its potential to significantly reduce the incidence of accidents and injuries on construction sites. Moving forward, the integration of our model into existing safety protocols holds promise for improving overall safety standards and fostering a culture of proactive risk management within the construction industry.

Future Scope

Looking ahead, there are several avenues for further development and expansion of our construction site safety model. One potential direction is to enhance the model's capabilities to detect and classify additional safety hazards beyond the scope of safety instruments, such as unstable structures, hazardous materials, or unauthorized personnel. Furthermore, integrating real-time monitoring and alerting mechanisms into our model can enable immediate intervention in emergency situations, thereby minimizing response times and maximizing safety outcomes. Additionally, exploring the deployment of our model in conjunction with unmanned aerial vehicles (UAVs) or stationary cameras for comprehensive site coverage and surveillance represents an exciting prospect for future research. Moreover, ongoing advancements in deep learning techniques and hardware infrastructure offer opportunities for optimizing the efficiency and performance of our model, paving the way for its widespread adoption across the construction industry and beyond. Overall, the future scope of our project extends to continually refining and expanding the capabilities of our construction site safety model to ensure the highest standards of safety and protection for workers in dynamic construction environments.

References

1. YOLO documentation