Field Worker Safety Detection System

A project report submitted to

Dr. Geetha S

in partial fulfilment of the requirement for the course of

CSE 1902: Technical Answers to Real World Problems (TARP)

in

B.Tech. COMPUTER SCIENCE ENGINEERING

Vandalur – Kelambakkam Road

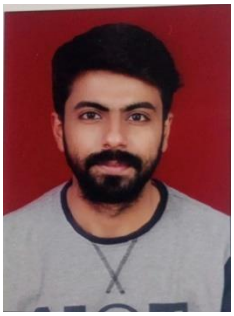
Chennai – 600127

April – 2023

❖ ABSTRACT

A field worker safety detection system is a piece of technological equipment that is intended to monitor and, more importantly, safeguard the safety of employees who are in distant or outside work environments. This system often consists of a mix of hardware and software that can identify possible risks and immediately notify employees as well as management of any potential dangers, so assisting in the prevention of accidents and injuries that may occur on the job. Before beginning work in mines, building sites, or any other hazardous environment, our product will check to see if personnel are clad in all the necessary protective gear and have attained a certain level of safety. A gas sensor that can alert employees in the event of a gas leak and a timer that can record when workers enter and exit the building are also included. Field worker safety detection systems have applications in a wide number of sectors, including agriculture, construction, oil & gas, and mining. They may give financial savings to businesses by minimizing the number of accidents and injuries that occur on the job, which in turn helps to enhance the safety of employees in hazardous or remote work environments. They can also assist improve worker safety in general.

❖ TEAM MEMBERS



TUSHAR MOOLCHANDANI – 20BAI1092
(tushar.moolchandani2020@vitstudent.ac.in)

AKSHAT KESHAV DHAMALE – 20BAI1308
(akshatkeshav.dhamale2020@vitstudent.ac.in)

KARTIK DEEPU – 20BCE1441
(kartik.deepu2020@vitstudent.ac.in)

SIDDHESH KUSHARE – 20BCE1448
(siddheshvikas.kushare2020@vitstudent.ac.in)

AYUSH KAPRI – 20BCE1458
(Ayush.kapri2020@vitstudent.ac.in)

❖ ACKNOWLEDGEMENT

Chapter-1. INTRODUCTION

1.1 Domain of the Problem

In today's fast-paced and rapidly changing world, safety has become a top priority for businesses, governments, and individuals alike. Workplace safety is of utmost importance, especially in high-risk environments such as construction sites, industrial facilities, and chemical plants. To ensure the safety of workers in these environments, safety equipment detection systems have become an essential tool.

One of the key components of safety equipment detection systems is the use of cameras to monitor workers and their use of personal protective equipment (PPE). These cameras are strategically placed at the entrance of worksite to capture images and videos of workers and their surroundings. By analysing this footage, the system can detect whether workers are wearing the appropriate PPE before entering a potentially dangerous area, such as hard hats, safety boots, gloves, and high-visibility vests.

To ensure the privacy of workers, the cameras used in safety equipment detection systems are typically designed to capture only the relevant information necessary to detect the presence of PPE and potential safety hazards. The footage is usually analysed in real-time by software that uses artificial intelligence and machine learning algorithms to detect and alert safety personnel to any potential safety violations.

Moreover, traditional safety monitoring methods, such as periodic safety inspections and spot checks, are often insufficient in detecting and addressing safety violations in real-time. This delay can lead to serious accidents and injuries.

Therefore, there is a need for a more effective and efficient safety monitoring system that can detect and alert safety personnel to PPE violations in real-time. A safety equipment detection system that uses cameras and sensors to monitor workers' use of PPE and identify potential safety hazards can help address this problem.

The problem statement for safety equipment detection is how to develop a system that can accurately detect the use of PPE by workers and identify potential safety hazards in real-time, thereby enhancing workplace safety and reducing the risk of accidents and injuries.

To develop a comprehensive understanding of safety equipment detection systems, it is essential to have domain knowledge about the following areas:

1. Personal Protective Equipment (PPE): PPE refers to equipment designed to protect workers from hazards that may cause injury or illness, such as hard hats, gloves, vests, etc. Domain knowledge about PPE is crucial in developing a safety equipment detection system that accurately detects the use of appropriate PPE.
2. Sensors and Cameras: Safety equipment detection systems typically use sensors and cameras to monitor workers and their use of PPE. Domain knowledge about the types of sensors and cameras available and their capabilities is essential in developing a system that can accurately detect PPE use and potential safety hazards.

3. Artificial Intelligence (AI) and Machine Learning: Safety equipment detection systems typically use AI and machine learning algorithms to analyse the data captured by sensors and cameras. Domain knowledge about AI and machine learning is essential in developing algorithms that can accurately identify and alert safety personnel to potential safety violations.
4. Safety Regulations: Workplace safety is regulated by various government agencies, such as the Occupational Safety and Health Administration (OSHA) in the United States. Domain knowledge about these regulations is essential in developing a safety equipment detection system that complies with legal requirements.
5. Industry-Specific Hazards: Different industries have different safety hazards. Domain knowledge about the specific hazards of a particular industry is essential in developing a safety equipment detection system that accurately detects potential safety violations and hazards.

1.2 Motivation

The health and safety of employees in any industry is a fundamental component of any business. When an employer demonstrates that it cares enough about its workers to take the precautions needed to keep them safe, a certain level of trust is established between that employer and its workforce.

Our projects' goals are to assist the firm in streamlining the process of ensuring that their personnel working on the pitch are safe by ensuring that only those who possess the necessary personal protective equipment are permitted to access the pitch. This not only ensures that safety regulations are followed, but it also helps the firm save money by reducing the amount of insurance it must pay out to workers who sustain injuries because they were not wearing their personal protective equipment (PPE).

1.3 Problem Statement

The goal is to design a system that can continually monitor the health and safety of employees in the field and offer real-time notifications in the event of any emergency or potentially dangerous circumstance. The system should be able to monitor the location of employees, their movements, and their activity levels, and then utilize this information to identify any possible dangers or mishaps. In addition, workers should be able to swiftly and easily transmit emergency warnings via the system, and managers should have real-time insight into the health and safety of their employees through the system. The system should be built to be user-friendly, durable, and dependable, and it should be able to function in areas that are either remote or hard.

Field safety is a must for the safety of a worker as well as the company. Many systems try to tackle this problem but generally follow the same systematic approach of using image detection to solve this problem.

In Safety Helmet Detection Based on YOLOv5 and other similar works such as Construction Safety Equipment Detection System, Visual Detection of Personal Protective Equipment and Safety Gear on Industry Workers, A Smart System for

Personal Protective Equipment Detection in Industrial Environments Based on Deep Learning, Deep Learning Detection of Personal Protective Equipment to Maintain Safety Compliance on Construction Sites, etc as mentioned in the literature review previously all discuss the same a similar generalized approach to tackle this problem.

- Detect worker.
- Detect location of where PPE is supposed to be.
- Using image detection check if the worker has his PPE on.
- Algorithms such as YOLO, ViBe, R-CNN, Fast R-CNN, Faster R-CNN are used to detect the images as these are the fastest and most efficient image detection algorithms.

1.4 Objectives

a. **Preventing accidents:** The system's real-time alerts and notifications can assist employees and managers in taking the necessary steps to stop accidents before they happen by making sure that all the workers are wearing all the required personal protective safety equipment.

b. **Monitoring for compliance:** The safety detection system seeks to monitor for worker adherence to safety protocols and directives, such as donning the proper protective gear and by doing so, adhering to safety procedures. The technology can see employees and notify them when there are safety policy violations. This can aid in fostering a culture of safety and ensuring that employees are adhering to the appropriate safety procedures to avoid mishaps.

c. **Productivity improvement**: The safety detection system can contribute to productivity improvement by limiting downtime and absenteeism due to illness or injury and reducing accidents and injuries. Due to their inability to fulfill their duties because of accidents or risks at work, employees who become ill or injured at work reduce output.

d. The system can help employees stay healthy and productive by reducing accidents and injuries, and the business can save money by avoiding the costs of missed productivity and worker compensation claims.

e. **Increasing safety culture**: By encouraging staff to take safety seriously and fostering a sense of responsibility for one another's safety, a safety detection system can aid in fostering a culture of safety inside an organization. The system can be used to monitor performance indicators for safety, such as the volume of safety incidents, near-misses, and safety audits. The company can show its commitment to safety by disseminating this information to staff members, managers, and supervisors. This will also motivate staff members to actively participate in ensuring a safe working environment.

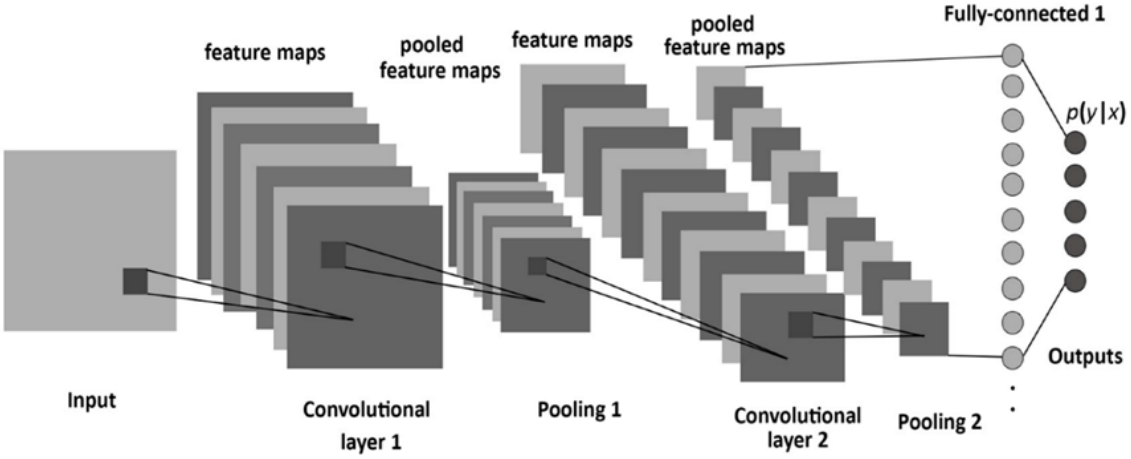
f. **Giving data insights:** The system can offer useful data insights that can be utilized to enhance safety processes and stop future occurrences by gathering data on safety incidents, near misses, and other safety-related events.
g. Making data-driven judgements about safety policies and procedures can be done using this data to spot patterns and trends in safety events and dangers.
h. For instance, the organization can take action to change or replace the equipment if the data reveals that it is linked to a high number of accidents or injuries in order to increase worker safety. Similar to this, the company can offer additional training or supervision to solve this issue if the data reveals that employees are routinely failing to follow safety rules in a particular region of the workplace.

1.5 Outlines

1.5.1 Deep Learning

For problems involving image detection, deep learning has proven to be an effective method. Image detection is the process of locating specific items or features inside an image and classifying them appropriately. Deep learning models use multiple-layered artificial neural networks to learn aspects of photos and categorize them using those features. For applications requiring image detection, convolutional neural networks (CNNs) are frequently used deep learning models. Convolutional, pooling, and fully linked layers are some of the layers that make up CNNs. By applying a series of learnable filters that move over the image, convolutional layers extract features from the input image. The feature maps are then down sampled by the pooling layers to reduce their computational complexity and dimensionality. Convolutional and pooling layers extract characteristics, which are then divided into many categories by fully connected layers.

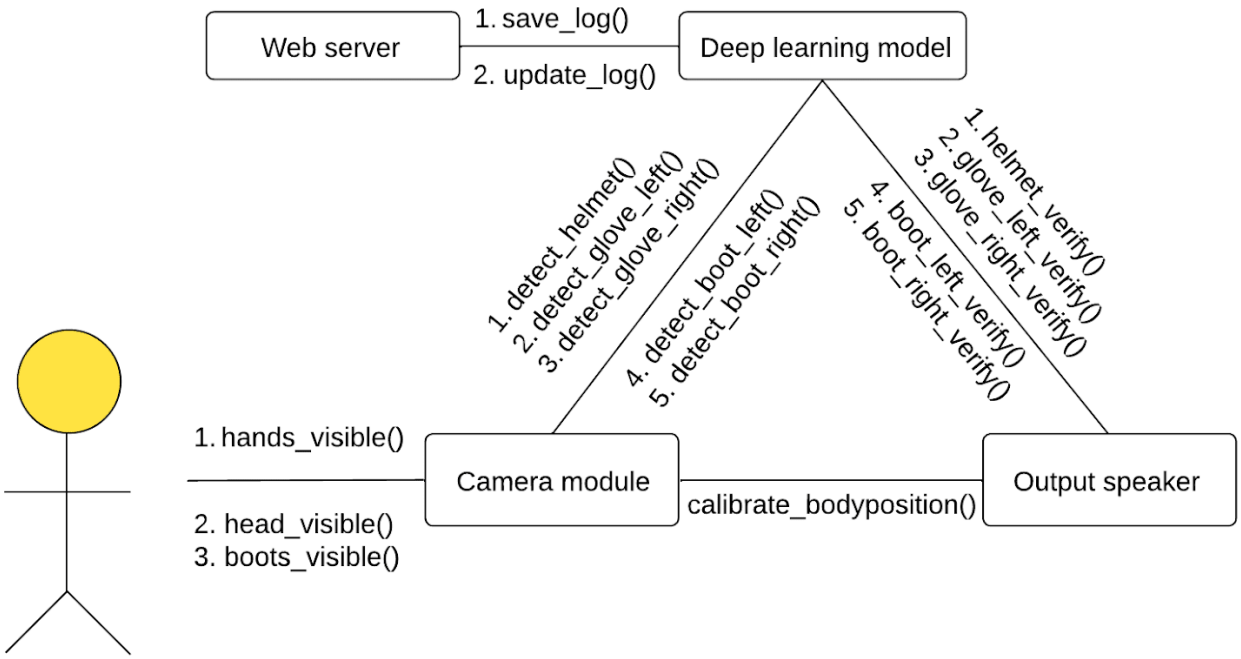
While training a CNN, a huge dataset of labelled images is fed into the network, and the network's weights are changed to reduce the difference between the predicted and actual labels. During the process of "backpropagation," which involves sending the error back across the network, the weights are changed. By feeding new photos through the network and generating predictions for the objects or characteristics in the images, a CNN can be used for image detection tasks after being trained. Deep learning applications for image detection include face recognition, object identification, and medical picture analysis.



CNN Architecture

1.5.2 Architecture

The UML communication diagram is a kind of interaction diagram that demonstrates how objects communicate with one another. The object diagram may be extended into a communication diagram, which displays the objects as well as the messages that go from one to another. A communication diagram will not only display the relationships between items, but it will also illustrate the messages that go from one to another. A communication diagram will not only display the relationships between items, but it will also illustrate the messages that the objects send to one another.



Camera module: Basic camera module attached to arduino circuit.

hands_visible() : Checks if person's hands are visible in the camera.

head_visible() : Checks if person's head is visible in the camera.

boots_visible() : Checks if person’s boots are visible in the camera.

Deep learning model: AI model deployed as an api endpoint where the camera will send images once all essential body parts are visible.

detect_helmet() : Detects if person is wearing helmet or not.

detect_glove_right() , detect_glove_left() : Detects if person is wearing gloves or not.

detect_boot_right(), detect_boot_left() : Detects if person is wearing boots or not.

Web server: Remote server where data recorded of a person and time will be saved and be updated regularly.

save_log() : Saves current data into the server.

update_log() : Updates current data according to ID.

Output Speaker: Speaker module attached to arduino board which will give outputs based on the processing done by deep learning model.

helmet_verify() , glove_left_verify(), glove_right_verify(), boot_left_verify(), boot_right_verify() : Returns true or false for detection of helmet, gloves and boots respectively.

Chapter-2. Literature Survey

2.1 Earlier Proposed Methods and Techniques

In research paper [1] , the authors propose an IoT-based system that uses IoT devices to collect data on air quality pollutants, such as carbon monoxide and methane, in the mine, this data is then sent to the Azure machine learning platform for analysis, where the authors have implemented a machine learning model that can predict the levels of pollutants in real-time. The authors proposed a framework for data preprocessing, feature extraction, and model selection to improve the performance of the prediction model. The system uses IoT devices to collect data on air quality pollutants, such as carbon monoxide and methane, in the mine. This data is then sent to the Azure machine learning platform for analysis and this has a 16.9% improvement from regression model

In [2] by Fangbo Zhou , et al. , the authors propose a system for detecting safety helmets in images and videos of construction and industrial workers to improve safety. They used the YOLOv5 (You Only Look Once version 5) object detection algorithm to detect safety helmets in the images and videos. The authors argue that this is an important task, as it can help to identify workers who are not wearing safety helmets, which can help to prevent accidents and injuries in these environments.object detection algorithms, such as Faster R-CNN and RetinaNet, and found that YOLOv5 had a better performance in terms of speed and accuracy.It is to be noted that an improvement of 9.75%, on average, is observed across the various models

In [3] by Kang Li, et al. . presents a system for automatically detecting whether workers are wearing safety helmets in construction and industrial environments. The system is intended to improve safety by identifying workers who are not wearing helmets and alerting them or a supervisor. the ViBe background modeling algorithm is employed. Moreover, based on the result of motion objects segmentation, real-time human classification framework C4 is applied to locate pedestrians. Uses of C4 framework significantly increases the object detection

The results of the study [4] by Arjya Das Mohammad, et al. show that their proposed deep learning model was able to accurately detect the presence or absence of a face mask with high precision. The model was tested on a variety of real-world scenarios, including different lighting conditions and diverse facial expressions, and it performed well in all cases. In conclusion, the authors propose that their deep learning model can be useful in various settings to enforce the use of face masks and prevent the spread of COVID-19. The use of computer vision technology in combination with deep learning offers a promising solution for real-time face mask detection in public spaces.Uses a dataset of images of people wearing and not wearing face masks to train their deep learning model. The exact details of the dataset, such as the number of images, their sources, and the diversity of the images, are not specified in the paper. However, it is stated that the dataset

was created specifically for the purpose of face mask detection and it was used to train the deep learning model developed in the study. This offers an accuracy up to 95.77%

[5] by Venkata Santosh Kumar et al. ,presents a system that uses computer vision and deep learning techniques to detect whether construction workers are wearing Personal Protective Equipment (PPE) such as safety helmets, gloves, and high-visibility vests. The system is intended to improve safety by identifying workers who are not wearing the required PPE and alerting them or a supervisor. The study makes use of the Convolutional Neural Networks model, which was developed by applying transfer learning to a base version of the YOLOv3 deep learning network

Nath, et al. of [6] propose using deep learning techniques to analyze images and videos of construction workers to detect whether they are wearing PPE. They present a system that uses a combination of deep neural networks and computer vision techniques to detect PPE in images and videos.

In [7] by Yogesh Kawade, et al. The authors propose a system that uses computer vision and deep learning techniques to automatically detect and classify workers who are wearing or not wearing required safety equipment, such as hard hats, safety vests, and safety glasses. The authors used convolutional neural networks (CNNs) to train the system on a large dataset of images of construction workers wearing and not wearing safety equipment. The results of the study show that the proposed system was able to accurately detect the presence or absence of required safety equipment with high precision. The system was tested on real-world construction sites and it performed well under various conditions, including different lighting conditions and diverse facial expressions. This paper uses a YOLOv3 model to apprehend hard hats and PPE vests. Ultimately identification of proper PPE using geometric relationships of the outputs of OpenPose and YOLOv3 were used with the following accuracies:

Hard hat - Precision : 94.19 Recall : 90.78

PPE vest - Precision : 99.59 Recall : 99.28

Jonathan Karlsson, et al. [8] presents a method for detecting the use of personal protective equipment (PPE) and safety gear on industrial workers. The authors use computer vision and deep learning techniques to automatically detect and classify workers who are wearing or not wearing required PPE and safety gear, such as hard hats, safety vests, and safety glasses. The system was trained on a large dataset of images of industrial workers wearing and not wearing PPE and safety gear. The results of the study show that the proposed system was able to accurately detect the presence or absence of required PPE and safety gear with high precision. The system was tested on real-world industrial settings and it performed well under various conditions, including different lighting conditions and diverse facial expressions. Dataset obtained from kaggle of hardhats and safety vests with positive and negative samples. YOLOv4 model is used for object detection with the following accuracy:

Precision : 3m - 7m 98 ~ 100%

Recall : 3m - 7m 93 ~ 100%

In [9] by Gioatan Gallo, et al. , the authors use computer vision and deep learning techniques to automatically recognize workers with or without their required PPE such as helmets, safety vests and goggles. A convolutional neural network (CNN) was trained using a large data set of images of industrial workers with and without PPE. The results of the study indicates that the proposed system was able to accurately detect the presence or absence of her required PPE with high accuracy. The system has been tested in a real industrial environment and performed well under a variety of conditions, including different lighting conditions and different facial expressions. Finally, the authors suggest that their PPE detection system may significantly improve safety in industrial environments. The use of computer vision and deep learning technology offers a promising solution for real-time monitoring and enforcement of his PPE use in the industry.

The research paper by ND Nath, et al. [10] presents a deep learning-based system for detecting personal protective equipment (PPE) in order to maintain safety compliance on construction sites. The system is designed to improve safety compliance by using computer vision techniques to automatically detect if workers are wearing the correct PPE. The paper describes the development and testing of the deep learning model and discusses the results, including its accuracy and potential applications in real-world construction site environments. Radio Frequency Identification (RFID) tags (Kelm et al., 2013), Local Area Network (LAN) (Barro-Torres et al., 2012), and short-range transponders (Naticchia et al., 2013), and checking sensor signals for cases of PPE non-compliance deep learning (DL) have created opportunities for using convolutional neural network (CNN) algorithms to more precisely detect PPE components. able to detect workers without any hat or vest (W), with hat (WH), with vest (WV), and with both hat and vest (WHV) with 90% accuracy, and the color of individual PPE components with 77% accuracy.

[11] by Xiao Ke, et al., presents a deep learning-based solution for detecting Personal Protective Equipment (PPE) on workers in real-time using green edge computing. The system is capable of detecting PPE with over 100 frames per second (FPS) and aims to improve worker safety in industries.channel pruning algorithm based on the BN layer scaling factor γ to further reduce the size of the detection model Reduction of computational effort by 32% and detection by 25%

Pedro Torres, et al. [12] presents a robust and real-time component for detecting Personal Protective Equipment (PPE) in an industrial setting. The author has made use of the YOLO-v3 for hard hat detection which gave an increase of 21.52% and 13.96% when comparing the YOLO-v4-AP1 and YOLO-v4-AP2

Saudi, et al. , in [13] presents an image detection model for evaluating the safety conditions of construction workers using Faster R-CNN. The model is used to detect and

analyze images of construction workers on a construction site to assess if they are following safe practices and wearing appropriate personal protective equipment. ResNet152 and Faster RCNN, Deep CNN Google Inception v3 offers accuracy of 70%.

[14] by Moohialdin, at al., proposes a novel computer vision (CV) system to detect the construction workers' PPE and postures in a real-time manner. Python data-labeling tool was used to annotate the selected datasets and the labeled datasets were used to build a detection model based on the TensorFlow environment and higher accuracy in classifying the postures over 72% and 64% in model testing and validation has been detected.

Deep Learning-Based Safety Helmet Detection in Engineering Management Based on Convolutional Neural Networks [15] proposes computer vision-based automatic solution for real-time detection of workers PPE. The presented work uses the SSD-MobileNet algorithm that is based on convolutional neural networks.The precision of the trained model is 95% and the recall is 77%

In [16], Ferdous, et al. , use computer vision (CV) based automatic PPE detection system that has been implemented to detect various types of PPE. (YOLO) family's anchor-free architecture, YOLOX. YOLOX-m performs approximately 3.29% better

[17] offers a way for detection in real time if workers are wearing PPE or not. The detector is developed using the YOLOv4 computer vision model which specially performs well in real time object detection. YOLOv4 computer vision model detector weights to TensorFlow format to check live detection performance and added features like live object count and record keeping. mAP of the object detector is found to be 79%

In "Applying the Haar-cascade Algorithm for Detecting Safety Equipment in Safety Management Systems for Multiple Working Environments" [18], Phuc, et al. ,apply the algorithm to calculate a score to determine the dangerousness of the current working environment based on the safety equipment and working environment. With this data, the system decides whether it is necessary to give a warning signal. principal component analysis Haar cascade algorithm were the algorithms used in this paper. CARs for this varied from 66.8% to 68.2% with very low error rate.

Qiu, et al., [19] proposes a method for detecting and evaluating the effectiveness of safety protection equipment in order to ensure its proper functioning and to prevent accidents. The method involves using sensors and algorithms to detect potential safety hazards and evaluate the performance of the protective equipment in response to those hazards. The goal is to improve safety by ensuring that protective equipment is functioning correctly and providing the necessary protection. The target detection algorithm used in this is based on linear distance and angle constraints

A deep learning based solution for construction equipment detection: from development to deployment [20] presents a deep learning based solution for detecting construction equipment in real-world environments. The solution is developed and tested, then

deployed to a construction site to demonstrate its ability to detect different types of construction equipment. The deep learning model uses computer vision techniques to identify and classify equipment in images and videos, with the goal of improving safety and efficiency in the construction industry. The results show the effectiveness of the proposed solution and its potential for practical deployment. The deep learning algorithms used above yield a 90% rate of accuracy

Marks, et al. in [21] presents a method for testing the effectiveness of proximity detection and alert technology used in construction equipment operations. The goal of the technology is to improve safety by detecting potential collisions and alerting operators to take corrective action. The method involves conducting tests in real-world environments to evaluate the performance of the technology in detecting potential hazards and providing effective warnings to operators. The results of the tests are used to determine the effectiveness of the technology and identify any areas for improvement to ensure safe and efficient operation of construction equipment emerging radio frequency (RF) remote sensing technology to demonstrate the ability of the test method to evaluate the capability of proximity detection and alert systems to provide alerts when heavy construction equipment and workers are in too close proximity to each other. An accuracy of 89% has been obtained by the authors.

"Automated detection of workers and heavy equipment on construction sites: A convolutional neural network approach" [22], proposes a method for automating the detection of workers and heavy equipment on construction sites using a Convolutional Neural Network (CNN) approach. The CNN model is trained on construction site images and videos to identify workers and heavy equipment and their location. The goal is to improve safety and efficiency on construction sites by providing real-time information about the presence and location of workers and equipment. The results of the study show that the proposed CNN-based method is effective for worker and equipment detection, and has potential for practical application on construction sites. Algorithms such as Improved Faster Regions with Convolutional Neural Network Features (IFaster R-CNN) approach is used to automatically detect the presence of objects in real-time.

Wang, et al., in [23] present a method for predicting safety hazards among construction workers and equipment using computer vision and deep learning techniques. The method involves collecting data from construction sites and using it to train a deep learning model to recognize and predict potential safety hazards. The goal is to improve safety on construction sites by providing real-time warnings and recommendations for avoiding hazards. The results show that the proposed approach is effective for predicting safety hazards and has potential for practical application on construction sites. Region based CNN framework named Faster R-CNN has been used to detect workers standing on scaffolds.

"Substation Safety Awareness Intelligent Model: Fast Personal Protective Equipment Detection using GNN Approach" [24] by Zhao, et al., proposes an intelligent model for

detecting personal protective equipment (PPE) in substation environments using a Graph Neural Network (GNN) approach. The goal is to improve safety awareness and compliance with PPE requirements by detecting workers who are not wearing required protective gear. The GNN model is trained on substation images and videos to identify workers and their PPE, with the goal of providing real-time warnings and increasing awareness of PPE requirements. The results show that the proposed GNN-based model is effective for PPE detection and has potential for practical application in substation environments. The author utilizes a few-shot based graph neural network (GNN) technique to detect PPE.

2.2 Limitation in the Present Models

In most of the works published, people have resorted to using popular images detection algorithms such as YOLO, R-CNN, Fast R-CNN, etc, each has its own downsides and advantages.

We know that YOLO (You Only Look Once) is a popular and effective approach to object detection in images, it also has some potential downsides.

- Limited accuracy for small objects: YOLO is known to struggle with detecting small objects accurately, as its design prioritizes detecting larger objects first. This can lead to missed detections or inaccurate bounding boxes for smaller objects in an image.
- Limited accuracy for occluded objects: When objects in an image are partially occluded, YOLO may have difficulty accurately detecting and localizing them. This is because it relies on identifying and classifying the entire object within the bounding box.
- Prone to false positives: Due to the way YOLO makes predictions, it can sometimes produce false positives by misclassifying certain regions of an image as objects.
- Computationally intensive: YOLO requires a significant amount of computational power, particularly for real-time applications. This can be a limiting factor for some use cases, particularly on lower-powered devices.
- Requires large datasets for training: As with any machine learning model, YOLO requires a large and diverse dataset for effective training. This can be a challenge for some applications, particularly those with limited access to high-quality training data.

Another famous algorithm used is R-CNN (Region-based Convolutional Neural Network), but it also has its own cons:

- Slow inference time: R-CNN involves multiple stages of computation, including generating region proposals, feature extraction, and classification. This can make inference times relatively slow, particularly compared to faster models like YOLO.
- High memory usage: R-CNN requires storing large feature maps for each region proposal, which can lead to high memory usage during both training and inference. This can be a limiting factor for some applications, particularly on lower-powered devices.

- Training can be slow: Due to its complexity, training an R-CNN model can be a slow and resource-intensive process, particularly for large datasets.
- Limited scalability: R-CNN is designed to work with a fixed set of region proposals, which can limit its ability to scale to larger datasets or applications where objects are densely packed or overlapping.
- May require additional pre-processing: R-CNN may require additional pre-processing steps, such as selective search or other region proposal algorithms, to generate high-quality proposals for object detection. These steps can add additional complexity to the overall system.

While successors of the R-CNN models can cover up some of its limitations, they are unable to do so for themselves have a few shortcomings such as

- Slow training time: Fast R-CNN still requires a significant amount of computational power and time to train, particularly for larger datasets. This can be a limiting factor for some applications.
- Limited scalability: Like R-CNN, Fast R-CNN is designed to work with a fixed set of region proposals, which can limit its ability to scale to larger datasets or applications with densely packed or overlapping objects.
- Computationally intensive: While Fast R-CNN is faster than R-CNN in terms of inference time, it still requires a significant amount of computational power for real-time applications.
- Limited accuracy for small objects: Like other region-based approaches, Fast R-CNN can struggle with accurately detecting and localizing small objects in images.
- May require additional pre-processing: Fast R-CNN may require additional pre-processing steps, such as selective search or other region proposal algorithms, to generate high-quality proposals for object detection. These steps can add additional complexity to the overall system.
- Computationally intensive: Faster R-CNN is a computationally intensive algorithm, which means it requires a lot of processing power and time to train and run. This can make it difficult to use in real-time applications.
- Limited to rectangular bounding boxes: Faster R-CNN is limited to detecting rectangular bounding boxes around objects. This means it may not work well for objects with irregular shapes, such as animals or plants.
- Limited to fixed aspect ratios: Faster R-CNN assumes that objects have a fixed aspect ratio, which can be a limitation for objects that do not follow this assumption.
- Not very effective for small objects: Faster R-CNN is not very effective at detecting small objects. This is because the size of the object relative to the image size is too small to provide enough contextual information for accurate detection.
- Limited to single-scale detection: Faster R-CNN is limited to single-scale detection, which means it can only detect objects at a fixed scale. This can be a limitation for objects that appear at different scales in an image.

- Limited by training data: Like any other deep learning model, Faster R-CNN requires a large amount of training data to achieve high accuracy. Limited training data can result in suboptimal performance.

Another algorithm used to solve this problem is ViBe and has the following disadvantages.

- Memory Usage: ViBe requires a significant amount of memory to store the background model for each pixel in a video frame. This can become a problem for high-resolution videos or long-duration videos, where the memory requirements can be significant.
- Initialization Time: ViBe requires an initialization phase before it can start subtracting the background from the video frames. During this phase, the algorithm needs to observe a certain number of frames to build the background model. This initialization time can be a disadvantage in scenarios where real-time processing is required.
- Sensitivity to Parameter Tuning: ViBe has several parameters that need to be tuned for optimal performance, such as the number of history frames used to build the background model and the number of matches required for a pixel to be considered foreground. Incorrect parameter tuning can lead to false detections or missed detections.
- Difficulty with Dynamic Backgrounds: ViBe works well for static background scenes, but it may not perform well in scenarios where the background is dynamic, such as in scenes with moving trees, water, or shadows. In such cases, the background model may not accurately represent the scene, leading to false detections or missed detections.
- Limited Adaptability: ViBe's background model is based on a fixed number of history frames, and it does not adapt well to gradual changes in the scene, such as lighting changes or slow-moving objects. This can result in false detections or missed detections.
- Limited Robustness: ViBe is vulnerable to sudden changes in the scene, such as camera movements or sudden illumination changes, which can cause false detections or missed detections.

2.3 Inference

The project's goal is to make sure that employees are following safety rules and wearing protective equipment such helmets, vests, gloves, masks, and boots. Because of this, building sites may become less dangerous places for employees to be. The project's ability to provide real-time monitoring of safety equipment use is a big advantage. This is especially helpful on big construction sites, where it may be difficult to monitor employee compliance with mandatory safety equipment wear. A safety officer's ability to monitor worker compliance with safety laws and take remedial action is greatly enhanced using an object detection model.

The cost of accidents on building sites might be reduced thanks to this effort. Injuries and illnesses sustained by workers account for roughly 4% of India's annual GDP loss, according to the National Safety Council (NSC). Medical expenses, time away from the job, and compensation payments all add up when a construction worker is injured on the job. By minimizing the frequency of accidents and injuries, this initiative may help lower these expenses and boost the construction industry's bottom line.

The initiative may also help foster a norm of safety on building sites. Employers that use an object detection model to check whether employees are wearing protective gear are sending a clear statement that they care about their employees' well-being. This may assist create a more secure workplace by encouraging employees to take safety precautions seriously. In conclusion, the construction sector in India might benefit from the improvements in safety measures made possible by the project aimed at recognizing safety equipment at building sites using an object detection model. The project may foster a culture of safety and lessen the monetary cost of accidents if personnel are required to wear the appropriate protective equipment. There are, nevertheless, obstacles to overcome in carrying out this project; thus, thorough preparation and sufficient funding are essential to guarantee its success.

Chapter-3. Proposed Method

3.1 Analysis of the Problem Statement

- Problem Definition: The primary goal of safety wear detection is to develop effective and dependable methods of detecting and monitoring safety wear usage in a variety of work environments, in order to ensure worker safety and well-being. Workers must wear appropriate safety gear to protect themselves from workplace hazards and reduce the risk of injury.
- Background Research: While workplace safety has been a significant priority for many firms, safety gear detection has grown in prominence in recent years. To protect workers from risks, it is critical to use safety equipment such as helmets, gloves, vests,boots and masks. Overall, background research in safety wear detection emphasises the relevance of this topic in promoting workplace safety, as well as the promise of modern technology to improve safety wear detection and worker protection.
- Gap Analysis: Despite major developments in safety wear detection systems in recent years, there are still some gaps in the area that must be filled. These are a few examples of safety wear detecting gaps:
    1. Accuracy of detection: One of the most significant gaps in safety wear detection is the accuracy of the sensors used to identify safety wear. Certain sensors may be insufficiently sensitive to detect particular types of safety wear, or they may be incapable of distinguishing between different types of wear.
    2. Acceptance of workers: Some workers may be hesitant to wearing safety equipment, even if it is required by regulations or workplace norms. Workers may try to avoid the system or refuse to wear the needed gear, making it challenging to apply safety wear detection systems.
    3. Cost: Several safety wear detection systems are still very pricey, keeping them out of reach for smaller enterprises or those on a tight budget. Even when safety wear detection technologies are available, this might be a considerable hurdle to their use.
- Proposed Solution: Here are some possible solutions to the gaps in safety wear detection:
    1. Improved sensors: Improving detection accuracy requires the development of more sensitive and accurate sensors for safety wear detection. These sensors should be able to identify many types of safety equipment, such as gloves, helmets, and protective clothes. Machine learning methods can also be utilised to increase sensor data analysis accuracy.
    2. Safety wear detection systems should be connected with other workplace safety systems, such as environmental sensors and safety monitoring software. This

integration will provide a more comprehensive picture of workplace safety conditions and allow for proactive safety measures.

3. Solutions that are inexpensive: Creating cost-effective safety wear detection solutions is critical to ensuring that they can be implemented by enterprises of all sizes. One approach is to employ low-cost sensors and wearables that are simple to deploy and maintain.
4. Worker engagement and training: Increasing worker acceptability of safety wear detection systems by involving workers in the process and providing them with training on the importance of safety gear. Employees should be involved in the selection of safety equipment and trained on how to properly wear it.

- Feasibility Study: To establish the viability and potential success of adopting safety wear detection in the workplace, a feasibility study is required.The research should look into how safety wear detection will be integrated into existing safety measures and whether it would affect existing workflows. The study should also investigate how to manage and sustain safety wear detection over time.
- System development: Many stages are involved in the creation of a safety wear detection system, including:
    1. Gathering datasets: The first phase is to collect datatsets. The criteria should include the types of safety equipment that is helmet,gloves,masks,vests and boots that must be identified, the detection accuracy and sensitivity, and any connection with current safety systems.
    2. Design: After gathering the requirements, the system can be designed. Selecting the relevant sensors and wearables, implementing software algorithms for data analysis, and designing the user interface are all part of this process.
    3. Development of a prototype: A prototype of the safety wear detection system can then be created. This includes testing the chosen sensors and wearables as well as designing software for data processing. The prototype should be tested to confirm that it meets the specifications and is accurate and dependable.
    4. Testing and validation: The safety wear detection system will be tested and validated in the following stage. This entails putting the system through its paces in various industrial locations and situations to guarantee that it can detect safety equipment accurately and reliably. To guarantee that the system delivers a comprehensive view of workplace safety, it should be verified against existing safety systems.
- Solution and validation: The answer for field safety wear detection involves putting in place a system that can detect whether workers are wearing the proper safety gear in the job. This system may employ sensors and wearables to identify the presence of safety equipment such as hard hats, safety glasses, and high-visibility clothes. These sensors' data can then be evaluated using software algorithms to determine whether workers are wearing the proper safety equipment.

3.2 Methodology

3.2.1 Brief of the Proposed Methods

The datasets include various images of safety wear that a worker may require on field with marking for the specific parts of the images that we are interested in detecting. A testing set of images has also been included to measure the accuracy of the various images that were used in training the model.

To detect the safety equipment of workers in various fields, we have proposed a IOT based architecture integrated with computer vision methods to minimize cost and increase effectiveness.

Here the deep learning model we have chosen to implement is YOLOv4 to perform object detection using keras backend. Keras is a high-level neural network API written in Python which enables the seamless implementation of algorithms using the Tensorflow framework. YOLO is a cutting-edge algorithm that utilizes the principles of CNN, the You only look once (YOLO) algorithm is an object detection system targeted for real-time processing. YOLO divides the input image into an S ×S grid. Each grid cell predicts only one object and a fixed number of boundary boxes. For each grid cell, it predicts B boundary boxes, and each box has one box confidence score. Then, it detects one single object, regardless of the number of boxes B. Finally, it predicts C conditional class probabilities (one per class for the likeliness of the object class). Each boundary box contains 5 elements: (x, y, w, h) and a box confidence score. The confidence score reflects how likely the box is to contain an object.

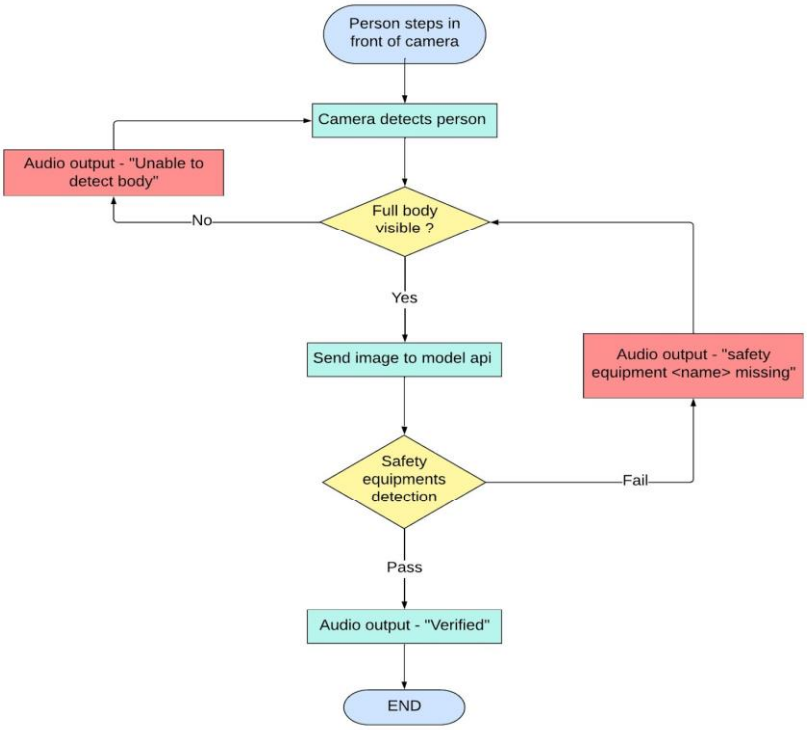


Fig 3.1: Proposed Detection System Diagram

As soon as a person enters the camera frame, our camera module will try to detect the person using the mediapipe api pipeline for human pose detection. Once all the body points are in frame, one snapshot of person will be taken and sent to the YOLOv4 model api endpoint which will analyse it and return whether any of the safety equipment is missing. Once all safety equipment is verified, the person is verified, and it will be indicated via a speaker node. In case of any failures in calibration of person or safety equipment detection, the speaker node will notify the user.

Modules Used:

I. Image Capturing Module: It is used for capturing images of employees from construction and sending it to the computing device for processing and object detection.
II. Labelling Images: In object detection first, we need to train the model using some labelled images and so this is an important module. Images of workers will be labelled. More the number of images in the training dataset, the greater the accuracy.
III. Object Detection: This module is responsible for object detection from the images, such as workers, helmets, masks, vests, boots, gloves.
IV. Searching: A string will be returned when an image is given an input to find out what equipment is missing, we will search the string and add them to lists. The main functionality of this module is to find out missing equipment using string search algorithm.
V. Text to Speech: The functionality of this module is to create a string variable that holds all missing objects and using libraries convert them into mp3 files for voice message.
VI. Alert generation: Functionality of this module is to alert the worker that he does not have the required safety equipment and a voice message will be used for alerting.

Model Deployment: Finally, the trained models will be deployed to the system.

3.2.2 Object Detection and Recognition Process

Object detection is a process that involves finding and differentiating a variety of things that are included inside an image or scene. In the realm of computer vision, object detection algorithms aid computers in locating certain things inside still pictures or moving movies, which enables the computers to carry out a variety of activities on their own. Nevertheless, the employment of image processing-based approaches, which have been historically used, should be avoided owing to the longer processing time, more complicated methodologies, and lower levels of accuracy that may be attained using these techniques. However, object detection algorithms that are based on Deep Learning have recently come to be widely regarded as the most effective object detection algorithms. This is due to the fact that these algorithms are able to learn image and object features on their own, in addition to the superior power these algorithms possess in accurately identifying the objects.

Image classification, image classification along with localization, and object detection are the three primary methods that are used in computer vision-based object identification tasks. When we talk about image classification, we're referring to the process of analyzing a picture and placing it into one of a certain set of classes from among a long number of possible categories. It is the most fundamental application of computer vision. This method predicts the general category of the picture, but it does not single out the individual things that are included inside it. Because of this, a new category has been established, and it is called categorization with localization. In this category, the identification and localization of an item is accomplished using numerous annotations. Nevertheless, this also has a restriction, the most significant of which is that it can only identify a single category of things. Object detection is the category that must be addressed in order to complete labelling and categorization of various items.

Deep learning has shown to deliver the highest potential level of performance and outcomes in computer vision applications, which are increasingly concentrating on the object detection category. In this area, object identification may be accomplished using a variety of methods; however, region based CNNs are generally regarded as the most effective approach.

The You Only Look Once (YOLO) algorithm suggests employing an end-to-end neural network that can forecast bounding boxes and class probabilities simultaneously. It takes a different strategy from the one that was used by earlier object identification algorithms, which reused classifiers to carry out the detection function.

YOLO has reached results that are considered to be state-of-the-art, outperforming existing real-time object identification algorithms by a significant margin. This was accomplished by taking a fundamentally unique approach to object recognition.

YOLO is able to make all of its predictions by utilising a single layer that is fully connected, in contrast to other algorithms such as Faster RCNN, which work by first identifying possible regions of interest through the use of the Region Proposal Network and then performing recognition on each of those regions individually.

Techniques that make use of Region Proposal Networks go through numerous iterations for each picture, but YOLO just has to go through one iteration to get the job done.

3.2.3 Development of YOLO

1. The YOLO (You Only Look Once) system detects objects in real time. It detects objects in pictures and video frames using a single neural network.
2. One of YOLO's most important applications is in the field of safety wear detection, where it is used to detect whether workers on construction sites or in industrial settings are wearing appropriate safety gear such as helmets, gloves and vests.

3. YOLO is particularly well-suited for safety wear detection because it can detect many things in a single image and reliably pinpoint the location of each object. As a result, it is far faster and more precise than standard object detection systems.
4. Large datasets of photos and videos are required to train YOLO for safety ware detection. These databases must include a variety of photos of workers in various environments and wearing various forms of safety equipment.
5. Once trained on a dataset, YOLO can detect safety equipment in real-time video feeds from cameras on construction sites or in industrial environments. This can help to improve worker safety by ensuring that workers are always wearing suitable safety equipment.
6. YOLO is also employed in various worker safety applications, such as detecting hazardous compounds and identifying possible safety concerns on construction sites.
7. As computer vision technology advances, YOLO and other object detection systems are anticipated to become more precise and efficient. This has the potential to significantly improve worker safety and other aspects of industrial and occupational health and safety.

3.3 YOLO and Field Worker Safety Detection System

The safety monitoring system we'll go over is made up of seven modules that work together to identify employees who don't have the necessary safety equipment and notify them to the need to wear it. Image Capturing, Image Labeling, Object Detection, Searching, Text to Speech, Alert Generation, and Model Deployment are among the components. The Image Capturing Module is in charge of photographing building employees. High-quality images that can be used to recognise objects on the site are essential. Cameras strategically placed around the construction site are usually used to apply the module, and these cameras are linked to a computing device that processes the images.

Image Labeling is a critical component of the safety tracking system. This module labels the images captured by the Image Capturing module, which is required to teach the model to recognise objects in the images. The amount of labelled images used for training determines the model's accuracy. Object Detection is the module in charge of recognising things in images. The safety equipment worn by employees, such as helmets, vests, gloves, boots, and masks, is the focus of this case. Object detection algorithms are used to identify these objects in the images and determine whether or not the employees are wearing them.

The Searching module is in charge of searching the Object Detection module's data for missing safety equipment. A string is given when an image is entered into the system. The Searching module searches the string for absent safety equipment and adds the results to the relevant lists. The Text to Speech module creates a string variable that contains all of the

missing safety equipment, and then converts the text into mp3 files using libraries. The mp3 files are used to deliver voice messages to employees who are lacking safety equipment.

The Generation of Alert module is in charge of informing employees who are lacking safety equipment. The module plays a voice message alerting the worker to the missing safety equipment using the mp3 files produced by the Text to Speech module. The final module is the Model Deployment module, which is in charge of putting the learned models into execution on the system. This guarantees that the system can work in real-time and continuously monitor the construction site.

Finally, the safety monitoring system we discussed is an important tool in the building industry. It can help to ensure that workers wear the required safety equipment by utilising cutting-edge technology, lowering the risk of accidents and injuries. The accuracy of the modules, particularly the Image Labeling, Object Detection, and Searching modules, determines the system's efficacy. To provide a reliable safety monitoring system for building sites, the modules must function in unison.

3.3.1 Technique Used

In order to produce an efficient device for construction site safety detection, our team will be working on both the hardware and software sides of the problem.

The following types of hardware components will be used in the completion of our project:

1. Camera: A tool for taking pictures of employees for the purpose of ensuring their safety.
2. A piece of computer equipment equipped with a GPU for use in the processing of images and the deployment of models
3. A speaker and screen for showing the safety equipment that has been misplaced.

The following is a list of software components that are going to be used:

1. Frameworks such as OpenCV and TensorFlow will be used as library resources.
2. Real-time object identification is accomplished with the help of the YOLO model
3. Websites that are open source and utilised for picture classification, such as makesense.ai
4. A text-to-speech library called gTTS that may announce the absence of necessary safety devices.

Device setup:

1. The cameras have to be positioned at a height of 1.71 metres, which will guarantee that the picture of persons of varying heights will be accurate.
2. Pictures should only be taken if it is possible to see all of the points that make up the skeleton model.

3. The photographs of the capture should be provided to the model that has been deployed.
4. The findings will be computed, and if the confidence in the output for an item is more than sixty percent, then it indicates that the object in question is there.
5. The speaker will sound an alarm if any piece of equipment is missing, for example "Helmet Missing."

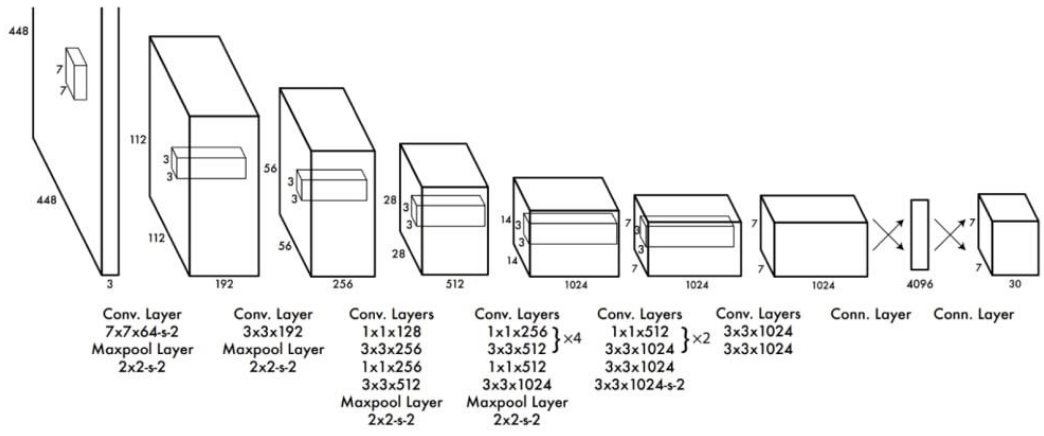
Components:

1. Picture labelling technologies like makesense.ai
2. Object identification, segmentation, and classification may be accomplished via the use of image processing models such as YOLO.
3. Deep Learning models, such as CNN, for locating and categorising objects in digital photographs.
4. The GTTS library is a Google text-to-speech conversion tool that will be used to transform text to speech.

Tensor Board is going to be used in order to carry out model assessment and show accuracy score.

3.3.2 Architecture of the YOLO project

In order to identify objects in a picture, the YOLO method employs a basic deep convolutional neural network. This is a diagram depicting the structure of the convolutional neural network (CNN) model that underpins YOLO.



The Architecture. Our detection network has 24 convolutional layers followed by 2 fully connected layers. Alternating 1×1 convolutional layers reduce the features space from preceding layers. We pretrain the convolutional layers on the ImageNet classification task at half the resolution (224×224 input image) and then double the resolution for detection.

ImageNet is used to pre-train the model's first 20 convolution layers, which are then implemented using a temporary average pooling and fully linked layer. As it has been shown that enhancing a pre-trained network with convolution and linked layers enhances its performance, this model is then repurposed for detection. The class probabilities and bounding box locations are predicted by the last fully connected layer in YOLO.

YOLO creates a square grid out of the supplied picture. The detection of an item is delegated to the grid cell containing its centre. Predicted B bounding boxes and associated confidence ratings are shown in each grid cell. These scores represent the degree to which the model agrees with the prediction that an item is within the box.

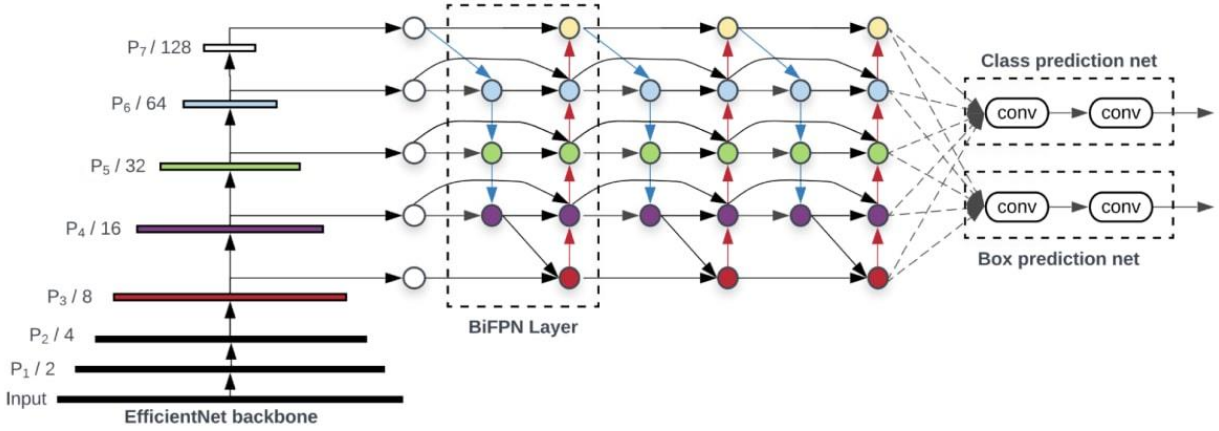
Each grid cell in YOLO is predicted to have numerous bounding boxes. One bounding box predictor per item is preferred during training. After determining which prediction has the greatest current IOU with the ground truth, YOLO designates a single predictor as "responsible" for forecasting an item. As a result, the various bounding box predictors end up becoming more specialized. The predictors improve their memory as a whole by being more accurate at predicting specific item sizes, shapes, and categories.

Non-maximal suppression is a crucial method in YOLO models (NMS). NMS is a post-processing procedure used to enhance object detection precision and performance. Several bounding boxes may be created for the same item in object detection. It's the same thing, even if its bounding boxes are in various places or overlap. In order to generate a single bounding box per object in an image, NMS is used to detect and discard duplicate or inaccurate bounding boxes.

3.3.3 YOLO V5 Review

In 2020, the same group that created the original YOLO algorithm released YOLO v5, which is now maintained as an open-source project by Ultralytics. The latest edition of YOLO has various new features and enhancements on top of the ones seen in earlier versions.

In contrast to YOLO, YOLO v5 employs the EfficientDet architecture (shown below), which is based on the EfficientNet network design. YOLO v5's improved accuracy and generalizability to additional object classes are the result of a more sophisticated design.



EfficientDet architecture – It employs EfficientNet [39] as the backbone network, BiFPN as the feature network, and shared class/box prediction network. Both BiFPN layers and class/box net layers are repeated multiple times based on different resource constraints as shown in Table 1.

The object identification model's training data also differs between YOLO and YOLO v5. The PASCAL VOC dataset, which was used to teach YOLO, has 20 different types of objects. Nevertheless, YOLO v5 was trained on D5, a far bigger and more varied dataset that has 600 item types in total.

In YOLO v5, a new technique known as "dynamic anchor boxes" is used to generate the anchor boxes. Clustering the ground truth bounding boxes and then employing the cluster centres as anchor boxes is the method used here. This enables the anchor boxes to better fit the dimensions and contours of the items that have been identified.

Spatial pyramid pooling (SPP) is a new pooling layer in YOLO v5 that may be used to decrease the feature maps' spatial resolution. As SPP gives the model a broader perspective, it is able to better recognise even the smallest of things. While YOLO v4 also employs SPP, the SPP architecture in YOLO v5 has been enhanced in a number of ways that allow for higher performance.

While training the model, YOLO v4 and v5 use a loss function that is conceptually comparable. Nevertheless, "CIoU loss," a new word introduced in YOLO v5, is a version of the IoU loss function that is meant to enhance the model's performance on unbalanced datasets.

3.3.4 Integration

Here, we will talk about how to combine YOLOv5 with worker safety monitoring on the construction site. Specifically, we will focus on locating protective apparel and equipment, including helmets, masks, vests, shoes, and gloves. In addition, we will discuss how we modified the label datasets and created the data.yaml file used for training the model.

In order to train a YOLOv5 model to detect workers wearing safety gear on construction sites, we require a labelled dataset of images containing the safety gear objects that we wish to detect. There are numerous publicly accessible datasets that can be used for object recognition. Open Images, COCO, and Pascal VOC are a few examples. Nevertheless, it is conceivable that these datasets do not contain any relevant images for our use case.

In order to develop our dataset, we collected images of individuals working on construction sites from a variety of sources, including websites that sell stock photos and public domain image archives. Then, we labelled the images using a tool called Makesense.ai, which allows us to draw bounding boxes around objects of interest and then label those bounding boxes with the class names to which they relate.

The following step was to create the data.yaml file, which contains information about the dataset. This file contains the number of classes, the names of the classes, and the locations of the image and label files.

Some of dataset were imported from roboflow and so we modified their labels according to our data.yaml file label and then moved these files in one folder.

Now that the dataset and data.yaml file have been generated, a YOLOv5 model for identifying hazardous working conditions on construction sites can be trained. PyTorch, a well-known deep learning infrastructure, is utilized throughout the training of the YOLOv5 model. The steps involved in training the model are as follows:

- Configure the YOLOv5 repository located on GitHub.
- After downloading the pre-trained weights from the YOLOv5 repository, you can place them in a subfolder.
- Make modifications to the repository's yaml file so that it corresponds to our data.yaml file.
- Initiate the training process.
- The model weights will be stored in the runs/train/construction_site_safety/weights/best.pt directory once the training is complete. This directory will be automatically created. These weights can be applied to new images so that conclusions can be drawn about them.

Here, we discussed the procedure for integrating YOLOv5 to detect worker safety on construction sites. We reviewed the procedures for preparing the dataset, configuring YOLOv5, training the model, and drawing conclusions from new images.

It is crucial to remember that the success of the model depends not only on the quality and quantity of the training data, but also on the hyperparameter settings. Due to the fact that the model was created for specific object recognition tasks, it is conceivable that it will perform poorly in certain situations.

In conclusion, YOLOv5 is an extremely powerful and adaptable instrument for object detection, and the integration of this tool into construction site safety measures can improve those measures. With additional research and development, it may be possible to enhance workplace safety in a variety of different industries.

3.4 Mobile Application

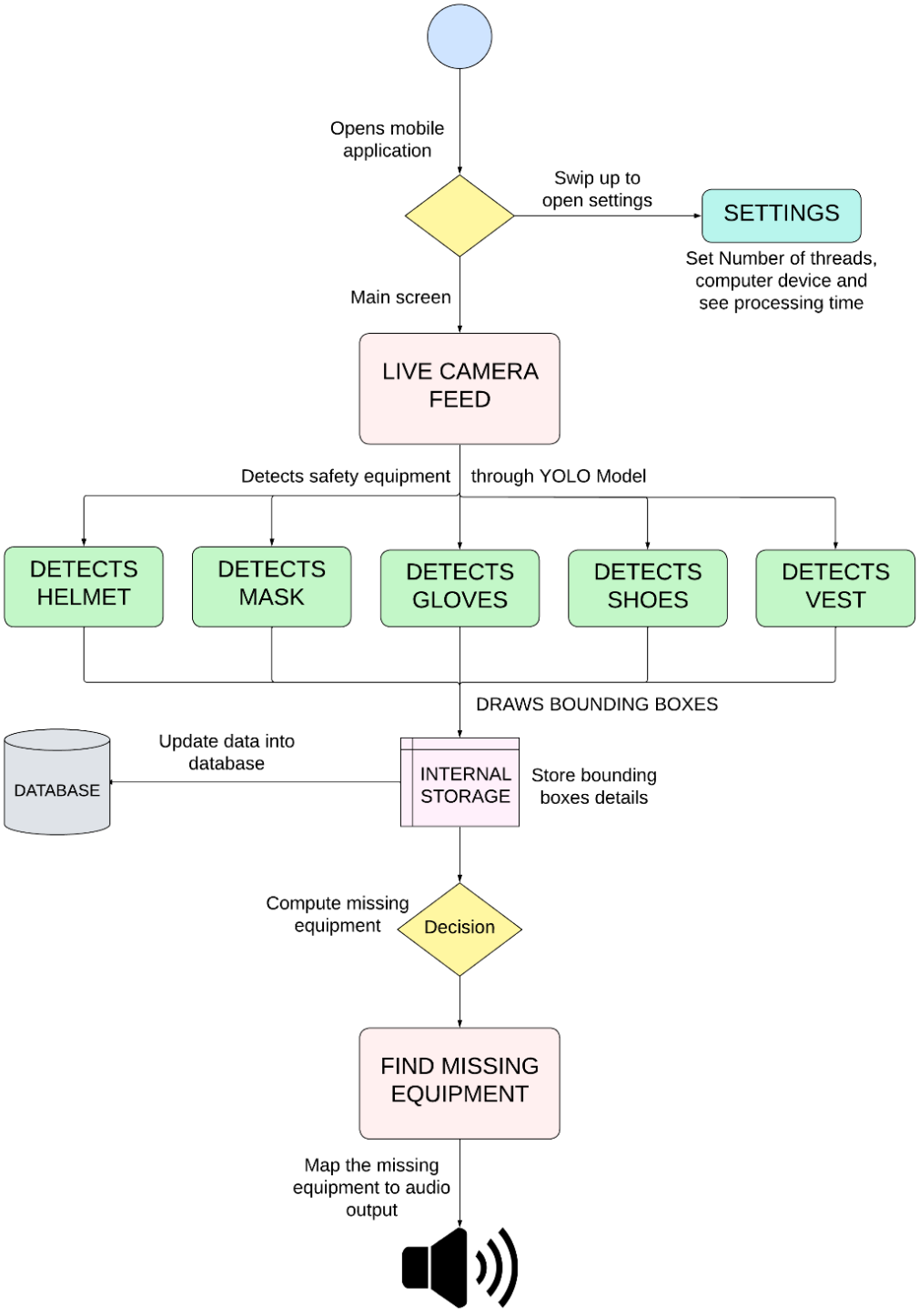
3.4.1 Overview of the Application

The Java mobile application for safety equipment detection is based on the You Only Look Once (YOLOv5) object detection algorithm, which is a deep learning-based algorithm used for real-time object detection tasks. The YOLOv5 algorithm is known for its high accuracy and speed, making it an ideal choice for mobile applications.

The mobile application uses the device's camera to capture a live video feed, which is then processed frame by frame using the YOLOv5 model. The model analyzes each frame and generates a set of bounding boxes around the objects detected in the image. Each bounding box is associated with a confidence score that represents the likelihood of the object being present in the image.

The app then filters out the objects with low confidence scores and identifies the objects that correspond to safety equipment. If the model detects that any safety equipment is missing, the app displays a notification message indicating which equipment is required to ensure workplace safety.

The YOLOv5 model used in the app is optimized for mobile devices, making it highly efficient in real-time object detection tasks. The app utilizes advanced computer vision techniques to enhance the accuracy of the model's detection results, including anchor boxes, which are used to improve the localization accuracy of objects in the image.



3.4.2 Mobile Application and Detection System Integration

The java android application is integrated with safety equipment detection system using YOLOv5 which is running on Python, and SQL database to store data. The development of this app can be broken down into many phases.

Training the Model: The YOLOv5 model needs to be trained on a large dataset of safety equipment images. This involves using a labeled dataset and training the model using deep learning frameworks such as PyTorch or TensorFlow. The training process typically involves several epochs, where the model is trained on batches of images and the weights are updated to minimize the loss.

Model Export and Integration: Once the model is trained, it needs to be exported and integrated into the Java Android application. The model can be exported in several formats, such as ONNX, TorchScript, or TensorFlow Lite. The exported model can then be imported into the application's codebase and used for object detection.

Object Detection Algorithm: The Java Android application needs to include an object detection algorithm that processes the video feed from the device's camera and returns the bounding boxes around the detected objects. The YOLOv5 model can be used as the backbone of the object detection algorithm, which can be implemented using frameworks such as OpenCV or TensorFlow Lite.

Real-Time Object Detection: The Java Android application needs to be optimized for real-time object detection tasks. This involves using advanced computer vision techniques such as non-maximum suppression (NMS) and anchor boxes to improve the accuracy and efficiency of the model.

User Interface: The Java Android application needs to have a user interface that displays the video feed from the device's camera, the detection results, and notification messages. The user interface needs to be designed to be intuitive and easy to use, with appropriate feedback mechanisms to guide the user.

Testing: The Java Android application needs to be tested thoroughly to ensure that it functions as expected. This involves testing the application in different environments, such as different mobile devices and platforms, and testing its performance in detecting safety equipment in real-time.

Some Java classes which are used for integrating java activities with XML files to update UI components when helmet, vest, shoe, gloves and mask are detected and draw bounding boxes.

AppCompatActivity: This is a base class for activities in Android that provides support for action bar, Toolbar, and other UI features. You can extend this class to create your own activity classes for displaying the safety equipment detection results.

ImageView: This is a UI component that displays images in Android. You can add an ImageView component to your layout XML file to display the camera preview or the image with the detected objects.

CheckBox: This is a UI component that allows the user to select one or more options from a list of choices. You can add CheckBox components to your layout XML file for each type of safety equipment you want to detect.

FrameLayout: This is a UI component that allows you to overlay multiple views on top of each other. You can add a FrameLayout component to your layout XML file to display the bounding boxes around the detected objects.

Paint: This is a class in Android that allows you to draw graphics and text on a canvas. You can use this class to draw bounding boxes around the detected objects in the FrameLayout.

Canvas: This is a class in Android that provides a drawing surface for drawing graphics and text. You can use this class in conjunction with the Paint class to draw bounding boxes on the FrameLayout.

SurfaceView: This is a UI component in Android that provides a dedicated drawing surface for a view hierarchy. You can use this class to display the camera preview and draw bounding boxes around the detected objects.

SurfaceHolder: This is a class in Android that provides access to the underlying surface of a SurfaceView. You can use this class to lock and unlock the canvas for drawing graphics and text.

ObjectDetector: This is a Java class that encapsulates the object detection algorithm using the YOLOv5 model or another model of your choice. This class can be used to detect safety equipment in real-time and update the visibility of the CheckBox components and draw bounding boxes around the detected objects.

3.5 System Development Process

3.5.1 Overview

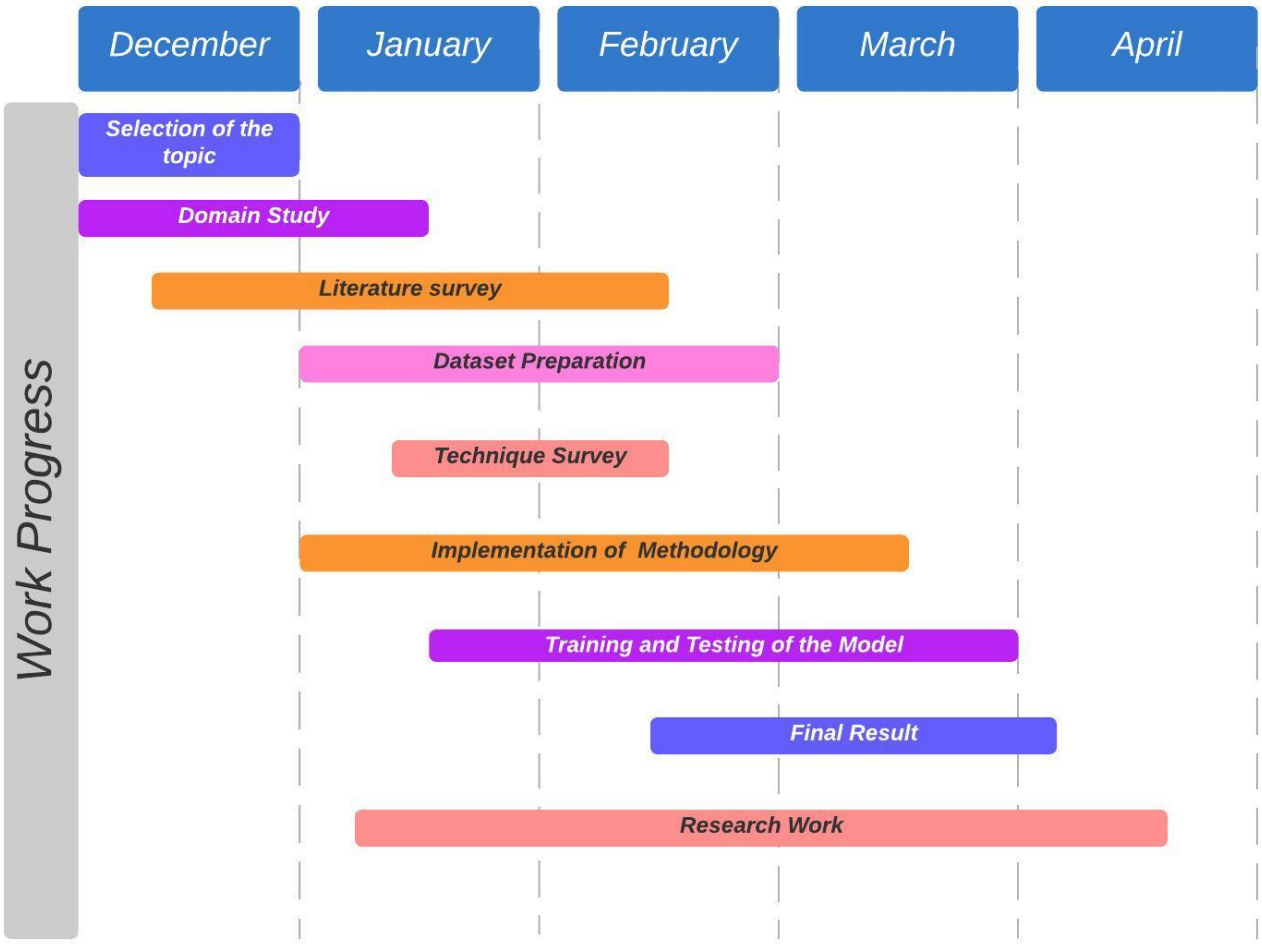
Protective gloves, vests, shoes, and masks are just some of the accessories that may be included into a field worker safety detection system

Protecting employees' hands from tools and other sharp items is a common safety concern in the workplace. Cut-resistant gloves are one solution. Workers may increase their visibility in poor light by wearing luminous vests. Steel-toed shoes protect employees from being crushed by falling items. Workers may avoid inhaling dangerous chemicals or dust by using protective masks.

These pieces of PPE may have sensors or other technology added to them to further ensure the safety of the workers who wear them. For instance, sensors built into a smart glove may

measure fluctuations in heat, pressure, or vibration to alert the wearer of impending danger. A smart vest might similarly be equipped with GPS or motion sensors to monitor the wearer's whereabouts and actions on the job.

Overall, a field worker safety detection system may be improved by combining PPE with sensors and other technologies. When used in conjunction with modern sensors and alarm systems, personal protective equipment (PPE) may offer an additional layer of safety for employees in potentially dangerous workplaces.



Chapter-4. Experimental Setup, Tests and Results

4.1 Experimental Setup Technical Specification

Table 4.1: Specifications of Software

Sr. No	Software	Version	Use Case
1.	Python	3.8	A robust programming language that facilitates development with multiple machine learning and deep learning library options.
2.	Java	8.281	A versatile, powerful and structured programming language which is used here to develop mobile application.
2.	Colab	1.0.0	Google Colab is a cloud-based utility for writing and executing code in a Jupyter Notebook environment.
3	Jupyter	5.3	To create and share computational documents online, the Jupyter Notebook was the first web tool of its kind.
4.	Makesense.ai	1.0.0	Free image annotation toola
5.	Roboflow	1.0.0	A platform which provides dataset of labeled images
6.	Android Studio	2022.1.1	IDE for android application development using java.

Table 4.2: Specification of Libraries

Sr. No	Software	Version	Use Case
1.	Tensorflow	2.7	A deep learning library
2.	Pytorch	1.13	A machine learning framework based on python and torch
3.	Roboflow	1.0.0	A library in python to import annotated datasets in colab
4.	Opencv-Python	4.7.0	offers a standard platform for application development in the field of computer vision.

Table 4.3: System Specification

Sr. No	Software	Version	Use Case
1.	ColabVM	1.0.0.1	Operating System.
2.	Intel Xeon	2.2 GHz	Processor.
3.	12 GB RAM	DDR5	Memory that is used for running the applications.
4.	Nvidia Tesla K80	12 GB VRAM	GPU that is used for running the deep learning methodologies.

4.2 Dataset Preparation

The compilations of data, whether text, images, or videos, that serve a particular purpose. The most important inputs for constructing deep learning algorithms such as convolutional neural networks (CNNs) are data sets. Without them, the networks would be largely ineffective.

CNNs are self-learning networks that modify their neural network weights and biases via cost reduction and backpropagation using dataset images. If the datasets are of poor quality or insufficient size, the learning and training of the network will be calamitous, resulting in useless or even dangerous results due to false or true negative detections.

In addition, datasets are the primary premise for comparing different methodologies. When using images to generate outputs or results that will be utilised elsewhere, datasets play a crucial role in determining which algorithm performs better in terms of speed, efficiency, and precision. In order to ensure the success of the deep learning algorithms being developed, it is crucial to carefully select and curate datasets for any particular project.

Since there is no dataset that contains the required number of images in a single package, gloves, shoes, vest, helmet, and mask images from various categories are collected and used

to create a large data pool for training the convolutional neural network in this work. Due to the use of a variety of image types, the neural network is guaranteed to function in a variety of scenarios, resulting in high robustness and dependability.

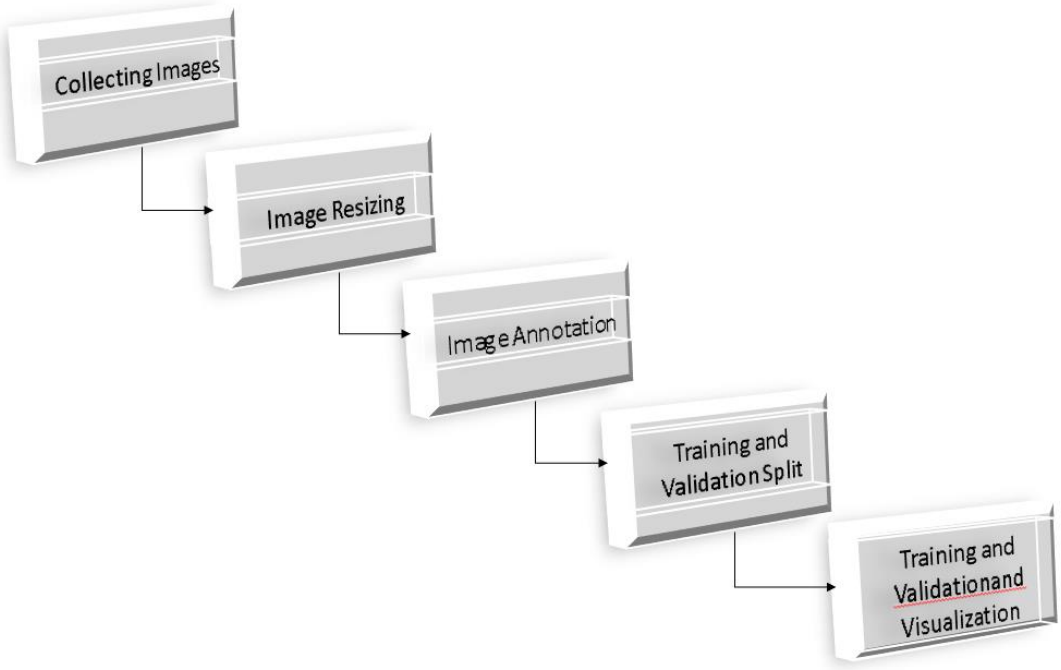


Fig. 4.1 Dataset Flow

In order to cut down on processing time, the preparation of the dataset requires resizing the pictures to a size that is more appropriate for the model. In addition, picture annotation is a part of this package. In order to annotate an image, it is necessary to painstakingly draw bounding boxes and label objects of interest that can be found within each picture. The process of object identification in computer vision relies heavily on annotation; in its absence, the process is rendered ineffective. Data that has been annotated in some way gives a neural network that is undergoing training information about the differences between various objects. After receiving an adequate amount of training, the neural network is able to learn the characteristics of the annotated and labelled objects, which enables it to differentiate and identify the necessary objects contained within a test picture. The workflow of the complete process of preparing the dataset is illustrated in Figure 1.

Since this work includes identifying a helmet, vest, mask, mittens and shoes, the dataset that was used in this study is an aggregation of multiple datasets that were selected to construct the best possible training dataset. The datasets that were utilised in this research were acquired from roboflow, and the other images were labelled using the website makesense.ai. A large number of images from these datasets were compiled into a single dataset, which was then vetted to include only images of the helmet, mittens, vest, mask, and shoes. Some examples of

these images are presented in Figure 4.2. Out of this massive accumulation of images, a total of 11793 were chosen by hand and annotated so that they could be used for training, validation, and testing purposes.

Fig. 4.2 Sample Images in dataset

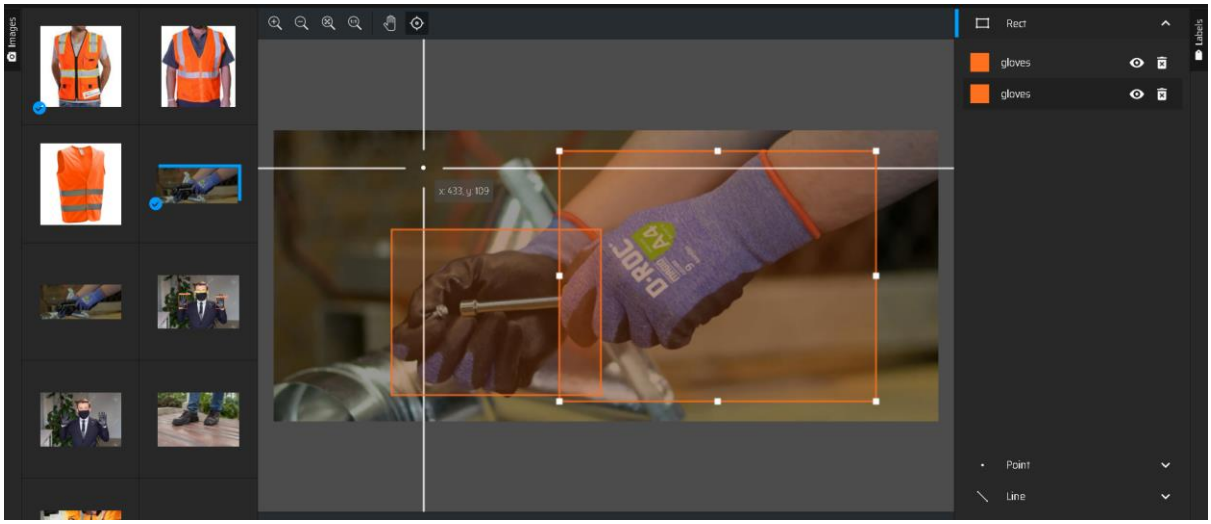
Images were annotated using Makesense.ai. MakeSense.ai is a business that specialises in computer vision and offers a diverse selection of annotation and labelling services for use in machine learning programmes. Image annotation, video annotation, text annotation, and voice annotation are just some of the services that they provide.

MakeSense.ai employs a wide variety of tools and methods for the image annotation process, which enables it to accurately label objects contained within a picture. One such programme is called LabelImg, and it is utilised quite frequently for various object detection duties. The users of LabelImg are able to open each image that is included in the training set and identify objects of interest within the images by drawing bounding boxes around the objects. Customers of the business are given the ability to train and validate machine learning models, thereby enhancing their ability to recognise and categorise objects with a high degree of precision.

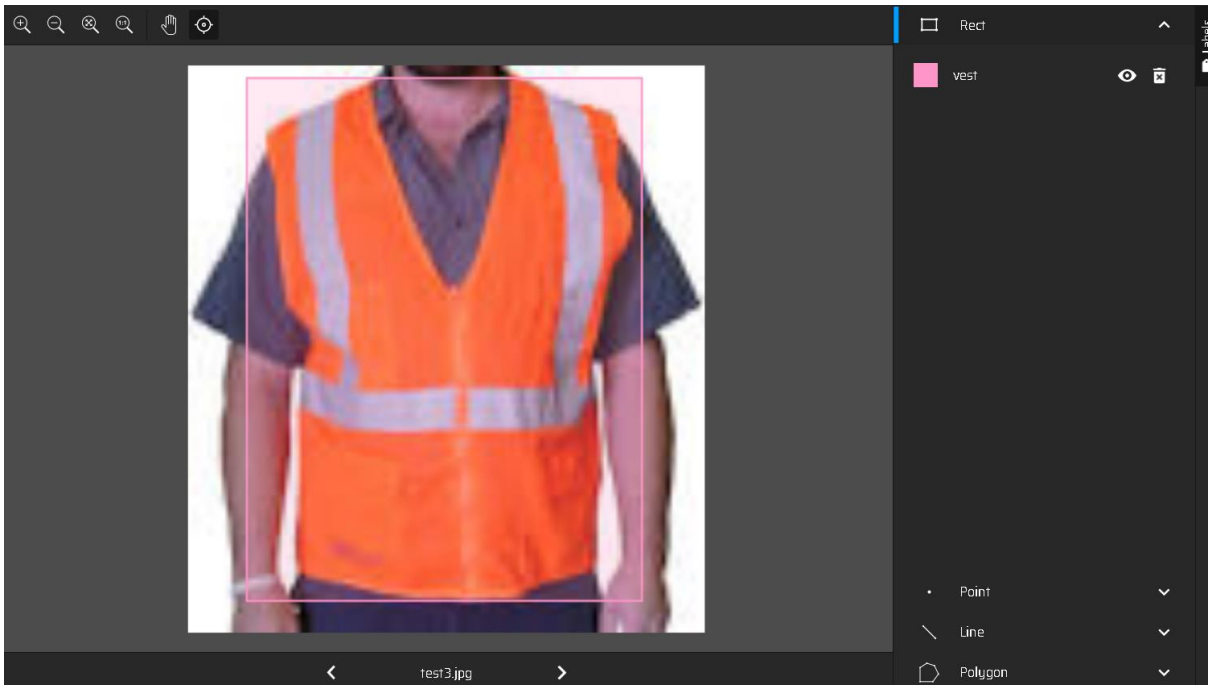
MakeSense.ai utilises a binary file storage structure known as TF Record, which is also utilised by Tensorflow, in order to optimise the input data for training and validation of the machine learning models. This is done by Tensorflow. This format offers improved functionality by reducing the amount of storage space required, as well as the amount of time needed to read

and copy data. TF Records has the ability to combine several datasets into a single repository of input data, which makes it much simpler to process massive datasets. This strategy is especially beneficial when it comes to the processing of massive quantities of data, which is typically the case in applications that involve machine learning.

MakeSense.ai is able to provide their customers with labelling and annotation services of a high calibre as a whole as a result of their utilisation of tools such as LabelImg and TF



(a)



(b)

Figure 4.3: Dataset Annotation Process

4.3 Results of Experimental Tests

Experiments utilised all colab resources, including jupyter notebook, an Intel core processor, 12 GB of virtual memory, and a Google GPU, GPU. In addition, Python, Tensorflow, and Jupyter Notebook were configured in an environment in order to create a testing environment for implementing the algorithms.

The model was developed using Python and the CNN-based Yolov5 algorithm. The annotated dataset was divided into three for the objectives of experimentation. The first part, consisting of 9700 images, is the training dataset, the second part, consisting of 1000 images, is the validation dataset, and the third part, consisting of 400 images, is the test dataset. Image enhancement techniques such as rotation, flip, colour saturation, hue, and contrast are programmed to be implemented during the training procedure. The training procedure was initiated after the batch size was specified to 64. Thus, the training procedure was conducted for a total of 100 epochs. The entire procedure required approximately seven hours to complete. The results of training, validation, and testing demonstrate that the proposed methodology is highly precise and effective. The network achieves a 98% detection rate on the training dataset, 93% detection rate on the validation dataset, and 91% detection rate on the test dataset. Table 4.4 provides an overview of the results.

Table 4.4: Results of the Proposed System

Dataset	Total Number of Images	Accuracy Rate
Train	9700	98 %
Validation	1000	93%
Test	400	91 %

Following training, an Android application was developed. Using Java, the application was developed. Tensorflowlite was derived from the trained model.(**About application)**

Fig. 4.4, Fig. 4.5, Fig 4.6 and Fig. 4.7 shows YOLOv5 detection results, Fig. 4.8, Fig. 4.9 and Fig 4.10 represents the evaluation metrics obtained after training the model. The Application functions are displayed in Fig. 4.11 and Fig. 4.12.

Fig. 4.4 Object detection by the model

Fig. 4.5

Fig 4.6

Fig 4.7

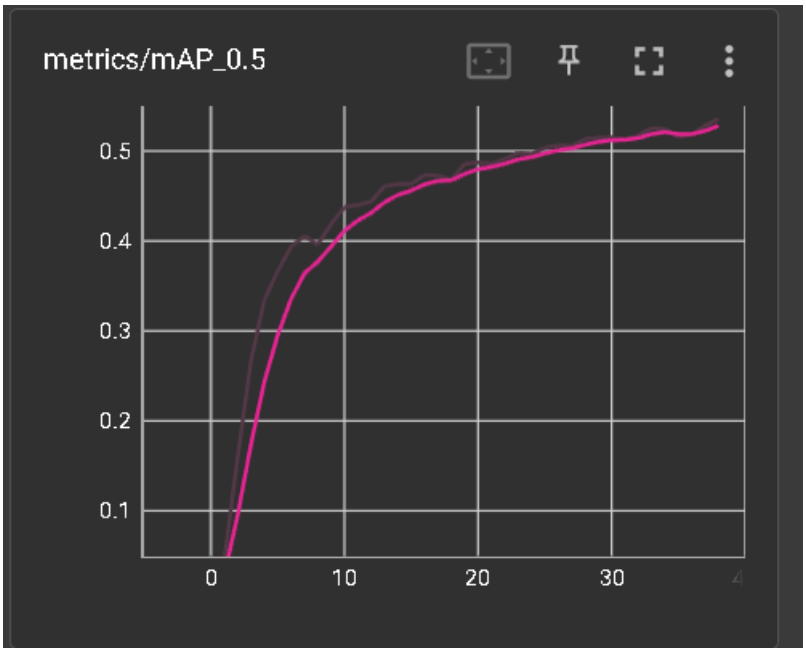


Fig 4.8 Evaluation metrics

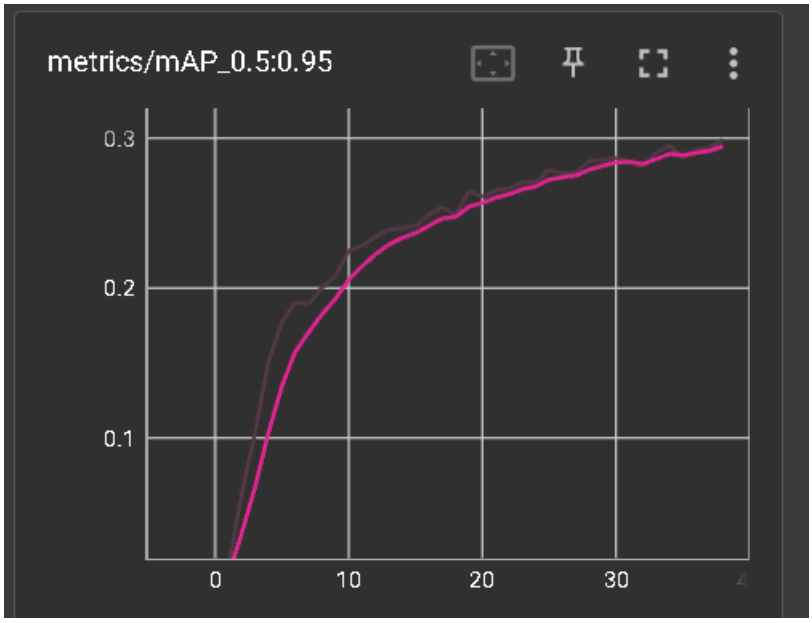


Fig. 4.9 Map 95 evaluation

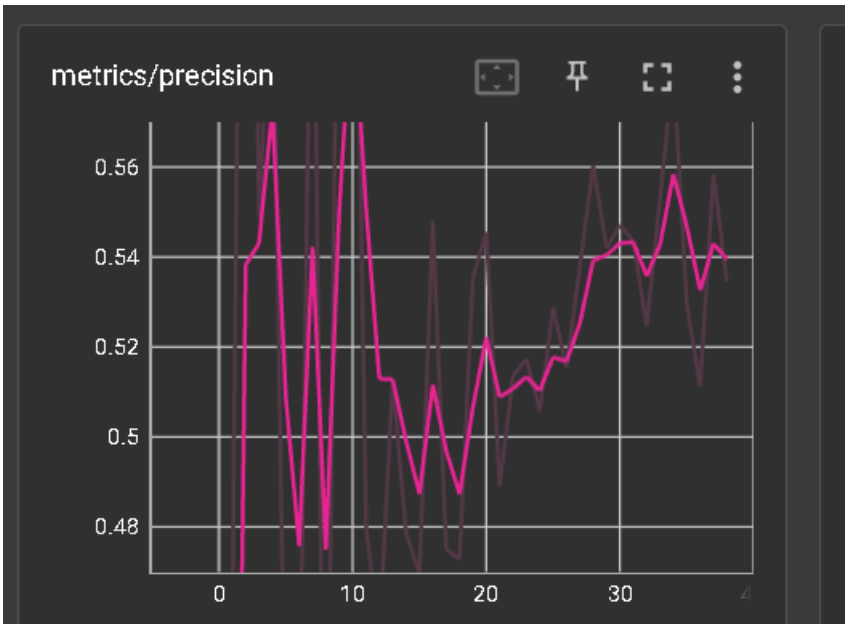


Fig. 4.10 Precision

Chapter-5. Conclusion and Future Scope

5.1 Conclusion

One other distinction that can be made between YOLO and YOLO v5 is with regard to the training data that is utilized to learn the object detection model. YOLO received its training using the PASCAL VOC dataset, which has 20 different object categories. YOLO v5, on the other hand, was trained using a dataset called D5 that is far bigger and more varied, and it has a total of 600 item types.

YOLO version 5 implements a whole new approach for producing the anchor boxes, which is referred to as "dynamic anchor boxes." It includes employing a clustering technique to group the ground truth bounding boxes into clusters and then utilizing the centroids of the clusters as the anchor boxes. This may be accomplished by clicking here. This makes it possible for the anchor boxes to have a size and form that is more closely matched with the identified items.

In addition, YOLO version 5 incorporates the "spatial pyramid pooling" (SPP) idea. SPP is a form of pooling layer that is used to lower the spatial resolution of the feature maps. Since it enables the model to see the objects at a variety of sizes, SPP is used in order to enhance the detection performance on tiny items. YOLO v4 makes use of SPP as well, but YOLO v5 incorporates significant enhancements to the SPP design, which enables it to do more and provide better results.

Both YOLO v4 and YOLO v5 use a comparable loss function while training the model. Nevertheless, version 5 of YOLO includes a new concept known as "CIoU loss," which is a variation of the IoU loss function that is intended to enhance the model's performance when applied to unbalanced datasets.

5.2 Future Scope

The area of safety wear detection is continuously growing, and a wide range of potential technical advancements that could become accessible in the future might have a big impact on it. The following is a list of some of the potential outcomes:

Improved Skills in the Area of Sensing: The improvement of the sensors that are used in the detection of safety wear is one area that has the possibility for further development. As a result of advancements in technology, sensors are gaining both sensitivity and precision. This might potentially improve the capacity to identify safety wear and provide information that is more particular on the safety gear that is being used.

Artificial Intelligence: Artificial intelligence (AI) is being used in an increasing number of situations to recognize protective gear that persons are wearing. It is possible to teach algorithms that use machine learning to recognize the many different kinds of safety equipment and to analyze sensor data in order to find anomalies that can indicate the

presence of potential dangers. These algorithms can also be used to determine whether or not a sensor is functioning properly.

Accuracy enhancements: While item identification systems such as YOLO already have a very high level of precision, it is almost certain that ongoing improvements in machine learning algorithms will lead to even higher levels of accuracy and precision in recognizing protective equipment. This is because YOLO currently has a very high level of accuracy.

Integration with Other Systems: In order to provide a more comprehensive view of the safety circumstances in the workplace, safety wear detection could be coupled with other safety systems, such as environmental sensors or safety monitoring software. This would allow for the provision of a more comprehensive view of the safety circumstances.

Chapter-6. References

[1] Ding, W., Liu, J., Zhang, W., Zhang, W., & Wang, Q. (2020). An Internet of Things System for Underground Mine Air Quality Pollutant Prediction Based on Azure Machine Learning. Journal of Sensors, 2020, 1-10.

[2] F. Zhou, H. Zhao and Z. Nie, "Safety Helmet Detection Based on YOLOv5," 2021 IEEE International Conference on Power Electronics, Computer Applications (ICPECA), Shenyang, China, 2021, pp. 6-11, doi: 10.1109/ICPECA51329.2021.9362711.

[3] Kang Li and Xiaoguang Zhao and Jiang Bian and Min Tan Automatic Safety Helmet Wearing Detection CoRR abs/1802.00264 2018

[4] A. Das, M. Wasif Ansari and R. Basak, "Covid-19 Face Mask Detection Using TensorFlow, Keras and OpenCV," 2020 IEEE 17th India Council International Conference (INDICON), New Delhi, India, 2020, pp. 1-5, doi: 10.1109/INDICON49873.2020.9342585.

[5] Delhi, Venkata Santosh Kumar, R. Sankarlal, and Albert Thomas. "Detection of personal protective equipment (PPE) compliance on construction site using computer vision based deep learning techniques." Frontiers in Built Environment 6 (2020): 136.

[6] Nath, Nipun D., Amir H. Behzadan, and Stephanie G. Paal. "Deep learning for site safety: Real-time detection of personal protective equipment." Automation in Construction 112 (2020): 103085.

[7] Construction Safety Equipment Detection System Yogesh Kawade, Mayur Kasture, Yash Mallawat, Aniruddh Dubal Student, B.Tech., Construction Engineering and Management, Symbiosis Skills and Professional University, Pune, Maharashtra, India Asst. Prof., Construction Engineering, Symbiosis Skills and Professional University, Pune, Maharashtra, Pune, India

[8] Karlsson, Jonathan, et al. "Visual Detection of Personal Protective Equipment and Safety Gear on Industry Workers." arXiv preprint arXiv:2212.04794 (2022).

[9] Gallo, Gionatan, et al. "A smart system for personal protective equipment detection in industrial environments based on deep learning." 2021 IEEE International Conference on Smart Computing (SMARTCOMP). IEEE, 2021.

[10] Nath, Nipun D., and Amir H. Behzadan. "Deep learning detection of personal protective equipment to maintain safety compliance on construction sites." Construction Research Congress 2020: Computer Applications. Reston, VA: American Society of Civil Engineers, 2020.

[11] Ke, Xiao, Wenyao Chen, and Wenzhong Guo. "100+ FPS detector of personal protective equipment for worker safety: A deep learning approach for green edge computing." Peer-to-peer networking and applications (2022): 1-23.

[12] Torres, Pedro, et al. "A Robust Real-time Component for Personal Protective Equipment Detection in an Industrial Setting." *ICEIS (1)*. 2021.

[13] Saudi, Madihah Mohd, et al. "Image detection model for construction worker safety conditions using faster R-CNN." International Journal of Advanced Computer Science and Applications (2020).

[14] Moohialdin, Ammar, et al. "A Real-Time Computer Vision System for Workers' PPE and Posture Detection in Actual Construction Site Environment." EASEC16: Proceedings of The 16th East Asian-Pacific Conference on Structural Engineering and Construction, 2019. Springer Singapore, 2021.

[15] Li, Yange, et al. "Deep learning-based safety helmet detection in engineering management based on convolutional neural networks." Advances in Civil Engineering 2020 (2020): 1-10.

[16] Ferdous, Md, and Sk Md Masudul Ahsan. "PPE detector: a YOLO-based architecture to detect personal protective equipment (PPE) for construction sites." PeerJ Computer Science 8 (2022): e999.

[17] Protik, Adban Akib, Amzad Hossain Rafi, and Shahnewaz Siddique. "Real-Time personal protective equipment (PPE) detection using YOLOv4 and TensorFlow." 2021 IEEE Region 10 Symposium (TENSYMP). IEEE, 2021.

[18] Phuc, Le Tran Huu, et al. "Applying the Haar-cascade Algorithm for detecting safety equipment in safety management systems for multiple working environments." Electronics 8.10 (2019): 1079.

[19] Qiu, Jianzhong, et al. "A Target Detection and Evaluation Method for Safety Protection Equipment." Journal of Physics: Conference Series. Vol. 1966. No. 1. IOP Publishing, 2021.

[20] Arabi, Saeed, Arya Haghighat, and Anuj Sharma. "A deep learning based solution for construction equipment detection: from development to deployment." arXiv preprint arXiv:1904.09021 (2019).

[21] Marks, Eric Daniel, and Jochen Teizer. "Method for testing proximity detection and alert technology for safe construction equipment operation." *Construction Management and Economics* 31.6 (2013): 636-646.

[22] Fang, Weili, et al. "Automated detection of workers and heavy equipment on construction sites: A convolutional neural network approach." Advanced Engineering Informatics 37 (2018): 139-149.

[23] Wang, Mingzhu, et al. "Predicting safety hazards among construction workers and equipment using computer vision and deep learning techniques." ISARC. Proceedings of the International Symposium on Automation and Robotics in Construction. Vol. 36. IAARC Publications, 2019.

[24] Zhao, Meng, and Masoud Barati. "Substation Safety Awareness Intelligent Model: Fast Personal Protective Equipment Detection using GNN Approach." IEEE Transactions on Industry Applications (2023).

❖ Appendix

➔Main Activity

```
package org.tensorflow.lite.examples.detection;

import androidx.appcompat.app.AppCompatActivity;

import android.app.ActivityManager;
import android.content.Context;
import android.content.Intent;
import android.content.pm.ConfigurationInfo;
import android.graphics.Bitmap;
import android.graphics.Canvas;
import android.graphics.Color;
import android.graphics.Matrix;
import android.graphics.Paint;
import android.graphics.RectF;
import android.os.Bundle;
import android.os.Handler;
import android.widget.Button;
import android.widget.CheckBox;
import android.widget.ImageView;
import android.widget.Toast;

import org.tensorflow.lite.examples.detection.customview.OverlayView;
import org.tensorflow.lite.examples.detection.env.ImageUtils;
import org.tensorflow.lite.examples.detection.env.Logger;
import org.tensorflow.lite.examples.detection.env.Utils;
import org.tensorflow.lite.examples.detection.tflite.Classifier;
import org.tensorflow.lite.examples.detection.tflite.YoloV5Classifier;
import org.tensorflow.lite.examples.detection.tracking.MultiBoxTracker;

import java.io.IOException;
import java.util.LinkedList;
import java.util.List;

public class MainActivity extends AppCompatActivity {

    public static final float MINIMUM_CONFIDENCE_TF_OD_API = 0.3f;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

//        checkbox_meat = findViewById(R.id.checkbox_meat);
        cameraButton = findViewById(R.id.cameraButton);
        detectButton = findViewById(R.id.detectButton);
        imageView = findViewById(R.id.imageView);

        cameraButton.setOnClickListener(v -> startActivity(new Intent(MainActivity.this, DetectorActivity.class)));

        detectButton.setOnClickListener(v -> {
            Handler handler = new Handler();

            new Thread(() -> {
                final List<Classifier.Recognition> results = detector.recognizeImage(cropBitmap);
                handler.post(new Runnable() {
```

```
            @Override
            public void run() {
                handleResult(cropBitmap, results);
            }
          });
      }).start();

    });
    this.sourceBitmap = Utils.getBitmapFromAsset(MainActivity.this, "kite.jpg");

    this.cropBitmap = Utils.processBitmap(sourceBitmap, TF_OD_API_INPUT_SIZE);

    this.imageView.setImageBitmap(cropBitmap);

    initBox();
    ActivityManager activityManager = (ActivityManager) getSystemService(Context.ACTIVITY_SERVICE);
    ConfigurationInfo configurationInfo = activityManager.getDeviceConfigurationInfo();

    System.err.println(Double.parseDouble(configurationInfo.getGlEsVersion()));
    System.err.println(configurationInfo.reqGlEsVersion >= 0x30000);
    System.err.println(String.format("%X", configurationInfo.reqGlEsVersion));
}

private static final Logger LOGGER = new Logger();

public static final int TF_OD_API_INPUT_SIZE = 640;

private static final boolean TF_OD_API_IS_QUANTIZED = false;

private static final String TF_OD_API_MODEL_FILE = "yolov5s.tflite";

private static final String TF_OD_API_LABELS_FILE = "file:///android_asset/coco.txt";

// Minimum detection confidence to track a detection.
private static final boolean MAINTAIN_ASPECT = true;
private Integer sensorOrientation = 90;

private Classifier detector;

private Matrix frameToCropTransform;
private Matrix cropToFrameTransform;
private MultiBoxTracker tracker;
private OverlayView trackingOverlay;

protected int previewWidth = 0;
protected int previewHeight = 0;

private Bitmap sourceBitmap;
private Bitmap cropBitmap;

private CheckBox checkbox_meat;

private Button cameraButton, detectButton;
private ImageView imageView;

private void initBox() {
    previewHeight = TF_OD_API_INPUT_SIZE;
    previewWidth = TF_OD_API_INPUT_SIZE;
    frameToCropTransform =
            ImageUtils.getTransformationMatrix(
                    previewWidth, previewHeight,
                    TF_OD_API_INPUT_SIZE, TF_OD_API_INPUT_SIZE,
                    sensorOrientation, MAINTAIN_ASPECT);
```

```
        cropToFrameTransform = new Matrix();
        frameToCropTransform.invert(cropToFrameTransform);

        tracker = new MultiBoxTracker(this);
        trackingOverlay = findViewById(R.id.tracking_overlay);
        trackingOverlay.addCallback(
                canvas -> tracker.draw(canvas));

        tracker.setFrameConfiguration(TF_OD_API_INPUT_SIZE, TF_OD_API_INPUT_SIZE, sensorOrientation);

        try {
          detector =
                  YoloV5Classifier.create(
                          getAssets(),
                          TF_OD_API_MODEL_FILE,
                          TF_OD_API_LABELS_FILE,
                          TF_OD_API_IS_QUANTIZED,
                          TF_OD_API_INPUT_SIZE);
        } catch (final IOException e) {
          e.printStackTrace();
          LOGGER.e(e, "Exception initializing classifier!");
          Toast toast =
                  Toast.makeText(
                          getApplicationContext(), "Classifier could not be initialized", Toast.LENGTH_SHORT);
          toast.show();
          finish();
        }
      }

      private void handleResult(Bitmap bitmap, List<Classifier.Recognition> results) {
        final Canvas canvas = new Canvas(bitmap);
        final Paint paint = new Paint();
        paint.setColor(Color.RED);
        paint.setStyle(Paint.Style.STROKE);
        paint.setStrokeWidth(2.0f);

        final List<Classifier.Recognition> mappedRecognitions =
                new LinkedList<Classifier.Recognition>();

        for (final Classifier.Recognition result : results) {
          final RectF location = result.getLocation();
          if (location != null && result.getConfidence() >= MINIMUM_CONFIDENCE_TF_OD_API) {
            canvas.drawRect(location, paint);
//            cropToFrameTransform.mapRect(location);
//
//            result.setLocation(location);
//            mappedRecognitions.add(result);
          }
        }
//        tracker.trackResults(mappedRecognitions, new Random().nextInt());
//        trackingOverlay.postInvalidate();
        imageView.setImageBitmap(bitmap);
      }
    }
```

➔Detector Activity

```
package org.tensorflow.lite.examples.detection;

import android.content.res.AssetFileDescriptor;
import android.content.res.AssetManager;
import android.graphics.Bitmap;
import android.graphics.Bitmap.Config;
import android.graphics.Canvas;
import android.graphics.Color;
import android.graphics.Matrix;
import android.graphics.Paint;
import android.graphics.Paint.Style;
import android.graphics.RectF;
import android.graphics.Typeface;
import android.media.ImageReader.OnImageAvailableListener;
import android.media.MediaPlayer;
import android.os.SystemClock;
import android.util.Log;
import android.util.Size;
import android.util.TypedValue;
import android.widget.CheckBox;
import android.widget.Toast;

import java.io.IOException;
import java.util.ArrayList;
import java.util.HashMap;
import java.util.LinkedList;
import java.util.List;

import org.tensorflow.lite.examples.detection.customview.OverlayView;
import org.tensorflow.lite.examples.detection.customview.OverlayView.DrawCallback;
import org.tensorflow.lite.examples.detection.env.BorderedText;
import org.tensorflow.lite.examples.detection.env.ImageUtils;
import org.tensorflow.lite.examples.detection.env.Logger;
import org.tensorflow.lite.examples.detection.tflite.Classifier;
import org.tensorflow.lite.examples.detection.tflite.DetectorFactory;
import org.tensorflow.lite.examples.detection.tflite.YoloV5Classifier;
import org.tensorflow.lite.examples.detection.tracking.MultiBoxTracker;

public class DetectorActivity extends CameraActivity implements OnImageAvailableListener {
  private static final Logger LOGGER = new Logger();

  public static MediaPlayer player = new MediaPlayer();

  public static AssetManager globalassetmanager;

  private static final DetectorMode MODE = DetectorMode.TF_OD_API;
  private static final float MINIMUM_CONFIDENCE_TF_OD_API = 0.3f;
  private static final boolean MAINTAIN_ASPECT = true;
  private static final Size DESIRED_PREVIEW_SIZE = new Size(640, 640);
  private static final boolean SAVE_PREVIEW_BITMAP = false;
  private static final float TEXT_SIZE_DIP = 10;
  OverlayView trackingOverlay;
  private Integer sensorOrientation;

  private YoloV5Classifier detector;

  private long lastProcessingTimeMs;
  private Bitmap rgbFrameBitmap = null;
  private Bitmap croppedBitmap = null;
```

```
private Bitmap cropCopyBitmap = null;

private boolean computingDetection = false;

private long timestamp = 0;

private Matrix frameToCropTransform;
private Matrix cropToFrameTransform;

private MultiBoxTracker tracker;

private BorderedText borderedText;

private CheckBox helmet, mask, gloves, vest, shoes;

public int HELMET = 0, MASK = 0, VEST = 0, SHOES = 0, GLOVES = 0;

@Override
public void onPreviewSizeChosen(final Size size, final int rotation) {
  final float textSizePx =
      TypedValue.applyDimension(
          TypedValue.COMPLEX_UNIT_DIP, TEXT_SIZE_DIP, getResources().getDisplayMetrics());
  borderedText = new BorderedText(textSizePx);
  borderedText.setTypeface(Typeface.MONOSPACE);

  tracker = new MultiBoxTracker(this);

  final int modelIndex = modelView.getCheckedItemPosition();
  final String modelString = modelStrings.get(modelIndex);

  try {
    detector = DetectorFactory.getDetector(getAssets(), modelString);
  } catch (final IOException e) {
    e.printStackTrace();
    LOGGER.e(e, "Exception initializing classifier!");
    Toast toast =
        Toast.makeText(
            getApplicationContext(), "Classifier could not be initialized", Toast.LENGTH_SHORT);
    toast.show();
    finish();
  }

  int cropSize = detector.getInputSize();

  previewWidth = size.getWidth();
  previewHeight = size.getHeight();

  sensorOrientation = rotation - getScreenOrientation();
  LOGGER.i("Camera orientation relative to screen canvas: %d", sensorOrientation);

  LOGGER.i("Initializing at size %dx%d", previewWidth, previewHeight);
  rgbFrameBitmap = Bitmap.createBitmap(previewWidth, previewHeight, Config.ARGB_8888);
  croppedBitmap = Bitmap.createBitmap(cropSize, cropSize, Config.ARGB_8888);

  frameToCropTransform =
      ImageUtils.getTransformationMatrix(
          previewWidth, previewHeight,
          cropSize, cropSize,
          sensorOrientation, MAINTAIN_ASPECT);

  cropToFrameTransform = new Matrix();
  frameToCropTransform.invert(cropToFrameTransform);
```

```
    trackingOverlay = (OverlayView) findViewById(R.id.tracking_overlay);
    trackingOverlay.addCallback(
            new DrawCallback() {
              @Override
              public void drawCallback(final Canvas canvas) {
                tracker.draw(canvas);
                if (isDebug()) {
                  tracker.drawDebug(canvas);
                }
              }
            });

    tracker.setFrameConfiguration(previewWidth, previewHeight, sensorOrientation);
}

public  ArrayList<Object> getValues(){
    ArrayList<Object> vals = new ArrayList<Object>();

    vals.add(this.HELMET);
    vals.add(this.MASK);
    vals.add(this.GLOVES);
    vals.add(this.VEST);
    vals.add(this.SHOES);

    return vals;
}

protected void updateActiveModel() {
    // Get UI information before delegating to background
    final int modelIndex = modelView.getCheckedItemPosition();
    final int deviceIndex = deviceView.getCheckedItemPosition();
    String threads = threadsTextView.getText().toString().trim();
    final int numThreads = Integer.parseInt(threads);

    handler.post(() -> {
      if (modelIndex == currentModel && deviceIndex == currentDevice
          && numThreads == currentNumThreads) {
        return;
      }
      currentModel = modelIndex;
      currentDevice = deviceIndex;
      currentNumThreads = numThreads;

      // Disable classifier while updating
      if (detector != null) {
        detector.close();
        detector = null;
      }

      // Lookup names of parameters.
      String modelString = modelStrings.get(modelIndex);
      String device = deviceStrings.get(deviceIndex);

      LOGGER.i("Changing model to " + modelString + " device " + device);

      // Try to load model.

      try {
        detector = DetectorFactory.getDetector(getAssets(), modelString);
        // Customize the interpreter to the type of device we want to use.
        if (detector == null) {
          return;
        }
```

```
      }
      catch(IOException e) {
        e.printStackTrace();
        LOGGER.e(e, "Exception in updateActiveModel()");
        Toast toast =
            Toast.makeText(
                getApplicationContext(), "Classifier could not be initialized", Toast.LENGTH_SHORT);
        toast.show();
        finish();
      }


      if (device.equals("CPU")) {
        detector.useCPU();
      } else if (device.equals("GPU")) {
        detector.useGpu();
      } else if (device.equals("NNAPI")) {
        detector.useNNAPI();
      }
      detector.setNumThreads(numThreads);

      int cropSize = detector.getInputSize();
      croppedBitmap = Bitmap.createBitmap(cropSize, cropSize, Config.ARGB_8888);

      frameToCropTransform =
          ImageUtils.getTransformationMatrix(
              previewWidth, previewHeight,
              cropSize, cropSize,
              sensorOrientation, MAINTAIN_ASPECT);

      cropToFrameTransform = new Matrix();
      frameToCropTransform.invert(cropToFrameTransform);
    });
  }

  @Override
  protected void processImage() {
    ++timestamp;
    final long currTimestamp = timestamp;
    trackingOverlay.postInvalidate();

    // No mutex needed as this method is not reentrant.
    if (computingDetection) {
      readyForNextImage();
      return;
    }
    computingDetection = true;
    LOGGER.i("Preparing image " + currTimestamp + " for detection in bg thread.");

    rgbFrameBitmap.setPixels(getRgbBytes(), 0, previewWidth, 0, 0, previewWidth, previewHeight);

    readyForNextImage();

    final Canvas canvas = new Canvas(croppedBitmap);
    canvas.drawBitmap(rgbFrameBitmap, frameToCropTransform, null);
    // For examining the actual TF input.
    if (SAVE_PREVIEW_BITMAP) {
      ImageUtils.saveBitmap(croppedBitmap);
    }

    runInBackground(
        new Runnable() {
          @Override
```

```
public synchronized void run() {
    LOGGER.i("Running detection on image " + currTimestamp);
    final long startTime = SystemClock.uptimeMillis();
    final List<Classifier.Recognition> results = detector.recognizeImage(croppedBitmap);
    lastProcessingTimeMs = SystemClock.uptimeMillis() - startTime;

    Log.e("CHECK", "run: " + results.size());

    cropCopyBitmap = Bitmap.createBitmap(croppedBitmap);
    final Canvas canvas = new Canvas(cropCopyBitmap);
    final Paint paint = new Paint();
    paint.setColor(Color.RED);
    paint.setStyle(Style.STROKE);
    paint.setStrokeWidth(2.0f);

    float minimumConfidence = MINIMUM_CONFIDENCE_TF_OD_API;
    switch (MODE) {
      case TF_OD_API:
        minimumConfidence = MINIMUM_CONFIDENCE_TF_OD_API;
        break;
    }

    final List<Classifier.Recognition> mappedRecognitions =
        new LinkedList<Classifier.Recognition>();

    helmet = findViewById(R.id.checkbox_helmet);
    mask = findViewById(R.id.checkbox_mask);
    gloves = findViewById(R.id.checkbox_gloves);
    vest = findViewById(R.id.checkbox_vest);
    shoes = findViewById(R.id.checkbox_shoes);


    for(final Classifier.Recognition result : results){
      if (result.getTitle().equals("helmet")){
        helmet.setChecked(true);
        HELMET = 1;
      } else if (result.getTitle().equals("Mask")){
        mask.setChecked(true);
        MASK = 1;
      } else if (result.getTitle().equals("Gloves")){
        gloves.setChecked(true);
        GLOVES = 1;
      } else if (result.getTitle().equals("vest")){
        vest.setChecked(true);
        VEST = 1;
      } else if (result.getTitle().equals("safety_shoe")){
        shoes.setChecked(true);
        SHOES = 1;
      }
    }

    if (HELMET == 1 && VEST == 1 && MASK == 1 && GLOVES == 1 && SHOES == 1){
      try {
//
        helmet.setChecked(false);
        VEST = 0;
        vest.setChecked(false);
        MASK = 0;
        mask.setChecked(false);
        GLOVES = 0;
        gloves.setChecked(false);
        SHOES = 0;
        shoes.setChecked(false);
```

```
            } catch (Exception e) {
                throw new RuntimeException(e);
            }
        }

        Log.d("HELMET : ", Integer.toString(HELMET));
        Log.d("VEST : ", Integer.toString(VEST));
        Log.d("MASK : ", Integer.toString(MASK));
        Log.d("GLOVES : ", Integer.toString(GLOVES));
        Log.d("SHOES : ", Integer.toString(SHOES));
try {

            AssetFileDescriptor afd = getAssets().openFd("helmet_mask_vest_gloves_shoes.mp3");
            System.out.println("YYYYY "+HELMET+" " + MASK + " " + GLOVES + " " + VEST + " " + SHOES);
            if (HELMET == 0){
                System.out.println("YYYYY HELMET 0");
                if (VEST == 0){
                    if (MASK == 0){
                        if (GLOVES == 0){
                            if (SHOES == 0){
                                afd = getAssets().openFd("helmet_mask_vest_gloves_shoes.mp3");
                            } else {
                                afd = getAssets().openFd("helmet_mask_vest_gloves.mp3");
                            }
                        } else {
                            if (SHOES == 0){
                                afd = getAssets().openFd("helmet_mask_vest_shoes.mp3");
                            } else {
                                afd = getAssets().openFd("helmet mask and vest.mp3");
                            }
                        }
                    } else {
                        if (GLOVES == 0){
                            if (SHOES == 0){
                                afd = getAssets().openFd("helmet vest gloves shoes.mp3");
                            } else {
                                afd = getAssets().openFd("helmet vest gloves.mp3");
                            }
                        } else {
                            if (SHOES == 0){
                                afd = getAssets().openFd("helmet vest shoes.mp3");
                            } else {
                                afd = getAssets().openFd("helmet vest.mp3");
                            }
                        }
                    }
                } else {
                    if (MASK == 0){
                        if (GLOVES == 0){
                            if (SHOES == 0){
                                afd = getAssets().openFd("helmet mask gloves shoes.mp3");
                            } else {
                                afd = getAssets().openFd("helmet mask gloves.mp3");
                            }
                        } else { // GLOVES == 1
                            if (SHOES == 0){
                                afd = getAssets().openFd("helmet mask shoes.mp3");
                            } else {
                                afd = getAssets().openFd("helmet mask.mp3");
                            }
                        }
                    } else { // MASK == 1 HELMET == 0 VEST == 1
```

```
            if (GLOVES == 0){
               if (SHOES == 0){
                  afd = getAssets().openFd("helmet gloves shoes.mp3");
               } else {
                  afd = getAssets().openFd("helmet gloves.mp3");
               }
            } else { // GLOVES == 1
               if (SHOES == 0){
                  afd = getAssets().openFd("helmet shoes.mp3");
               } else {
                  afd = getAssets().openFd("helmet.mp3");
               }
            }
         }
      } else { // HELMET == 1
         System.out.println("YYYYY HELMET 1");
         if (VEST == 0){
            System.out.println("YYYYY VEST 0");
            if (MASK == 0){
               if (GLOVES == 0){
                  if (SHOES == 0){
                     afd = getAssets().openFd("maks vest gloves shoes.mp3");
                  } else {
                     afd = getAssets().openFd("mask vest gloves.mp3");
                  }
               } else { // GLOVES == 1
                  if (SHOES == 0){
                     afd = getAssets().openFd("mask vest shoes.mp3");
                  } else {
                     afd = getAssets().openFd("mask vest.mp3");
                  }
               }
            } else {
               System.out.println("YYYYY MASK 1");// MASK == 1 HELMET == 1 VEST == 0
               if (GLOVES == 0){
                  if (SHOES == 0){
                     afd = getAssets().openFd("vest gloves shoes.mp3");
                  } else {
                     afd = getAssets().openFd("vest gloves.mp3");
                  }
               } else {
                  System.out.println("YYYYY GLOVES 1 MP3");// GLOVES == 1
                  if (SHOES == 0){
                     afd = getAssets().openFd("vest shoes.mp3");
                  } else {
                     System.out.println("VEST MP3");
                     afd = getAssets().openFd("vest.mp3");
                  }
               }
            }
         } else { // VEST == 1, HELMET == 1
            if (MASK == 0){
               if (GLOVES == 0){
                  if (SHOES == 0){
                     afd = getAssets().openFd("mask gloves shoes.mp3");
                  } else {
                     afd = getAssets().openFd("mask gloves.mp3");
                  }
               } else { // GLOVES == 1
                  if (SHOES == 0){
                     afd = getAssets().openFd("mask shoes.mp3");
                  } else {
```

```
                        afd = getAssets().openFd("mask.mp3");
                    }
                }
            } else { // MASK == 1 HELMET == 1 VEST == 1
                if (GLOVES == 0){
                    if (SHOES == 0){
                        afd = getAssets().openFd("gloves shoes.mp3");
                    } else {
                        afd = getAssets().openFd("gloves.mp3");
                    }
                } else { // GLOVES == 1
                    if (SHOES == 0){
                        afd = getAssets().openFd("shoes.mp3");
                    } else {
                        afd = getAssets().openFd("verified.mp3");
                    }
                }
            }
        }
        if (!player.isPlaying()){
            player.reset();
            player.setDataSource(afd.getFileDescriptor(), afd.getStartOffset(), afd.getLength());
            afd.close();
            player.prepare();
            player.start();
        }

    } catch(Exception e){
        throw new RuntimeException(e);
    }


    for (final Classifier.Recognition result : results) {
        final RectF location = result.getLocation();
        if (location != null && result.getConfidence() >= minimumConfidence) {
            canvas.drawRect(location, paint);

            cropToFrameTransform.mapRect(location);

            result.setLocation(location);
            mappedRecognitions.add(result);
        }
    }

    tracker.trackResults(mappedRecognitions, currTimestamp);
    trackingOverlay.postInvalidate();

    computingDetection = false;

    runOnUiThread(
            new Runnable() {
                @Override
                public void run() {
                    showFrameInfo(previewWidth + "x" + previewHeight);
                    showCropInfo(cropCopyBitmap.getWidth() + "x" + cropCopyBitmap.getHeight());
                    showInference(lastProcessingTimeMs + "ms");
                }
            });
    }
});
}
```

```
    @Override
    protected int getLayoutId() {
      return R.layout.tfe_od_camera_connection_fragment_tracking;
    }

    @Override
    protected Size getDesiredPreviewFrameSize() {
      return DESIRED_PREVIEW_SIZE;
    }

    // Which detection model to use: by default uses Tensorflow Object Detection API frozen
    // checkpoints.
    private enum DetectorMode {
      TF_OD_API;
    }

    @Override
    protected void setUseNNAPI(final boolean isChecked) {
      runInBackground(() -> detector.setUseNNAPI(isChecked));
    }

    @Override
    protected void setNumThreads(final int numThreads) {
      runInBackground(() -> detector.setNumThreads(numThreads));
    }
}
```

➔Camera Activity

```
package org.tensorflow.lite.examples.detection;

import android.Manifest;
import android.app.Fragment;
import android.content.Context;
import android.content.pm.PackageManager;
import android.content.res.AssetManager;
import android.hardware.Camera;
import android.hardware.camera2.CameraAccessException;
import android.hardware.camera2.CameraCharacteristics;
import android.hardware.camera2.CameraManager;
import android.hardware.camera2.params.StreamConfigurationMap;
import android.media.Image;
import android.media.Image.Plane;
import android.media.ImageReader;
import android.media.ImageReader.OnImageAvailableListener;
import android.os.Build;
import android.os.Bundle;
import android.os.Handler;
import android.os.HandlerThread;
import android.os.Trace;
import androidx.annotation.NonNull;
import androidx.appcompat.app.AppCompatActivity;
import androidx.appcompat.widget.Toolbar;

import android.util.Size;
import android.view.Surface;
import android.view.View;
import android.view.ViewTreeObserver;
import android.view.WindowManager;
import android.widget.AdapterView;
import android.widget.ArrayAdapter;
```

```
import android.widget.CheckBox;
import android.widget.CompoundButton;
import android.widget.ImageView;
import android.widget.LinearLayout;
import android.widget.ListView;
import android.widget.TextView;
import android.widget.Toast;
import com.google.android.material.bottomsheet.BottomSheetBehavior;

import java.io.IOException;
import java.nio.ByteBuffer;
import java.util.ArrayList;

import org.tensorflow.lite.examples.detection.env.ImageUtils;
import org.tensorflow.lite.examples.detection.env.Logger;

public abstract class CameraActivity extends AppCompatActivity
    implements OnImageAvailableListener,
        Camera.PreviewCallback,
//        CompoundButton.OnCheckedChangeListener,
        View.OnClickListener {
  private static final Logger LOGGER = new Logger();

  private static final int PERMISSIONS_REQUEST = 1;

  private static final String PERMISSION_CAMERA = Manifest.permission.CAMERA;
  private static final String ASSET_PATH = "";
  protected int previewWidth = 0;
  protected int previewHeight = 0;
  private boolean debug = false;
  protected Handler handler;
  private HandlerThread handlerThread;
  private boolean useCamera2API;
  private boolean isProcessingFrame = false;
  private byte[][] yuvBytes = new byte[3][];
  private int[] rgbBytes = null;
  private int yRowStride;
  protected int defaultModelIndex = 0;
  protected int defaultDeviceIndex = 0;
  private Runnable postInferenceCallback;
  private Runnable imageConverter;
  protected ArrayList<String> modelStrings = new ArrayList<String>();

  private LinearLayout bottomSheetLayout;
  private LinearLayout gestureLayout;
  private BottomSheetBehavior<LinearLayout> sheetBehavior;

  protected TextView frameValueTextView, cropValueTextView, inferenceTimeTextView;
  protected ImageView bottomSheetArrowImageView;
  private ImageView plusImageView, minusImageView;
  protected ListView deviceView;
  protected TextView threadsTextView;
  protected ListView modelView;
  /** Current indices of device and model. */
  int currentDevice = -1;
  int currentModel = -1;
  int currentNumThreads = -1;

  private CheckBox helmet, mask, gloves, vest, shoes;

  ArrayList<String> deviceStrings = new ArrayList<String>();

  @Override
```

```
protected void onCreate(final Bundle savedInstanceState) {
  LOGGER.d("onCreate " + this);
  super.onCreate(null);
  getWindow().addFlags(WindowManager.LayoutParams.FLAG_KEEP_SCREEN_ON);

  setContentView(R.layout.tfe_od_activity_camera);
  Toolbar toolbar = findViewById(R.id.toolbar);
  setSupportActionBar(toolbar);
  getSupportActionBar().setDisplayShowTitleEnabled(false);

  if (hasPermission()) {
    setFragment();
  } else {
    requestPermission();
  }

  helmet = findViewById(R.id.checkbox_helmet);
  mask = findViewById(R.id.checkbox_mask);
  gloves = findViewById(R.id.checkbox_gloves);
  vest = findViewById(R.id.checkbox_vest);
  shoes = findViewById(R.id.checkbox_shoes);

  DetectorActivity det_obj = new DetectorActivity()
  ArrayList<Object> val = det_obj.getValues();

  System.out.println("YYYYYY " + val.get(0).toString());

  if(val.get(0).toString().equals("1")){
    helmet.setChecked(true);
  }


  threadsTextView = findViewById(R.id.threads);
  currentNumThreads = Integer.parseInt(threadsTextView.getText().toString().trim());
  plusImageView = findViewById(R.id.plus);
  minusImageView = findViewById(R.id.minus);
  deviceView = findViewById(R.id.device_list);
  deviceStrings.add("CPU");
  deviceStrings.add("GPU");
  deviceStrings.add("NNAPI");
  deviceView.setChoiceMode(ListView.CHOICE_MODE_SINGLE);
  ArrayAdapter<String> deviceAdapter =
        new ArrayAdapter<>(
              CameraActivity.this , R.layout.deviceview_row, R.id.deviceview_row_text, deviceStrings);
  deviceView.setAdapter(deviceAdapter);
  deviceView.setItemChecked(defaultDeviceIndex, true);
  currentDevice = defaultDeviceIndex;
  deviceView.setOnItemClickListener(
        new AdapterView.OnItemClickListener() {
          @Override
          public void onItemClick(AdapterView<?> parent, View view, int position, long id) {
            updateActiveModel();
          }
        });

  bottomSheetLayout = findViewById(R.id.bottom_sheet_layout);
  gestureLayout = findViewById(R.id.gesture_layout);
  sheetBehavior = BottomSheetBehavior.from(bottomSheetLayout);
  bottomSheetArrowImageView = findViewById(R.id.bottom_sheet_arrow);
  modelView = findViewById((R.id.model_list));

  modelStrings = getModelStrings(getAssets(), ASSET_PATH);
  modelView.setChoiceMode(ListView.CHOICE_MODE_SINGLE);
```

```
ArrayAdapter<String> modelAdapter =
      new ArrayAdapter<>(
            CameraActivity.this , R.layout.listview_row, R.id.listview_row_text, modelStrings);
modelView.setAdapter(modelAdapter);
modelView.setItemChecked(defaultModelIndex, true);
currentModel = defaultModelIndex;
modelView.setOnItemClickListener(
      new AdapterView.OnItemClickListener() {
        @Override
        public void onItemClick(AdapterView<?> parent, View view, int position, long id) {
          updateActiveModel();
        }
      });

ViewTreeObserver vto = gestureLayout.getViewTreeObserver();
vto.addOnGlobalLayoutListener(
   new ViewTreeObserver.OnGlobalLayoutListener() {
     @Override
     public void onGlobalLayout() {
       if (Build.VERSION.SDK_INT < Build.VERSION_CODES.JELLY_BEAN) {
         gestureLayout.getViewTreeObserver().removeGlobalOnLayoutListener(this);
       } else {
         gestureLayout.getViewTreeObserver().removeOnGlobalLayoutListener(this);
       }
       //              int width = bottomSheetLayout.getMeasuredWidth();
       int height = gestureLayout.getMeasuredHeight();

       sheetBehavior.setPeekHeight(height);
     }
   });
sheetBehavior.setHideable(false);

sheetBehavior.setBottomSheetCallback(
   new BottomSheetBehavior.BottomSheetCallback() {
     @Override
     public void onStateChanged(@NonNull View bottomSheet, int newState) {
       switch (newState) {
         case BottomSheetBehavior.STATE_HIDDEN:
           break;
         case BottomSheetBehavior.STATE_EXPANDED:
           {
             bottomSheetArrowImageView.setImageResource(R.drawable.icn_chevron_down);
           }
           break;
         case BottomSheetBehavior.STATE_COLLAPSED:
           {
             bottomSheetArrowImageView.setImageResource(R.drawable.icn_chevron_up);
           }
           break;
         case BottomSheetBehavior.STATE_DRAGGING:
           break;
         case BottomSheetBehavior.STATE_SETTLING:
           bottomSheetArrowImageView.setImageResource(R.drawable.icn_chevron_up);
           break;
       }
     }

     @Override
     public void onSlide(@NonNull View bottomSheet, float slideOffset) {}
   });

frameValueTextView = findViewById(R.id.frame_info);
cropValueTextView = findViewById(R.id.crop_info);
```

```
  inferenceTimeTextView = findViewById(R.id.inference_info);

  plusImageView.setOnClickListener(this);
  minusImageView.setOnClickListener(this);
}



protected ArrayList<String> getModelStrings(AssetManager mgr, String path){
  ArrayList<String> res = new ArrayList<String>();
  try {
    String[] files = mgr.list(path);
    for (String file : files) {
      String[] splits = file.split("\\.");
      if (splits[splits.length - 1].equals("tflite")) {
        res.add(file);
      }
    }

  }
  catch (IOException e){
    System.err.println("getModelStrings: " + e.getMessage());
  }
  return res;
}



protected int[] getRgbBytes() {
  imageConverter.run();
  return rgbBytes;
}

protected int getLuminanceStride() {
  return yRowStride;
}

protected byte[] getLuminance() {
  return yuvBytes[0];
}

/** Callback for android.hardware.Camera API */
@Override
public void onPreviewFrame(final byte[] bytes, final Camera camera) {
  if (isProcessingFrame) {
    LOGGER.w("Dropping frame!");
    return;
  }

  try {
    // Initialize the storage bitmaps once when the resolution is known.
    if (rgbBytes == null) {
      Camera.Size previewSize = camera.getParameters().getPreviewSize();
      previewHeight = previewSize.height;
      previewWidth = previewSize.width;
      rgbBytes = new int[previewWidth * previewHeight];
      onPreviewSizeChosen(new Size(previewSize.width, previewSize.height), 90);
    }
  } catch (final Exception e) {
    LOGGER.e(e, "Exception!");
    return;
  }
```

```
    isProcessingFrame = true;
    yuvBytes[0] = bytes;
    yRowStride = previewWidth;

    imageConverter =
        new Runnable() {
          @Override
          public void run() {
            ImageUtils.convertYUV420SPToARGB8888(bytes, previewWidth, previewHeight, rgbBytes);
          }
        };
    postInferenceCallback =
        new Runnable() {
          @Override
          public void run() {
            camera.addCallbackBuffer(bytes);
            isProcessingFrame = false;
          }
        };
    processImage();
  }

  /** Callback for Camera2 API */
  @Override
  public void onImageAvailable(final ImageReader reader) {
    // We need wait until we have some size from onPreviewSizeChosen
    if (previewWidth == 0 || previewHeight == 0) {
      return;
    }
    if (rgbBytes == null) {
      rgbBytes = new int[previewWidth * previewHeight];
    }
    try {
      final Image image = reader.acquireLatestImage();

      if (image == null) {
        return;
      }

      if (isProcessingFrame) {
        image.close();
        return;
      }
      isProcessingFrame = true;
      Trace.beginSection("imageAvailable");
      final Plane[] planes = image.getPlanes();
      fillBytes(planes, yuvBytes);
      yRowStride = planes[0].getRowStride();
      final int uvRowStride = planes[1].getRowStride();
      final int uvPixelStride = planes[1].getPixelStride();

      imageConverter =
          new Runnable() {
            @Override
            public void run() {
              ImageUtils.convertYUV420ToARGB8888(
                  yuvBytes[0],
                  yuvBytes[1],
                  yuvBytes[2],
                  previewWidth,
                  previewHeight,
                  yRowStride,
                  uvRowStride,
```

```
            uvPixelStride,
            rgbBytes);
      }
    };

  postInferenceCallback =
      new Runnable() {
        @Override
        public void run() {
          image.close();
          isProcessingFrame = false;
        }
      };

  processImage();
 } catch (final Exception e) {
  LOGGER.e(e, "Exception!");
  Trace.endSection();
  return;
 }
 Trace.endSection();
}

@Override
public synchronized void onStart() {
 LOGGER.d("onStart " + this);
 super.onStart();
}

@Override
public synchronized void onResume() {
 LOGGER.d("onResume " + this);
 super.onResume();

 handlerThread = new HandlerThread("inference");
 handlerThread.start();
 handler = new Handler(handlerThread.getLooper());
}

@Override
public synchronized void onPause() {
 LOGGER.d("onPause " + this);

 handlerThread.quitSafely();
 try {
  handlerThread.join();
  handlerThread = null;
  handler = null;
 } catch (final InterruptedException e) {
  LOGGER.e(e, "Exception!");
 }

 super.onPause();
}

@Override
public synchronized void onStop() {
 LOGGER.d("onStop " + this);
 super.onStop();
}

@Override
public synchronized void onDestroy() {
```

```
  LOGGER.d("onDestroy " + this);
  super.onDestroy();
}

protected synchronized void runInBackground(final Runnable r) {
  if (handler != null) {
    handler.post(r);
  }
}

@Override
public void onRequestPermissionsResult(
    final int requestCode, final String[] permissions, final int[] grantResults) {
  super.onRequestPermissionsResult(requestCode, permissions, grantResults);
  if (requestCode == PERMISSIONS_REQUEST) {
    if (allPermissionsGranted(grantResults)) {
      setFragment();
    } else {
      requestPermission();
    }
  }
}

private static boolean allPermissionsGranted(final int[] grantResults) {
  for (int result : grantResults) {
    if (result != PackageManager.PERMISSION_GRANTED) {
      return false;
    }
  }
  return true;
}

private boolean hasPermission() {
  if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.M) {
    return checkSelfPermission(PERMISSION_CAMERA) == PackageManager.PERMISSION_GRANTED;
  } else {
    return true;
  }
}

private void requestPermission() {
  if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.M) {
    if (shouldShowRequestPermissionRationale(PERMISSION_CAMERA)) {
      Toast.makeText(
          CameraActivity.this,
          "Camera permission is required for this demo",
          Toast.LENGTH_LONG)
        .show();
    }
    requestPermissions(new String[] {PERMISSION_CAMERA}, PERMISSIONS_REQUEST);
  }
}

// Returns true if the device supports the required hardware level, or better.
private boolean isHardwareLevelSupported(
    CameraCharacteristics characteristics, int requiredLevel) {
  int deviceLevel = characteristics.get(CameraCharacteristics.INFO_SUPPORTED_HARDWARE_LEVEL);
  if (deviceLevel == CameraCharacteristics.INFO_SUPPORTED_HARDWARE_LEVEL_LEGACY) {
    return requiredLevel == deviceLevel;
  }
  // deviceLevel is not LEGACY, can use numerical sort
  return requiredLevel <= deviceLevel;
}
```

```
private String chooseCamera() {
  final CameraManager manager = (CameraManager) getSystemService(Context.CAMERA_SERVICE);
  try {
    for (final String cameraId : manager.getCameraIdList()) {
      final CameraCharacteristics characteristics = manager.getCameraCharacteristics(cameraId);

      // We don't use a front facing camera in this sample.
      final Integer facing = characteristics.get(CameraCharacteristics.LENS_FACING);
      if (facing != null && facing == CameraCharacteristics.LENS_FACING_FRONT) {
        continue;
      }

      final StreamConfigurationMap map =
          characteristics.get(CameraCharacteristics.SCALER_STREAM_CONFIGURATION_MAP);

      if (map == null) {
        continue;
      }

      // Fallback to camera1 API for internal cameras that don't have full support.
      // This should help with legacy situations where using the camera2 API causes
      // distorted or otherwise broken previews.
      useCamera2API =
          (facing == CameraCharacteristics.LENS_FACING_EXTERNAL)
              || isHardwareLevelSupported(
                  characteristics, CameraCharacteristics.INFO_SUPPORTED_HARDWARE_LEVEL_FULL);
      LOGGER.i("Camera API lv2?: %s", useCamera2API);
      return cameraId;
    }
  } catch (CameraAccessException e) {
    LOGGER.e(e, "Not allowed to access camera");
  }

  return null;
}

protected void setFragment() {
  String cameraId = chooseCamera();

  Fragment fragment;
  if (useCamera2API) {
    CameraConnectionFragment camera2Fragment =
        CameraConnectionFragment.newInstance(
            new CameraConnectionFragment.ConnectionCallback() {
              @Override
              public void onPreviewSizeChosen(final Size size, final int rotation) {
                previewHeight = size.getHeight();
                previewWidth = size.getWidth();
                CameraActivity.this.onPreviewSizeChosen(size, rotation);
              }
            },
            this,
            getLayoutId(),
            getDesiredPreviewFrameSize());

    camera2Fragment.setCamera(cameraId);
    fragment = camera2Fragment;
  } else {
    fragment =
        new LegacyCameraConnectionFragment(this, getLayoutId(), getDesiredPreviewFrameSize());
  }
```

```
  getFragmentManager().beginTransaction().replace(R.id.container, fragment).commit();
}

protected void fillBytes(final Plane[] planes, final byte[][] yuvBytes) {
  // Because of the variable row stride it's not possible to know in
  // advance the actual necessary dimensions of the yuv planes.
  for (int i = 0; i < planes.length; ++i) {
    final ByteBuffer buffer = planes[i].getBuffer();
    if (yuvBytes[i] == null) {
      LOGGER.d("Initializing buffer %d at size %d", i, buffer.capacity());
      yuvBytes[i] = new byte[buffer.capacity()];
    }
    buffer.get(yuvBytes[i]);
  }
}

public boolean isDebug() {
  return debug;
}

protected void readyForNextImage() {
  if (postInferenceCallback != null) {
    postInferenceCallback.run();
  }
}

protected int getScreenOrientation() {
  switch (getWindowManager().getDefaultDisplay().getRotation()) {
    case Surface.ROTATION_270:
      return 270;
    case Surface.ROTATION_180:
      return 180;
    case Surface.ROTATION_90:
      return 90;
    default:
      return 0;
  }
}
@Override
public void onClick(View v) {
  if (v.getId() == R.id.plus) {
    String threads = threadsTextView.getText().toString().trim();
    int numThreads = Integer.parseInt(threads);
    if (numThreads >= 9) return;
    numThreads++;
    threadsTextView.setText(String.valueOf(numThreads));
    setNumThreads(numThreads);
  } else if (v.getId() == R.id.minus) {
    String threads = threadsTextView.getText().toString().trim();
    int numThreads = Integer.parseInt(threads);
    if (numThreads == 1) {
      return;
    }
    numThreads--;
    threadsTextView.setText(String.valueOf(numThreads));
    setNumThreads(numThreads);
  }
}

protected void showFrameInfo(String frameInfo) {
  frameValueTextView.setText(frameInfo);
}
```

```
  protected void showCropInfo(String cropInfo) {
    cropValueTextView.setText(cropInfo);
  }

  protected void showInference(String inferenceTime) {
    inferenceTimeTextView.setText(inferenceTime);
  }

  protected abstract void updateActiveModel();
  protected abstract void processImage();
  protected abstract void onPreviewSizeChosen(final Size size, final int rotation);
  protected abstract int getLayoutId();
  protected abstract Size getDesiredPreviewFrameSize();
  protected abstract void setNumThreads(int numThreads);
  protected abstract void setUseNNAPI(boolean isChecked);
}
```

➔Classifier

```
package org.tensorflow.lite.examples.detection.tflite;

import android.graphics.Bitmap;
import android.graphics.RectF;

import java.util.List;

/**
 * Generic interface for interacting with different recognition engines.
 */
public interface Classifier {
  List<Recognition> recognizeImage(Bitmap bitmap);

  void enableStatLogging(final boolean debug);

  String getStatString();

  void close();

  void setNumThreads(int num_threads);

  void setUseNNAPI(boolean isChecked);

  abstract float getObjThresh();

  /**
   * An immutable result returned by a Classifier describing what was recognized.
   */
  public class Recognition {
    /**
     * A unique identifier for what has been recognized. Specific to the class, not the instance of
     * the object.
     */
    private final String id;

    /**
     * Display name for the recognition.
     */
    private final String title;

    /**
     * A sortable score for how good the recognition is relative to others. Higher should be better.
     */
```

```
private final Float confidence;

/**
 * Optional location within the source image for the location of the recognized object.
 */
private RectF location;

private int detectedClass;

public Recognition(
      final String id, final String title, final Float confidence, final RectF location) {
   this.id = id;
   this.title = title;
   this.confidence = confidence;
   this.location = location;
}

public Recognition(final String id, final String title, final Float confidence, final RectF location, int detectedClass) {
   this.id = id;
   this.title = title;
   this.confidence = confidence;
   this.location = location;
   this.detectedClass = detectedClass;
}

public String getId() {
   return id;
}

public String getTitle() {
   return title;
}

public Float getConfidence() {
   return confidence;
}

public RectF getLocation() {
   return new RectF(location);
}

public void setLocation(RectF location) {
   this.location = location;
}

public int getDetectedClass() {
   return detectedClass;
}

public void setDetectedClass(int detectedClass) {
   this.detectedClass = detectedClass;
}

@Override
public String toString() {
   String resultString = "";
   if (id != null) {
      resultString += "[" + id + "] ";
   }

   if (title != null) {
      resultString += title + " ";
   }
```

```
            if (confidence != null) {
                resultString += String.format("(%.1f%%) ", confidence * 100.0f);
            }

            if (location != null) {
                resultString += location + " ";
            }

            return resultString.trim();
        }
    }
}
```

➔Yolo V5 Classifier

```
package org.tensorflow.lite.examples.detection.tflite;

import android.content.res.AssetFileDescriptor;
import android.content.res.AssetManager;
import android.graphics.Bitmap;
import android.graphics.RectF;
import android.os.Build;
import android.util.Log;
import android.media.MediaPlayer;

import org.tensorflow.lite.Interpreter;
import org.tensorflow.lite.Tensor;
import org.tensorflow.lite.examples.detection.MainActivity;
import org.tensorflow.lite.examples.detection.env.Logger;
import org.tensorflow.lite.examples.detection.env.Utils;
import org.tensorflow.lite.gpu.GpuDelegate;
import org.tensorflow.lite.nnapi.NnApiDelegate;

import java.io.File;
import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStream;
import java.io.InputStreamReader;
import java.nio.ByteBuffer;
import java.nio.ByteOrder;
import java.nio.MappedByteBuffer;
import java.util.ArrayList;
import java.util.Comparator;
import java.util.HashMap;
import java.util.Map;
import java.util.PriorityQueue;
import java.util.Vector;


public class YoloV5Classifier implements Classifier {

   public static AssetManager globalassetmanager;

   public static MediaPlayer player = new MediaPlayer();
   public static YoloV5Classifier create(
         final AssetManager assetManager,
         final String modelFilename,
         final String labelFilename,
         final boolean isQuantized,
         final int inputSize)
   throws IOException {
```

```
final YoloV5Classifier d = new YoloV5Classifier();
String actualFilename = labelFilename.split("file:///android_asset/")[1];
globalassetmanager = assetManager;
InputStream labelsInput = assetManager.open(actualFilename);
BufferedReader br = new BufferedReader(new InputStreamReader(labelsInput));
String line;
while ((line = br.readLine()) != null) {
   LOGGER.w(line);
   d.labels.add(line);
}
br.close();

try {
   Interpreter.Options options = (new Interpreter.Options());
   options.setNumThreads(NUM_THREADS);
   if (isNNAPI) {
      d.nnapiDelegate = null;
      // Initialize interpreter with NNAPI delegate for Android Pie or above
      if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.P) {
         d.nnapiDelegate = new NnApiDelegate();
         options.addDelegate(d.nnapiDelegate);
         options.setNumThreads(NUM_THREADS);
         options.setUseNNAPI(true);
      }
   }
   if (isGPU) {
      GpuDelegate.Options gpu_options = new GpuDelegate.Options();
      gpu_options.setPrecisionLossAllowed(true); // It seems that the default is true
      gpu_options.setInferencePreference(GpuDelegate.Options.INFERENCE_PREFERENCE_SUSTAINED_SPEED);
      d.gpuDelegate = new GpuDelegate(gpu_options);
      options.addDelegate(d.gpuDelegate);
   }
   d.tfliteModel = Utils.loadModelFile(assetManager, modelFilename);
   d.tfLite = new Interpreter(d.tfliteModel, options);
} catch (Exception e) {
   throw new RuntimeException(e);
}

d.isModelQuantized = isQuantized;
// Pre-allocate buffers.
int numBytesPerChannel;
if (isQuantized) {
   numBytesPerChannel = 1; // Quantized
} else {
   numBytesPerChannel = 4; // Floating point
}
d.INPUT_SIZE = inputSize;
d.imgData = ByteBuffer.allocateDirect(1 * d.INPUT_SIZE * d.INPUT_SIZE * 3 * numBytesPerChannel);
d.imgData.order(ByteOrder.nativeOrder());
d.intValues = new int[d.INPUT_SIZE * d.INPUT_SIZE];

d.output_box = (int) ((Math.pow((inputSize / 32), 2) + Math.pow((inputSize / 16), 2) + Math.pow((inputSize / 8), 2)) * 3);
if (d.isModelQuantized){
   Tensor inpten = d.tfLite.getInputTensor(0);
   d.inp_scale = inpten.quantizationParams().getScale();
   d.inp_zero_point = inpten.quantizationParams().getZeroPoint();
   Tensor oupten = d.tfLite.getOutputTensor(0);
   d.oup_scale = oupten.quantizationParams().getScale();
   d.oup_zero_point = oupten.quantizationParams().getZeroPoint();
}

int[] shape = d.tfLite.getOutputTensor(0).shape();
int numClass = shape[shape.length - 1] - 5;
```

```
        d.numClass = numClass;
        d.outData = ByteBuffer.allocateDirect(d.output_box * (numClass + 5) * numBytesPerChannel);
        d.outData.order(ByteOrder.nativeOrder());
        return d;
    }

    public int getInputSize() {
        return INPUT_SIZE;
    }
    @Override
    public void enableStatLogging(final boolean logStats) {
    }

    @Override
    public String getStatString() {
        return "";
    }

    @Override
    public void close() {
        tfLite.close();
        tfLite = null;
        if (gpuDelegate != null) {
            gpuDelegate.close();
            gpuDelegate = null;
        }
        if (nnapiDelegate != null) {
            nnapiDelegate.close();
            nnapiDelegate = null;
        }
        tfliteModel = null;
    }

    public void setNumThreads(int num_threads) {
        if (tfLite != null) tfLite.setNumThreads(num_threads);
    }

    @Override
    public void setUseNNAPI(boolean isChecked) {
//        if (tfLite != null) tfLite.setUseNNAPI(isChecked);
    }

    private void recreateInterpreter() {
        if (tfLite != null) {
            tfLite.close();
            tfLite = new Interpreter(tfliteModel, tfliteOptions);
        }
    }

    public void useGpu() {
        if (gpuDelegate == null) {
            gpuDelegate = new GpuDelegate();
            tfliteOptions.addDelegate(gpuDelegate);
            recreateInterpreter();
        }
    }

    public void useCPU() {
        recreateInterpreter();
    }

    public void useNNAPI() {
        nnapiDelegate = new NnApiDelegate();
```

```
    tfliteOptions.addDelegate(nnapiDelegate);
    recreateInterpreter();
}

@Override
public float getObjThresh() {
    return MainActivity.MINIMUM_CONFIDENCE_TF_OD_API;
}

private static final Logger LOGGER = new Logger();

// Float model
private final float IMAGE_MEAN = 0;

private final float IMAGE_STD = 255.0f;

//config yolo
private int INPUT_SIZE = -1;

private  int output_box;

private static final float[] XYSCALE = new float[]{1.2f, 1.1f, 1.05f};

private static final int NUM_BOXES_PER_BLOCK = 3;

// Number of threads in the java app
private static final int NUM_THREADS = 1;
private static boolean isNNAPI = false;
private static boolean isGPU = false;

private boolean isModelQuantized;

/** holds a gpu delegate */
GpuDelegate gpuDelegate = null;
/** holds an nnapi delegate */
NnApiDelegate nnapiDelegate = null;

/** The loaded TensorFlow Lite model. */
private MappedByteBuffer tfliteModel;

/** Options for configuring the Interpreter. */
private final Interpreter.Options tfliteOptions = new Interpreter.Options();

// Config values.

// Pre-allocated buffers.
private Vector<String> labels = new Vector<String>();
private int[] intValues;

private ByteBuffer imgData;
private ByteBuffer outData;

private Interpreter tfLite;
private float inp_scale;
private int inp_zero_point;
private float oup_scale;
private int oup_zero_point;
private int numClass;
private YoloV5Classifier() {
}

//non maximum suppression
protected ArrayList<Recognition> nms(ArrayList<Recognition> list) {
```

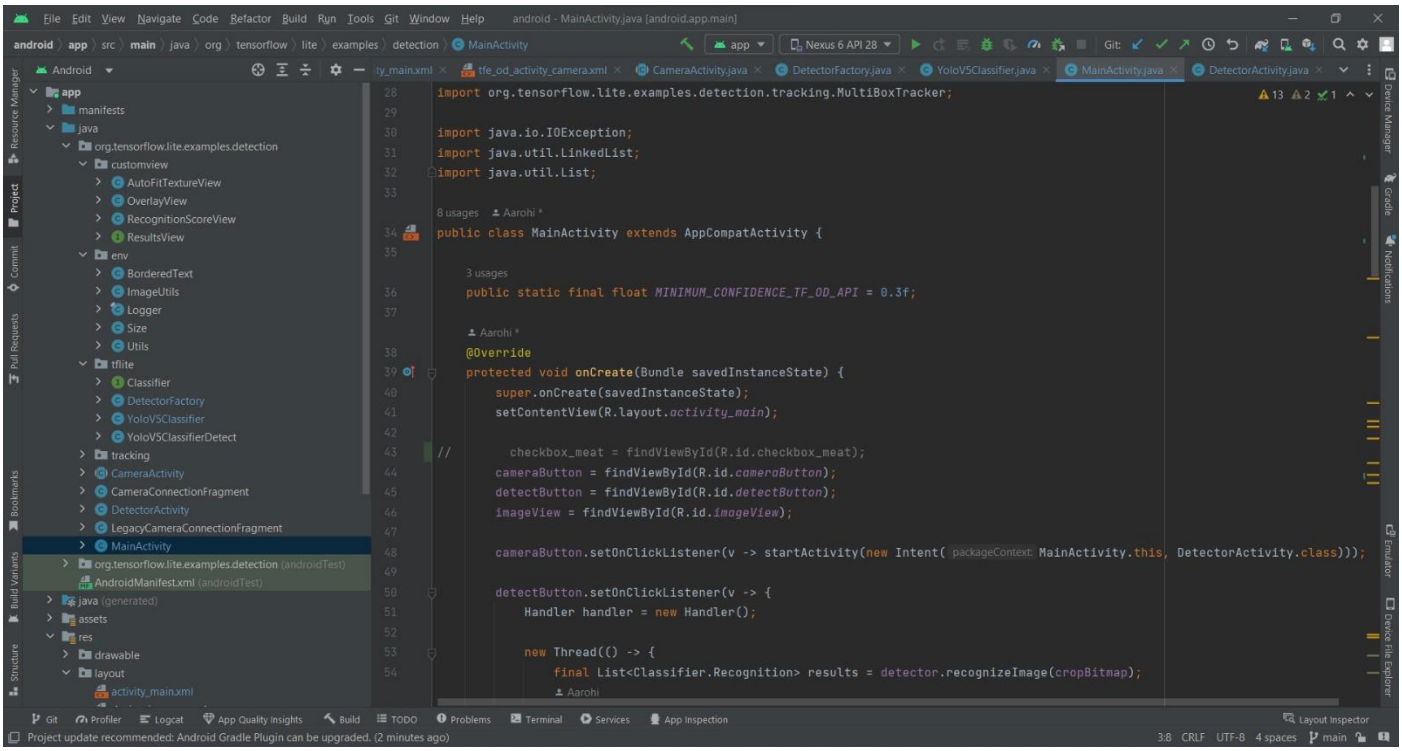
```
    ArrayList<Recognition> nmsList = new ArrayList<Recognition>();

    for (int k = 0; k < labels.size(); k++) {
      //1.find max confidence per class
      PriorityQueue<Recognition> pq =
          new PriorityQueue<Recognition>(
              50,
              new Comparator<Recognition>() {
                @Override
                public int compare(final Recognition lhs, final Recognition rhs) {
                  // Intentionally reversed to put high confidence at the head of the queue.
                  return Float.compare(rhs.getConfidence(), lhs.getConfidence());
                }
              });

      for (int i = 0; i < list.size(); ++i) {
        if (list.get(i).getDetectedClass() == k) {
          pq.add(list.get(i));
        }
      }

      //2.do non maximum suppression
      while (pq.size() > 0) {
        //insert detection with max confidence
        Recognition[] a = new Recognition[pq.size()];
        Recognition[] detections = pq.toArray(a);
        Recognition max = detections[0];
        nmsList.add(max);
        pq.clear();

        for (int j = 1; j < detections.length; j++) {
          Recognition detection = detections[j];
          RectF b = detection.getLocation();
          if (box_iou(max.getLocation(), b) < mNmsThresh) {
            pq.add(detection);
          }
        }
      }
    }
    return nmsList;
  }

  protected float mNmsThresh = 0.6f;

  protected float box_iou(RectF a, RectF b) {
    return box_intersection(a, b) / box_union(a, b);
  }

  protected float box_intersection(RectF a, RectF b) {
    float w = overlap((a.left + a.right) / 2, a.right - a.left,
        (b.left + b.right) / 2, b.right - b.left);
    float h = overlap((a.top + a.bottom) / 2, a.bottom - a.top,
        (b.top + b.bottom) / 2, b.bottom - b.top);
    if (w < 0 || h < 0) return 0;
    float area = w * h;
    return area;
  }

  protected float box_union(RectF a, RectF b) {
    float i = box_intersection(a, b);
    float u = (a.right - a.left) * (a.bottom - a.top) + (b.right - b.left) * (b.bottom - b.top) - i;
    return u;
  }
```

```
    protected float overlap(float x1, float w1, float x2, float w2) {
        float l1 = x1 - w1 / 2;
        float l2 = x2 - w2 / 2;
        float left = l1 > l2 ? l1 : l2;
        float r1 = x1 + w1 / 2;
        float r2 = x2 + w2 / 2;
        float right = r1 < r2 ? r1 : r2;
        return right - left;
    }

    protected static final int BATCH_SIZE = 1;
    protected static final int PIXEL_SIZE = 3;

    /**
     * Writes Image data into a {@code ByteBuffer}.
     */
    protected ByteBuffer convertBitmapToByteBuffer(Bitmap bitmap) {
        bitmap.getPixels(intValues, 0, bitmap.getWidth(), 0, 0, bitmap.getWidth(), bitmap.getHeight());
        int pixel = 0;

        imgData.rewind();
        for (int i = 0; i < INPUT_SIZE; ++i) {
            for (int j = 0; j < INPUT_SIZE; ++j) {
                int pixelValue = intValues[i * INPUT_SIZE + j];
                if (isModelQuantized) {
                    // Quantized model
                    imgData.put((byte) ((((pixelValue >> 16) & 0xFF) - IMAGE_MEAN) / IMAGE_STD / inp_scale +
inp_zero_point));
                    imgData.put((byte) ((((pixelValue >> 8) & 0xFF) - IMAGE_MEAN) / IMAGE_STD / inp_scale + inp_zero_point));
                    imgData.put((byte) (((pixelValue & 0xFF) - IMAGE_MEAN) / IMAGE_STD / inp_scale + inp_zero_point));
                } else { // Float model
                    imgData.putFloat((((pixelValue >> 16) & 0xFF) - IMAGE_MEAN) / IMAGE_STD);
                    imgData.putFloat((((pixelValue >> 8) & 0xFF) - IMAGE_MEAN) / IMAGE_STD);
                    imgData.putFloat(((pixelValue & 0xFF) - IMAGE_MEAN) / IMAGE_STD);
                }
            }
        }
        return imgData;
    }

    public ArrayList<Recognition> recognizeImage(Bitmap bitmap) {
        ByteBuffer byteBuffer_ = convertBitmapToByteBuffer(bitmap);

        Map<Integer, Object> outputMap = new HashMap<>();

//        float[][][] outbuf = new float[1][output_box][labels.size() + 5];
        outData.rewind();
        outputMap.put(0, outData);
        Log.d("YoloV5Classifier", "mObjThresh: " + getObjThresh());

        Object[] inputArray = {imgData};
        tfLite.runForMultipleInputsOutputs(inputArray, outputMap);

        ByteBuffer byteBuffer = (ByteBuffer) outputMap.get(0);
        byteBuffer.rewind();

        ArrayList<Recognition> detections = new ArrayList<Recognition>();

        float[][][] out = new float[1][output_box][numClass + 5];
        Log.d("YoloV5Classifier", "out[0] detect start");
        for (int i = 0; i < output_box; ++i) {
            for (int j = 0; j < numClass + 5; ++j) {
```

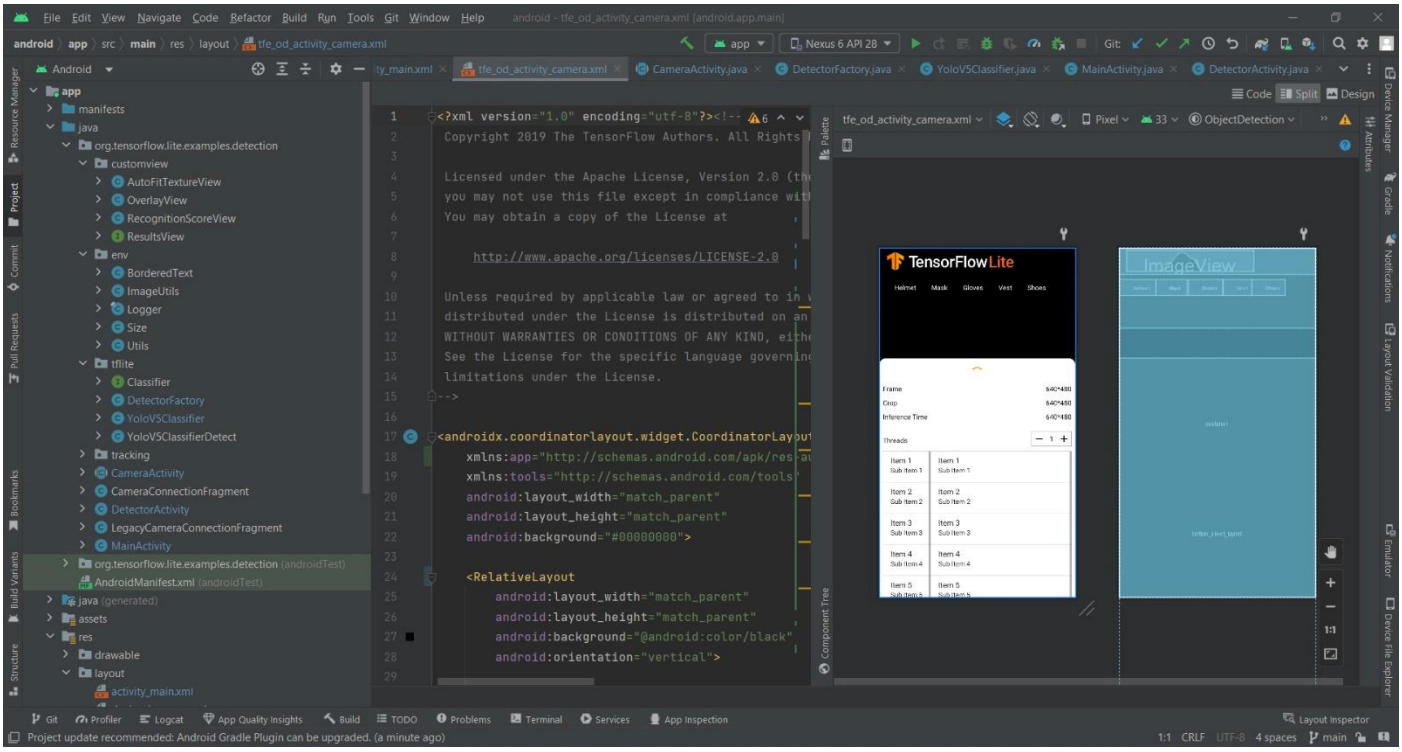
```
            if (isModelQuantized){
                out[0][i][j] = oup_scale * (((int) byteBuffer.get() & 0xFF) - oup_zero_point);
            }
            else {
                out[0][i][j] = byteBuffer.getFloat();
            }
        }
        // Denormalize xywh
        for (int j = 0; j < 4; ++j) {
            out[0][i][j] *= getInputSize();
        }
    }
    for (int i = 0; i < output_box; ++i){
        final int offset = 0;
        final float confidence = out[0][i][4];
        int detectedClass = -1;
        float maxClass = 0;

        final float[] classes = new float[labels.size()];
        for (int c = 0; c < labels.size(); ++c) {
            classes[c] = out[0][i][5 + c];
        }

        for (int c = 0; c < labels.size(); ++c) {
            if (classes[c] > maxClass) {
                detectedClass = c;
                maxClass = classes[c];
            }
        }

        final float confidenceInClass = maxClass * confidence;
        if (confidenceInClass > getObjThresh()) {
            final float xPos = out[0][i][0];
            final float yPos = out[0][i][1];

            final float w = out[0][i][2];
            final float h = out[0][i][3];
            Log.d("YoloV5Classifier",
                    Float.toString(xPos) + ',' + yPos + ',' + w + ',' + h);

            final RectF rect =
                    new RectF(
                            Math.max(0, xPos - w / 2),
                            Math.max(0, yPos - h / 2),
                            Math.min(bitmap.getWidth() - 1, xPos + w / 2),
                            Math.min(bitmap.getHeight() - 1, yPos + h / 2));

            Log.d("YOLOv5Classifier : ", labels.get(detectedClass));



            detections.add(new Recognition("" + offset, labels.get(detectedClass),
                    confidenceInClass, rect, detectedClass));
        }
    }

    Log.d("YoloV5Classifier", "detect end");
    final ArrayList<Recognition> recognitions = nms(detections);
    return recognitions;
}

public boolean checkInvalidateBox(float x, float y, float width, float height, float oriW, float oriH, int intputSize) {
    // (1) (x, y, w, h) --> (xmin, ymin, xmax, ymax)
```

```
        float halfHeight = height / 2.0f;
        float halfWidth = width / 2.0f;

        float[] pred_coor = new float[]{x - halfWidth, y - halfHeight, x + halfWidth, y + halfHeight};

        // (2) (xmin, ymin, xmax, ymax) -> (xmin_org, ymin_org, xmax_org, ymax_org)
        float resize_ratioW = 1.0f * intputSize / oriW;
        float resize_ratioH = 1.0f * intputSize / oriH;

        float resize_ratio = resize_ratioW > resize_ratioH ? resize_ratioH : resize_ratioW; //min

        float dw = (intputSize - resize_ratio * oriW) / 2;
        float dh = (intputSize - resize_ratio * oriH) / 2;

        pred_coor[0] = 1.0f * (pred_coor[0] - dw) / resize_ratio;
        pred_coor[2] = 1.0f * (pred_coor[2] - dw) / resize_ratio;

        pred_coor[1] = 1.0f * (pred_coor[1] - dh) / resize_ratio;
        pred_coor[3] = 1.0f * (pred_coor[3] - dh) / resize_ratio;

        // (3) clip some boxes those are out of range
        pred_coor[0] = pred_coor[0] > 0 ? pred_coor[0] : 0;
        pred_coor[1] = pred_coor[1] > 0 ? pred_coor[1] : 0;

        pred_coor[2] = pred_coor[2] < (oriW - 1) ? pred_coor[2] : (oriW - 1);
        pred_coor[3] = pred_coor[3] < (oriH - 1) ? pred_coor[3] : (oriH - 1);

        if ((pred_coor[0] > pred_coor[2]) || (pred_coor[1] > pred_coor[3])) {
            pred_coor[0] = 0;
            pred_coor[1] = 0;
            pred_coor[2] = 0;
            pred_coor[3] = 0;
        }

        // (4) discard some invalid boxes
        float temp1 = pred_coor[2] - pred_coor[0];
        float temp2 = pred_coor[3] - pred_coor[1];
        float temp = temp1 * temp2;
        if (temp < 0) {
            Log.e("checkInvalidateBox", "temp < 0");
            return false;
        }
        if (Math.sqrt(temp) > Float.MAX_VALUE) {
            Log.e("checkInvalidateBox", "temp max");
            return false;
        }

        return true;
    }
}
```

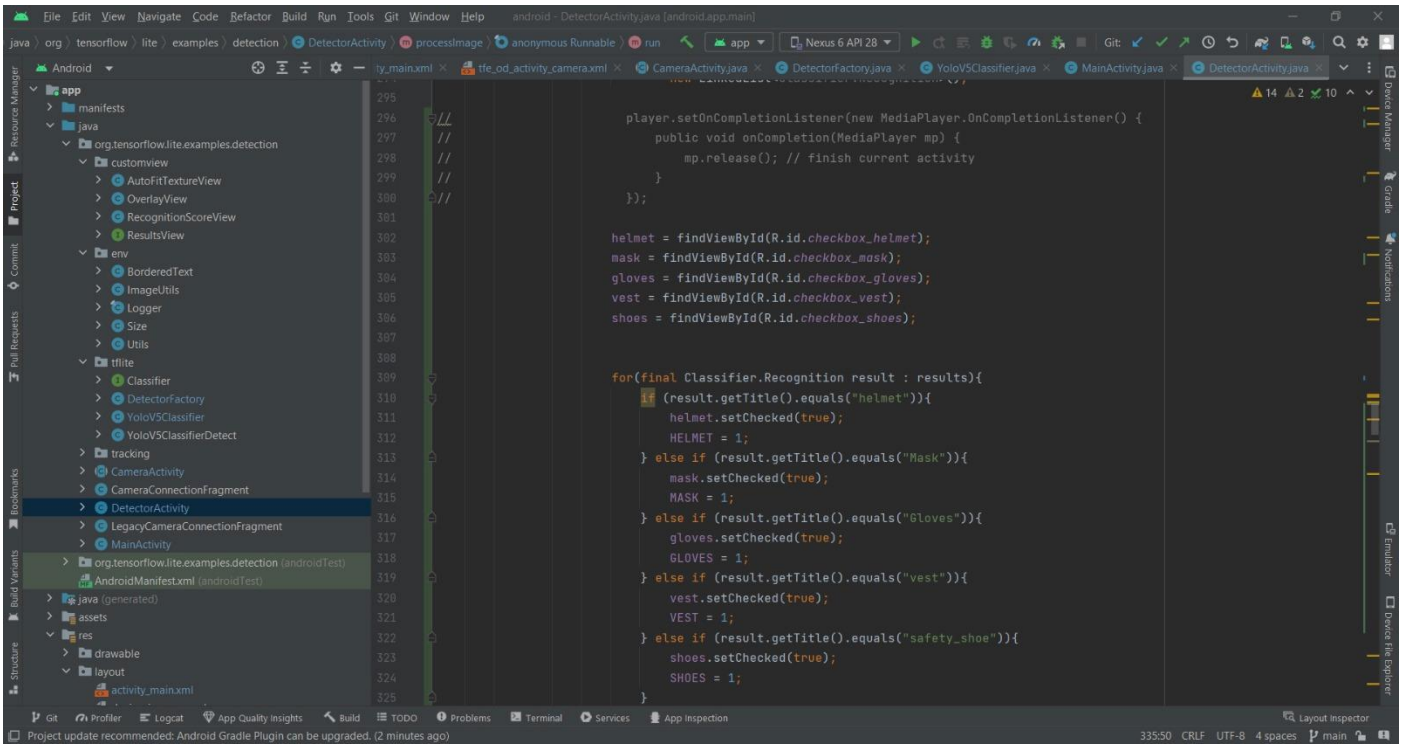
➔Java Android Application code



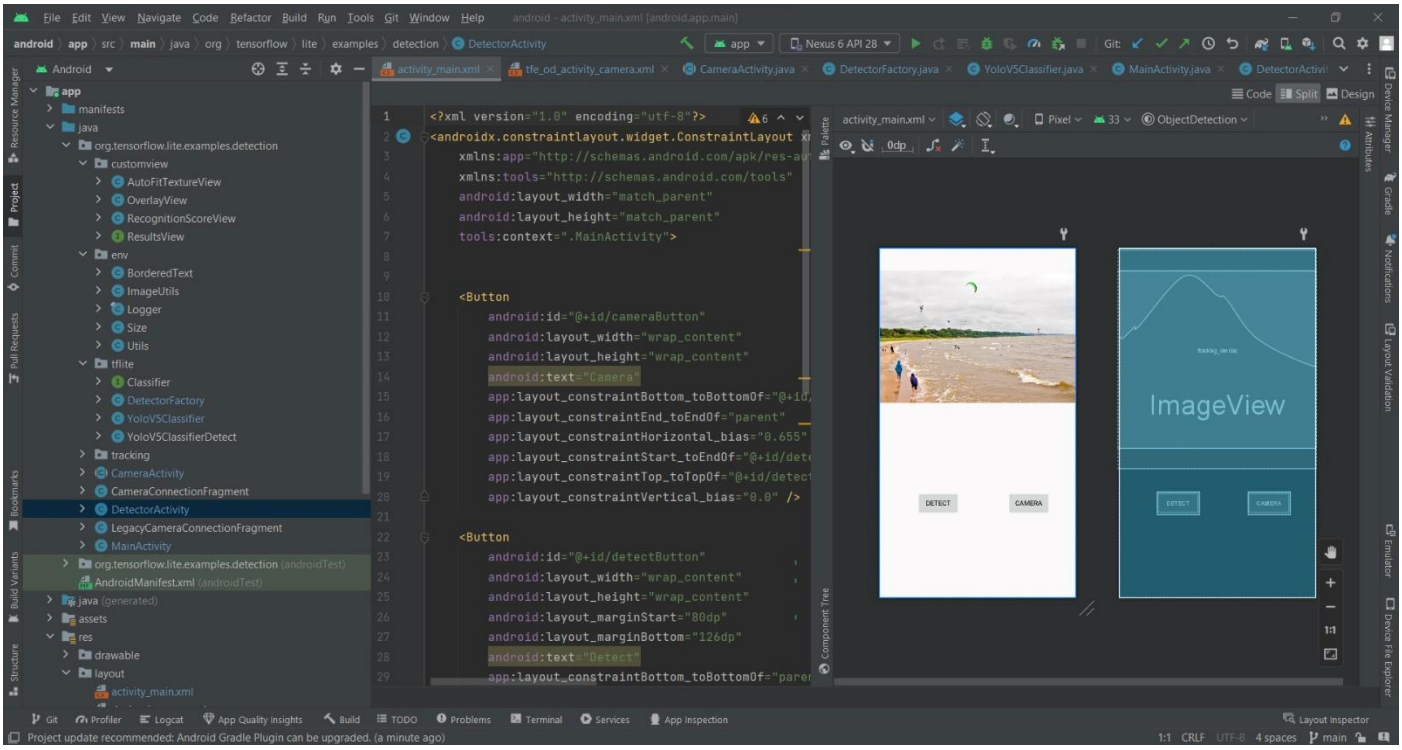
```
import org.tensorflow.lite.examples.detection.tracking.MultiBoxTracker;

import java.io.IOException;
import java.util.LinkedList;
import java.util.List;

8 usages  Aarohi *
public class MainActivity extends AppCompatActivity {

    3 usages
    public static final float MINIMUM_CONFIDENCE_TF_OD_API = 0.3f;

    Aarohi *
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

//        checkbox_meat = findViewById(R.id.checkbox_meat);
        cameraButton = findViewById(R.id.cameraButton);
        detectButton = findViewById(R.id.detectButton);
        imageView = findViewById(R.id.imageView);

        cameraButton.setOnClickListener(v -> startActivity(new Intent( packageContext: MainActivity.this, DetectorActivity.class)));

        detectButton.setOnClickListener(v -> {
            Handler handler = new Handler();

            new Thread(() -> {
                final List<Classifier.Recognition> results = detector.recognizeImage(cropBitmap);
                Aarohi
```



```
<?xml version="1.0" encoding="utf-8"?><!--
Copyright 2019 The TensorFlow Authors. All Rights

Licensed under the Apache License, Version 2.0 (th
you may not use this file except in compliance wit
You may obtain a copy of the License at

    http://www.apache.org/licenses/LICENSE-2.0

Unless required by applicable law or agreed to in
distributed under the License is distributed on an
WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, eith
See the License for the specific language governin
limitations under the License.
-->

<androidx.coordinatorlayout.widget.CoordinatorLayou
    xmlns:app="http://schemas.android.com/apk/res-a
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:background="#00000000">

    <RelativeLayout
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:background="@android:color/black"
        android:orientation="vertical">
```



```
//                    player.setOnCompletionListener(new MediaPlayer.OnCompletionListener() {
//                        public void onCompletion(MediaPlayer mp) {
//                            mp.release(); // finish current activity
//                        }
//                    });

                    helmet = findViewById(R.id.checkbox_helmet);
                    mask = findViewById(R.id.checkbox_mask);
                    gloves = findViewById(R.id.checkbox_gloves);
                    vest = findViewById(R.id.checkbox_vest);
                    shoes = findViewById(R.id.checkbox_shoes);


                    for(final Classifier.Recognition result : results){
                        if (result.getTitle().equals("helmet")){
                            helmet.setChecked(true);
                            HELMET = 1;
                        } else if (result.getTitle().equals("Mask")){
                            mask.setChecked(true);
                            MASK = 1;
                        } else if (result.getTitle().equals("Gloves")){
                            gloves.setChecked(true);
                            GLOVES = 1;
                        } else if (result.getTitle().equals("vest")){
                            vest.setChecked(true);
                            VEST = 1;
                        } else if (result.getTitle().equals("safety_shoe")){
                            shoes.setChecked(true);
                            SHOES = 1;
                        }
```



```
<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.ConstraintLayout
    xmlns:app="http://schemas.android.com/apk/res-a
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".MainActivity">


    <Button
        android:id="@+id/cameraButton"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Camera"
        app:layout_constraintBottom_toBottomOf="@+id/
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintHorizontal_bias="0.655"
        app:layout_constraintStart_toEndOf="@+id/dete
        app:layout_constraintTop_toTopOf="@+id/detect
        app:layout_constraintVertical_bias="0.0" />

    <Button
        android:id="@+id/detectButton"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_marginStart="80dp"
        android:layout_marginBottom="126dp"
        android:text="Detect"
        app:layout_constraintBottom_toBottomOf="parer
```