AUTOMATED HELMET DETECTION FOR MOTORCYCLIST USING DEEP LEARNING

A PROJECT REPORT

Submitted by

ASHIKA ANGEL J 422621104007

MONISHA R K 422621104027

VIJAYALAKSHMI R 422621104047

in partial fulfillment for the award of the degree

of

BACHELOR OF ENGINEERING

IN

COMPUTER SCIENCE AND ENGINEERING

UNIVERSITY COLLEGE OF ENGINEERING PANRUTI

ANNA UNIVERSITY : CHENNAI 600 025

MAY 2025

ANNA UNIVERSITY : CHENNAI 600 025

BONAFIDE CERTIFICATE

Certified that this project report **"AUTOMATED HELMET DETECTION FOR MOTORCYCLIST USING DEEP LEARNING"** is the bonafide work of **"ASHIKA ANGEL J (422621104007), MONISHA R K (422621104027), VIJAYALAKSHMI R (422621104047)"** who carried out the project work under my supervision.

SIGNATURE
Dr.D.Muruganandam,M.Tech.,Ph.D
HEAD OF DEPARTMENT
Computer Science and Engineering,
University College of Engineering,Panruti
Panruti-607106

SIGNATURE
Dr.A.Sasidhar,M.E.,Ph.D
SUPERVISOR
Computer Science and Engineering,
University College of Engineering,Panruti
Panruti-607106

Examination held on ……

INTERNAL EXAMINER **EXTERNAL EXAMINER**

ACKNOWLEDGEMENT

I would like to express my deepest gratitude to all those who contributed to the successful completion of my undergraduate project.

First and foremost, I extend my sincere thanks to Dean **Dr. S. Muthukumaran, M.E., Ph.D**., for providing the necessary resources and a conducive academic environment that enabled me to undertake this project.

I am immensely grateful to **Dr. D. Muruganandam, M.Tech., Ph.D.,** Head of the Department, Department of Computer Science and Engineering, University College of Engineering, Panruti, for their continuous support, encouragement, and valuable guidance throughout my academic journey.

My heartfelt appreciation goes to my project supervisor**, Dr. A. Sasidhar, M.E., Ph.D., TF/CSE,** University College of Engineering Panruti, for their expert guidance, insightful feedback, and unwavering patience. Their mentorship and constructive criticism were instrumental in shaping this project.

I would also like to thank all my professors and the technical staff of the department for their direct and indirect assistance during my work.

Lastly, I owe a special debt of gratitude to my family and friends for their unconditional love, motivation, and encouragement throughout my studies. Their belief in me kept me motivated during challenging times.

This accomplishment would not have been possible without the support of all these wonderful individuals.

ABSTRACT

Motorcycles serve as a prevalent mode of transportation. Nevertheless, riding a motorcycle entails significant risks, particularly when appropriate safety gear is not utilized. Helmets represent one of the most essential safety measures for individuals on two wheels, and failing to wear them can lead to serious injuries. At present, numerous researchers are concentrating on the detection of traffic violators; however, they have not yet succeeded in identifying the details of the individuals involved, as this necessitates the use of high-resolution cameras. This study categorizes the dataset into two groups: riders with helmets and riders without helmets. This paper introduces an automatic helmet detection system for motorcyclists utilizing deep learning techniques. To identify whether a rider is wearing a helmet, we employ the object detection algorithm known as YOLO. We have explored two versions of this algorithm, specifically YOLO v8 and v11. Both models were trained using a dataset comprising images of riders with and without helmets. Following the training process, we obtained the respective weights for each model. The dataset was gathered from real-time observations as well as online sources. We conducted a comparative analysis of both YOLO versions to determine which one achieved higher accuracy. The results indicated that YOLO v8 achieved an accuracy rate of 96%, while YOLO v11 reached 94%.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREVIATIONS

AHVDS	- AUTOMATIC HELMET VIOLATION DETECTION SYSTEM
CNN	- CONVOLUTIONAL NEURAL NETWORK
YOLO	- YOU ONLY LOOK ONCE
MLP	- MULTILAYER PERCEPTRON
GAN	- GENERATIVE ADVERSARIAL NETWORK
LLM	- LARGE LANGUAGE MODEL
OPENCV	- OPEN-SOURCE COMPUTER VISION
OCR	- OPTICAL CHARACTER RECOGNITION
R-CNN	- REGION-BASED CONVOLUTIONAL NEURAL NETWORK
CCTV	- CLOSED CIRCUIT TELEVISION
CSP	- CROSS STAGE PARTIAL
PANet	- PATH AGGREGATION METWORK
FPN	- FEATURE PYRAMID NETWORK
SPPF	- SPATIAL PYRAMID POOLING FAST
SSD	- SOLID STATE DRIVE
C2PSA	- CONVOLUTIONAL BLOCK WITH PARALLED SPATIAL ATTENTION

CHAPTER 1

INTRODUCTION

1.1 DEEP LEARNING

Deep learning is a type of machine learning that uses artificial neural networks to learn from data. Artificial neural networks are inspired by the human brain, and they can be used to solve a wide variety of problems, including image recognition, natural language processing, and speech recognition. Deep learning works by using artificial neural networks to learn from data. Neural networks are made up of layers of interconnected nodes, and each node is responsible for learning a specific feature of the data. Building on our previous example with images – in an image recognition network, the first layer of nodes might learn to identify edges, the second layer might learn to identify shapes, and the third layer might learn to identify objects. As the network learns, the weights on the connections between the nodes are adjusted so that the network can better classify the data. This process is called training, and it can be done using a variety of techniques, such as supervised learning, unsupervised learning, and reinforcement learning.

Once a neural network has been trained, it can be used to make predictions with new data it's received.

1.1.1 Deep Learning in Image Processing

Deep learning is revolutionizing image processing. It uses Convolutional Neural Networks (CNNs) to process image data more efficiently than traditional Multi-Layer Perceptrons (MLP).

a. Image Classification

Use of CNNs for Image Classification Tasks. Convolutional Neural Networks (CNNs) have revolutionized image classification by their ability to automatically learn and extract features from images. CNNs process visual data through multiple layers, each layer extracting increasingly complex features from the image. This hierarchical feature extraction makes CNNs highly effective for classifying images into predefined categories.

b. Image Segmentation

Semantic Segmentation: This technique involves classifying each pixel in an image into a category, such as identifying different parts of an object or distinguishing between various objects within the same image. Semantic segmentation is crucial for understanding the structure and content of images.

Instance Segmentation: Building on semantic segmentation, instance segmentation identifies and segments each object instance separately. This allows for distinguishing between multiple objects of the same category in a single image.

c. Image Generation and Enhancement

Generative Adversarial Networks (GANs) are used to generate highly realistic images by training two neural networks the generator and the discriminator in a competitive setting. The generator creates fake images, while the discriminator tries to distinguish between real and fake images. Over time, the generator becomes proficient at producing images that are indistinguishable from real ones.

d. Anomaly Detection in Images

Deep learning models, particularly autoencoders and CNNs, can be trained to detect anomalies in images. By learning the normal patterns in training data, these models can identify deviations or unusual patterns in new data, signaling potential anomalies.

e. Cross-modal Applications of LLMs

Image Captioning Large Language Models (LLMs) like GPT-3 and GPT-4 are employed to generate descriptive text for images. By understanding the context and content of an image, these models can produce accurate and meaningful captions, making images more accessible and searchable.

1.1.2 Working of Deep Learning

Neural networks, or artificial neural networks, attempt to mimic the human brain through a combination of data inputs, weights and bias, all acting as silicon neurons. These elements work together to accurately recognize, classify and describe objects within the data. Deep neural networks consist of multiple layers of interconnected nodes, each building on the previous layer to refine and optimize the prediction or categorization. This progression of computations through the network is called forward propagation. The input and output layers of a deep neural network are called visible layers. The input layer is where the deep learning model ingests the data for processing, and the output layer is where the final prediction or classification is made.

Another process called backpropagation uses algorithms, such as gradient descent, to calculate errors in predictions, and then adjusts the weights and biases of the function by moving backwards through the layers to train the model. Together, forward propagation and backpropagation enable a neural network to make predictions and correct for any errors. Over time, the algorithm

becomes gradually more accurate. Deep learning requires a tremendous amount of computing power. High-performance graphical processing units (GPUs) are ideal because they can handle a large volume of calculations in multiple cores with copious memory available. Distributed cloud computing might also assist. This level of computing power is necessary to train deep algorithms through deep learning. However, managing multiple GPUs on premises can create a large demand on internal resources and be incredibly costly to scale. For software requirements, most deep learning apps are coded with one of these three learning frameworks: JAX, PyTorch or TensorFlow.

1.1.3 Types of Deep Learning

Deep learning algorithms are incredibly complex, and there are different types of neural networks to address specific problems or datasets. Here are six. Each has its own advantages and they are presented here roughly in the order of their development, with each successive model adjusting to overcome a weakness in a previous model.

One potential weakness across them all is that deep learning models are often “black boxes,” making it difficult to understand their inner workings and posing interpretability challenges. But this can be balanced against the overall benefits of high accuracy and scalability.

- **CNNs**

  Convolutional neural networks (CNNs or ConvNets) are used primarily in computer vision and image classification applications. They can detect features and patterns within images and videos, enabling tasks such as object detection, image recognition, pattern recognition and face recognition. These networks harness principles from linear algebra, particularly matrix multiplication, to identify

patterns within an image. CNNs are a specific type of neural network, which is composed of node layers, containing an input layer, one or more hidden layers and an output layer. Each node connects to another and has an associated weight and threshold. If the output of any individual node is above the specified threshold value, that node is activated, sending data to the next layer of the network. Otherwise, no data is passed along to the next layer of the network.

- **RNNs**

  Recurrent neural networks (RNNs) are typically used in natural language and speech recognition applications as they use sequential or time-series data. RNNs can be identified by their feedback loops. These learning algorithms are primarily used when using time-series data to make predictions about future outcomes. Use cases include stock market predictions or sales forecasting, or ordinal or temporal problems, such as language translation, natural language processing (NLP), speech recognition and image captioning. These functions are often incorporated into popular applications such as Siri, voice search and Google Translate.

  RNNs use their "memory" as they take information from prior inputs to influence the current input and output. While traditional deep neural networks assume that inputs and outputs are independent of each other, the output of RNNs depends on the prior elements within the sequence. While future events would also be helpful in determining the output of a given sequence, unidirectional recurrent neural networks cannot account for these events in their predictions.

  RNNs share parameters across each layer of the network and share the same weight parameter within each layer of the network,

with the weights adjusted through the processes of backpropagation and gradient descent to facilitate reinforcement learning.

- **Autoencoders and variational autoencoders**

  Deep learning made it possible to move beyond the analysis of numerical data, by adding the analysis of images, speech and other complex data types. Among the first class of models to achieve this were variational autoencoders (VAEs). They were the first deep-learning models to be widely used for generating realistic images and speech, which empowered deep generative modeling by making models easier to scale, which is the cornerstone of what we think of as generative AI.

  Autoencoders work by encoding unlabeled data into a compressed representation, and then decoding the data back into its original form. Plain autoencoders were used for a variety of purposes, including reconstructing corrupted or blurry images. Variational autoencoders added the critical ability not just to reconstruct data, but also to output variations on the original data. This ability to generate novel data ignited a rapid-fire succession of new technologies, from generative adversarial networks (GANs) to diffusion models, capable of producing ever more realistic but fake images. In this way, VAEs set the stage for today's generative AI.

  Autoencoders are built out of blocks of encoders and decoders, an architecture that also underpins today's large language models. Encoders compress a dataset into a dense representation, arranging similar data points closer together in an abstract space. Decoders sample from this space to create something new while preserving the dataset's most important features.

- **GANs**

  Generative adversarial networks (GANs) are neural networks that are used both in and outside of artificial intelligence (AI) to create new data resembling the original training data. These can include images appearing to be human faces but are generated, not taken of real people. The “adversarial” part of the name comes from the back-and-forth between the two portions of the GAN: a generator and a discriminator. The generator creates something: images, video or audio and then producing an output with a twist. For example, a horse can be transformed into a zebra with some degree of accuracy. The result depends on the input and how well-trained the layers are in the generative model for this use case. The discriminator is the adversary, where the generative result (fake image) is compared against the real images in the dataset. The discriminator tries to distinguish between the real and fake images, video or audio. GANs train themselves. The generator creates fakes while the discriminator learns to spot the differences between the generator's fakes and the true examples. When the discriminator is able to flag the fake, then the generator is penalized. The feedback loop continues until the generator succeeds in producing output that the discriminator cannot distinguish.

- **Diffusion models**

  Diffusion models are generative models that are trained using the forward and reverse diffusion process of progressive noise-addition and denoising. Diffusion models generate data, most often images similar to the data on which they are trained, but then overwrite the data used to train them. They gradually add Gaussian noise to the training data until it’s unrecognizable, then learn a reversed “denoising” process that can synthesize output (usually

images) from random noise input. A diffusion model learns to minimize the differences of the generated samples versus the desired target. Any discrepancy is quantified and the model's parameters are updated to minimize the loss training the model to produce samples closely resembling the authentic training data.

Beyond image quality, diffusion models have the advantage of not requiring adversarial training, which speeds the learning process and also offering close process control. Training is more stable than with GANs and diffusion models are not as prone to mode collapse. But, compared to GANs, diffusion models can require more computing resources to train, including more fine-tuning.

- **Transformer models**

  Transformer models combine an encoder-decoder architecture with a text-processing mechanism and have revolutionized how language models are trained. An encoder converts raw, unannotated text into representations known as embeddings; the decoder takes these embeddings together with previous outputs of the model, and successively predicts each word in a sentence. Using fill-in-the-blank guessing, the encoder learns how words and sentences relate to each other, building up a powerful representation of language without having to label parts of speech and other grammatical features. Transformers, in fact, can be pretrained at the outset without a particular task in mind. After these powerful representations are learned, the models can later be specialized with much less data to perform a requested task.

  Several innovations make this possible. Transformers process words in a sentence simultaneously, enabling text processing in parallel, speeding up training. Earlier techniques including recurrent neural networks (RNNs) processed words one by one. Transformers

also learned the positions of words and their relationships, this context enables them to infer meaning and disambiguate words such as “it” in long sentences.

1.1.4 Applications of Deep Learning

Deep learning has numerous applications across various fields:

- **Computer Vision:** Used for image classification, object detection, and image segmentation.
- **Natural Language Processing (NLP):** Enables tasks like language translation, sentiment analysis, and speech recognition.
- **Healthcare:** Supports medical imaging specialists in analyzing and assessing images.
- **Finance:** Used for predictive analytics, fraud detection, and managing credit and investment portfolios.
- **Customer Service:** Powers chatbots and virtual assistants to provide personalized user experiences.

1.2 YOU ONLY LOOK ONCE(YOLO)

YOLO is a real-time object detection algorithm that can identify objects in an image and provide their bounding boxes and class probabilities in a single forward pass of a neural network. The original paper of YOLO (“You Only Look Once”) was published in 2016 by Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi. Ultralytics has developed an open-source implementation of the YOLO algorithm which has gained popularity in the computer vision community and is used in various applications. YOLO is based on the idea of segmenting an image into smaller images. The image is split into a square grid of dimensions S×S. The cell in which the center of an object, for instance, the center of the dog, resides, is the cell responsible for detecting that object. Each cell will predict B bounding boxes and a confidence score for each

box. The default for this architecture is for the model to predict two bounding boxes.

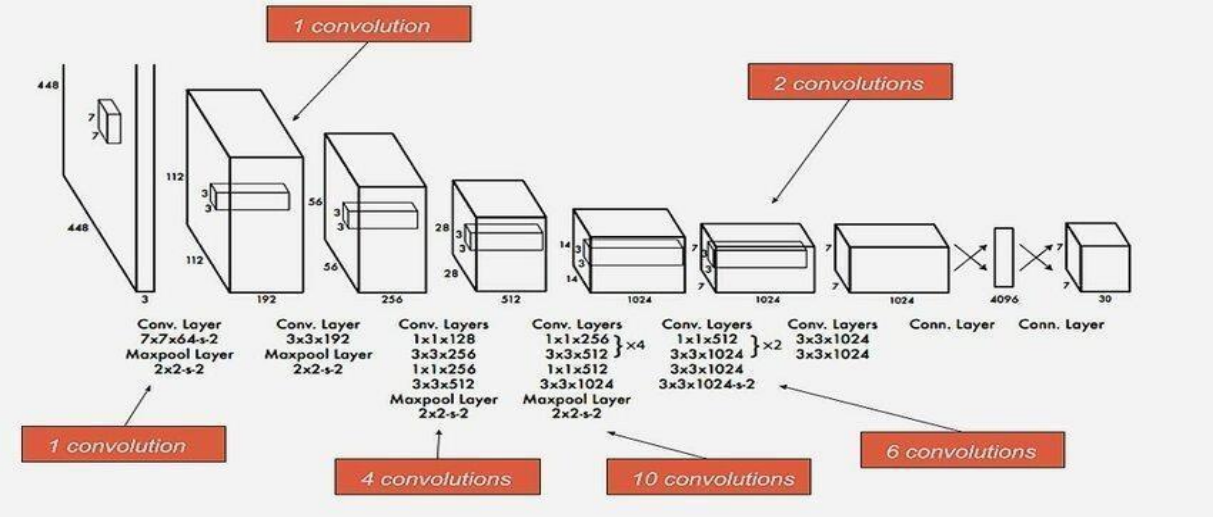

Figure 1.1 YOLO architecture

The YOLO model is made up of three key components: the head, neck, and backbone. The backbone is the part of the network made up of convolutional layers to detect key features of an image and process them. The backbone is first trained on a classification dataset, such as ImageNet, and typically trained at a lower resolution than the final detection model, as detection requires finer details than classification. The neck uses the features from the convolution layers in the backbone with fully connected layers to make predictions on probabilities and bounding box coordinates. The head is the final output layer of the network which can be interchanged with other layers with the same input shape for transfer learning. Some key features of yolo are

- Speed
- Detection accuracy
- Good generalization
- Open-source

CHAPTER 2

LITERATURE SURVEY

2.1 COMPREHENSIVE STUDY ON THE DEVELOPMENT OF AN AUTOMATIC HELMET VIOLATOR DETECTION SYSTEM (AHVDS) USING ADVANCED MACHINE LEARNING TECHNIQUES

Author: M. Saravanan, G.K. Rajini

Year: 2022

Many people still ride two-wheelers without wearing helmets, which leads to serious accidents and deaths. The government wants to catch these violators, but doing it manually is hard and expensive. This study builds an automatic system that detects if someone is not wearing a helmet while riding. It also reads the number plate of the violator and sends them an e-challan (fine) automatically. The system works in three steps: First, it finds the bike and rider using a camera. Second, it checks whether the rider and passenger are wearing helmets. Third, if not, it finds the vehicle number plate and extracts the number. The system uses deep learning and machine learning models like YOLO, Mask R-CNN, DenseNet-121, ResNet-101, and Inception ResNet V2. YOLO helps to find bikes in live traffic videos. Mask R-CNN helps to segment (cut out) the bike, rider, and number plate from the image. Fuzzy logic is used to decide if the rider or passenger is a violator. OCR (Optical Character Recognition) is used to read the number plate characters. Once the number is identified, it is sent to the government database, and an e-challan is sent to the violator's phone. The system was trained using real traffic videos from Tamil Nadu, India, and it worked well. The model was tested using over 1.4 million video frames and showed high accuracy and fast results. The system performs better than many older models and uses less time and memory. It is cheaper than using special helmet sensors or manual checks by police.

Advantages

- It detects helmet violators and reads number plates automatically.
- Works fast in real-time using AI and cameras without expensive tools.
- Helps send fines (e-challans) quickly and reduces manual police work.

Disadvantages

- Needs clear, high-quality video to work properly.
- May not detect violators correctly if view is blocked or poor lighting.
- Privacy and setup with government systems could be challenging.

2.2 HELMET DETECTION USING MACHINE LEARNING

Author: Chaitanya Srusti, Vibhav Deo, Dr. Rupesh C. Jaiswal

Year: 2022

The paper proposes a smart, real-time helmet detection system using machine learning to address the widespread issue of motorcycle riders neglecting helmet use, particularly in countries like India where two-wheelers are the most common form of transport. The authors highlight the high rate of motorcycle-related accidents and advocate for an automated solution to reduce dependence on traffic police. The system utilizes video footage from CCTV cameras, which is processed to extract frames containing riders. These frames are manually labelled and split into training and testing datasets. A pre-trained YOLO model (most likely YOLOv2 or YOLOv3) is then fine-tuned using the DarkFlow framework, allowing the team to leverage transfer learning for better efficiency and lower computational cost. The model is capable of detecting riders with and without helmets by analysing the video in real-time using the OpenCV library in Python. The trained model achieved a strong 96% accuracy

with an average detection time of 1.35 seconds, making it suitable for practical deployment. Additionally, the system ignores non-relevant elements like pedestrians, focusing only on motorcyclists. The paper concludes that the model could serve as a foundation for further innovations like integrating drone-based surveillance or automatic fine generation, ultimately contributing to improved road safety and efficient traffic law enforcement**.**

Advantages

- High accuracy (96%) and fast performance make it effective for real-time monitoring.
- Cost-effective and scalable using open-source tools and transfer learning.

Disadvantages

- May not perform well in poor lighting, low-resolution footage, or with non-standard helmets.
- No integration with license plate detection limits automatic penalty enforcement.

2.3 CNN-BASED AUTOMATIC HELMET VIOLATION DETECTION OF MOTORCYCLIST FOR AN INTELLIGENT TRANSPORTATION SYSTEM

Author: Tasbeeha Waris, Muhammad Asif, Sadia Zafar.

Year: 2022

The research paper focuses on developing a deep learning-based system for automatically detecting helmet violations by motorcyclists, which is a crucial application within Intelligent Transportation Systems (ITS). The system uses a type of artificial intelligence called a Convolutional Neural Network (CNN),

specifically a model known as Faster R-CNN, to analyse real-time video footage from surveillance cameras. This model is capable of both identifying the presence of motorcyclists and determining whether they are wearing helmets by locating and classifying the objects within each video frame. To train the model, the researchers collected and prepared a large dataset that includes images of riders with and without helmets, sourced from both public repositories and self-captured videos from various locations in Pakistan. They applied preprocessing techniques to clean the data and used annotation tools to label each image accurately. The result is a system that achieved an impressive 97.69% accuracy, proving its reliability in real-world scenarios. This approach not only reduces the burden on traffic authorities but also enhances road safety by promoting law enforcement and potentially reducing fatal road accidents caused by helmet negligence.

Advantages

- Highly accurate (97.69%) and effective in real-time detection of helmet violations using deep learning.
- Capable of distinguishing between helmets and other head coverings, reducing false positives.

Disadvantages

- Requires significant computational resources for training and deployment.
- Current scope is limited to helmet detection; other violations like number plate issues are not included yet.

2.4 REAL-TIME AUTOMATIC HELMET DETECTION OF MOTORCYCLISTS USING IMPROVED YOLOV5 DETECTOR

Authors: Wei Jia, Shiquan Xu, Zhen Liang, Yang Zhao, Hai Min, Shujie Li, Ye Yu

Year: 2021

It is a deep learning-based solution for detecting whether motorcyclists are wearing helmets in urban traffic environments. The system is based on an improved version of the YOLOv5 object detection model, which is known for its speed and accuracy. To enhance its performance further, the authors integrated two advanced techniques: triplet attention mechanisms, which help the model focus on relevant features in complex traffic scenes, and Soft Non-Maximum Suppression (Soft-NMS), which improves the detection of overlapping objects. To train and evaluate their model, the researchers introduced a new dataset called HFUT-MH, which includes images from various urban traffic conditions, ensuring the system's robustness across different scenarios. The enhanced YOLOv5 model achieved a mean Average Precision (mAP) of 97.7% and an F1-score of 92.7%, indicating a high level of accuracy in detecting helmet use. Moreover, the system processes video at 63 frames per second (FPS), making it viable for real-time applications. This technology can be used in traffic surveillance to automatically detect violations, promote helmet use, and ultimately contribute to road safety and accident prevention.

Advantages

- High detection accuracy (mAP of 97.7%, F1-score of 92.7%)
- Real-time processing capability (63 FPS)
- Improved model with triplet attention and Soft-NMS

Disadvantages

- Performance may degrade under adverse weather conditions (rain, fog)
- Accuracy decreases in low-light or nighttime conditions
- Detection may be hindered by partial occlusions (e.g., passengers)

2.5 DEEP LEARNING-BASED HELMET WEAR ANALYSIS OF MOTORCYCLE RIDER

Authors: B. Yogameena, K. Menaka, S. Saravana Perumal

Year: 2019

The 2019 paper introduces an intelligent video surveillance system designed to automatically detect whether motorcyclists are wearing helmets, using deep learning and computer vision techniques. The system is structured in multiple stages to ensure accurate detection in real-world traffic conditions. First, Gaussian Mixture Models (GMM) are used for motion detection, allowing the system to isolate moving objects, such as motorcycles, from the background. Once motion is detected, the system employs Faster R-CNN, a powerful deep learning-based object detector, to identify motorcycles and detect whether the riders are wearing helmets. In addition to helmet detection, the system also integrates a Convolutional Neural Network (CNN) combined with a Spatial Transformer Network (STN) to perform license plate recognition. This enables authorities to identify violators and take necessary action. The proposed approach was tested on six datasets collected from different cities, which included a variety of challenging real-world scenarios such as different lighting conditions, traffic densities, and occlusions. Despite these challenges, the

system demonstrated strong performance, proving its effectiveness and adaptability in diverse urban environments. This solution aids in traffic law enforcement and promotes safer riding habits.

Advantages

- Integrated system for detection and recognition
- Enables automatic penalty enforcement through license plate recognition
- High accuracy in favourable conditions (up to 95% for detection, 92% for recognition)

Disadvantages

- Lower accuracy on low-resolution or blurry footage
- Computationally heavy (Faster R-CNN models)
- Dependent on pre-trained models (e.g., VGG-16)

2.6 HELMET USE DETECTION OF TRACKED MOTORCYCLES USING CNN-BASED MULTI-TASK LEARNING

Authors: Hanhe Lin, Jeremiah D. Deng, Deike Albers, Felix Wilhelm Siebert

Year: 2020

The 2020 IEEE Access paper proposes a deep learning-based system for detecting helmet usage among motorcyclists using video surveillance footage. The system is built on a Convolutional Neural Network (CNN) with a

Multi-Task Learning (MTL) framework, allowing it to perform two critical tasks simultaneously: motorcycle tracking and helmet use classification. This dual-task approach helps the model learn shared features more effectively, improving both accuracy and efficiency.To support the development and evaluation of the model, the authors introduced the HELMET dataset, a large-scale dataset containing over 91,000 annotated video frames and tracking data for 10,006 motorcycles across 12 diverse urban locations in Myanmar. This dataset captures a wide variety of real-world traffic conditions, including different lighting, camera angles, and rider behaviors.The model achieved a 67.3% F-measure, indicating a reasonable balance between precision and recall in helmet detection, while maintaining real-time processing speeds exceeding 8 frames per second (FPS). Though not as fast as other models, it is sufficient for many practical applications. This system can be useful for traffic monitoring, promoting road safety, and enforcing helmet laws in busy urban environments.

Advantages

- Real-time performance on consumer GPUs (>8 FPS).
- Multi-task learning improves efficiency.
- Works well in urban traffic environments- Compatible with existing CCTV systems.

Disadvantages

- Performance drops under poor lighting or weather.
- May require retraining for use in different regions.
- Difficulty detecting multiple riders or occluded helmets.

CHAPTER 3

SYSTEM ANALYSIS

3.1 EXISTING SYSTEM

The surveyed studies consistently demonstrate the powerful application of deep learning, particularly Convolutional Neural Networks (CNNs) such as YOLO and Faster R-CNN, as the foundational technology for automatic helmet violator detection systems (AHVDS). These CNN models are adept at both precisely locating (object detection of motorcycles, riders, helmets, and number plates) and categorizing (classification of helmet presence or absence) crucial elements within traffic video feeds. To further enhance the capabilities of these systems, some researchers integrate supplementary techniques like Mask R-CNN for detailed pixel-level segmentation of objects, Fuzzy Logic for making nuanced violation decisions based on various factors, and Optical Character Recognition (OCR) to automatically extract alphanumeric characters from vehicle number plates. The development and evaluation of these sophisticated AI models heavily rely on comprehensive datasets, often meticulously collected from real-world traffic scenarios, ensuring the systems are trained on the complexities they will encounter in deployment. The reported performance metrics are notably high, with some advanced models achieving accuracy rates exceeding 97%, coupled with increasingly efficient processing speeds that pave the way for feasible real- time applications in traffic surveillance. While the primary focus remains the accurate detection of helmet violations, a significant advancement lies in the integration of number plate recognition, which enables the potential for fully automated e-challan generation, streamlining the enforcement process and contributing to improved road safety.

3.1.1 Disadvantages of Existing System

- Relies on Good Quality: Performance suffers significantly with poor lighting, low resolution, and obstructions.
- Struggles with Complex Scenarios: Detecting helmets accurately is challenging with occlusions or multiple riders.
- Demands High Computational Resources: Training and deployment require significant processing power.
- Faces Integration and Privacy Issues: Linking with government systems for fines raises privacy concerns and technical hurdles.
- Limited to Helmet Detection: Most current systems don't address other traffic violations.

3.2 PROPOSED SYSTEM

In this proposed study, we have developed a two-wheeler helmet detection system utilizing deep learning techniques. This project employs the YOLO object detection algorithm, specifically versions 8 and 11, to determine whether a rider is wearing a helmet. We gathered a dataset for real-time analysis, sourcing images from both real-time observations and online platforms. The images were then trained using both YOLO v8 and v11, with weights loaded during the training process. Subsequently, test images were evaluated to ascertain if the rider was wearing a helmet or not. After testing the image set, we utilized a confusion matrix to assess the accuracy of both models and identify which one produced superior results. This proposed system offers a technological solution for a significant real-time problem, with the potential for increased efficiency,

accuracy, and scalability compared to traditional methods of monitoring helmet usage.

3.2.1 Advantages of Proposed System

- **High Accuracy Potential**

  Deep learning models like YOLO, when trained on a diverse and well-annotated dataset, can achieve high accuracy in object detection tasks.

- **Automated and Objective Monitoring**

  The system provides an automated and impartial method for monitoring helmet usage, minimizing dependence on manual observation that may be susceptible to human error and inconsistencies.

- **Data-Driven Insights**

  The system is capable of producing significant data regarding helmet usage trends over time and in various locations.

CHAPTER 4

SYSTEM REQUIREMENTS

4.1 HARDWARE REQUIREMENTS

- **System :** Intel Processor I5 and Ryzen 5
- **Hard Disk :** 512 GB
- **RAM :** 16 GB

4.2 SOFTWARE REQUIREMENTS

- **Operating System:** Windows 11
- **Coding Language:** Python
- **Software :** Visual studio code, Jupyter Notebook

CHAPTER 5

SYSTEM DESIGN

5.1 YOU ONLY LOOK ONCE VERSION 8

5.1.1 Working of YOLOv8

YOLOv8 divides an image into a grid and predicts bounding boxes and class probabilities for each grid cell. Its backbone network extracts features, which are then fused in the neck to understand objects at different scales. The head uses these features to directly predict the bounding boxes and object classes without predefined anchors.

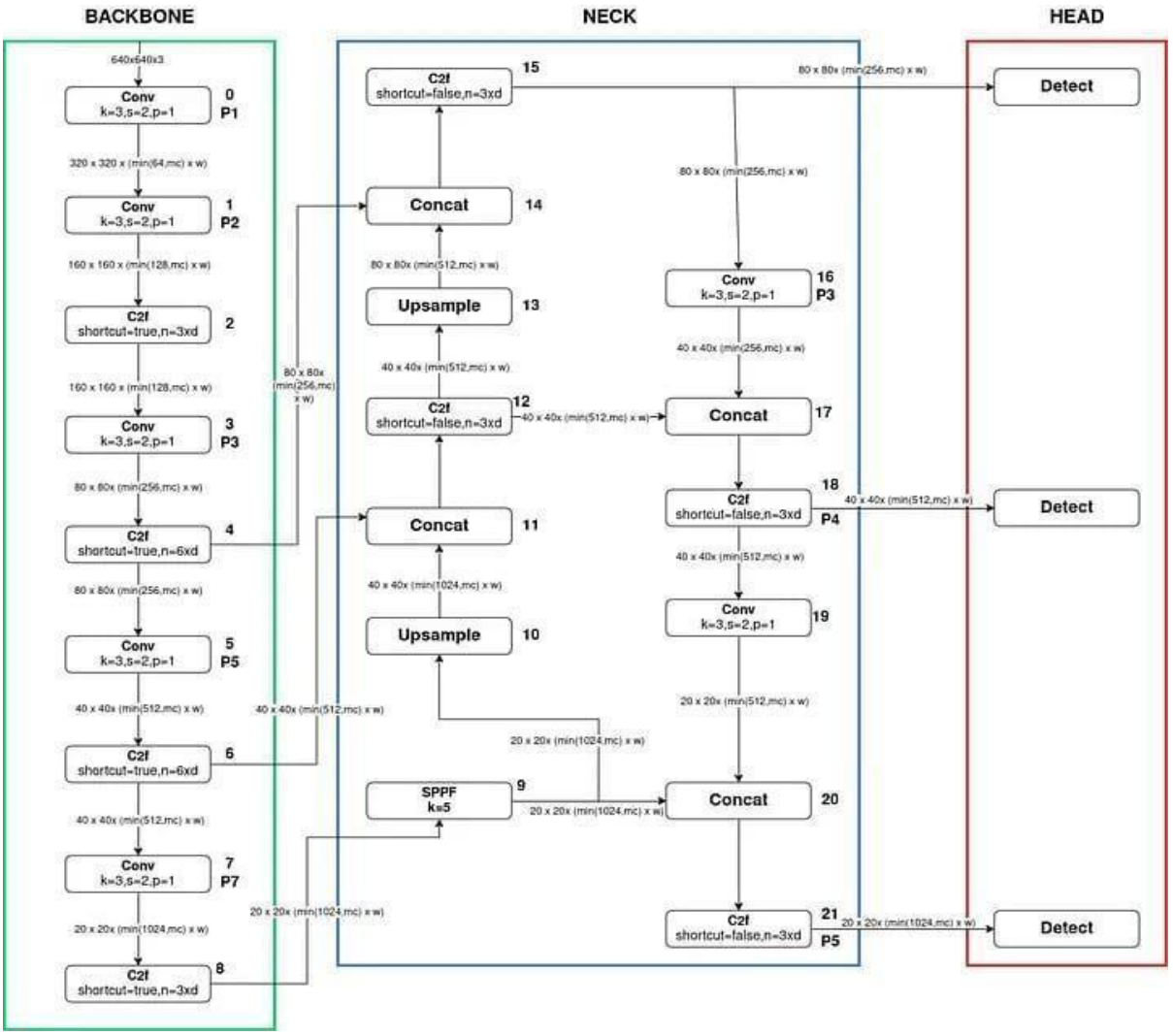

Figure 5.1 System Architecture of YOLOv8

5.1.1.1. Backbone

The backbone's primary goal is to transform the raw input image into a hierarchical set of feature maps, where earlier layers capture low-level details (edges, corners, textures) and deeper layers capture high-level semantic information (object parts, object categories).

1. **Initial Stem Layer:** The first convolutional layer with a kernel size of 3x3 and a stride of 2 acts as the "stem" of the network. This initial down sampling is crucial for reducing the computational load in the subsequent layers. The 3x3 kernel is effective at capturing local patterns.
2. **CSPDarknet Evolution and C2f Module:** YOLOv8's backbone is a refined version of CSPDarknet. The **Cross Stage Partial (CSP)** strategy is key here. CSP divides the feature map of a layer into two parts: one part goes through a series of convolutional blocks, while the other part is directly passed through and then merged (concatenated) with the processed part at the end. This design choice has several benefits:
    a. **Richer Gradients:** It alleviates the vanishing gradient problem, allowing for the training of deeper networks.
    b. **Reduced Computation:** By processing only a part of the feature map through the dense blocks, it reduces the computational cost and inference time.
    c. **Feature Diversity:** The merging of processed and unprocessed information encourages the network to learn more diverse features.
3. The **C2f module** is a central building block in the YOLOv8 backbone. It's an improvement over the C3 module used in YOLOv5. While the exact internal structure might have variations, its core principle involves using multiple bottleneck layers with residual connections arranged in a way that

maximizes gradient flow and feature integration. The n parameter in the diagram indicates the number of these bottleneck repetitions within the C2f module. The shortcut parameter controls whether a direct skip connection is added across the module.

4. **Progressive Downsampling:** The convolutional layers with a stride of 2 progressively reduce the spatial dimensions of the feature maps, creating a feature pyramid. Each level of this pyramid captures information at a different scale. The output feature maps from different stages of the backbone (P1 to P5) are then fed into the neck.

5.1.1.2. Neck

The neck plays a crucial role in aggregating these multi-scale features from the backbone. Objects in an image can vary significantly in size, and the neck ensures that the detector has access to appropriate feature resolutions for detecting both small and large objects effectively. YOLOv8 primarily uses a **PANet (Path Aggregation Network)** or a similar architecture in its neck.

1. **Feature Pyramid Network (FPN) Component:** The top-down pathway (starting from the deeper, semantically rich feature maps) is similar to FPN. Higher-level feature maps are upsampled and then combined (via concatenation) with lower-level feature maps. This allows the higher-level semantic information to enhance the lower-level, more spatially precise features.

2. **Bottom-Up Path Augmentation:** PANet adds a bottom-up pathway that starts from the lower-level features and goes up. These lower-level features are passed through convolutional layers and combined (concatenated) with the processed higher-level features. This shorter path helps in propagating accurate localization information from the lower layers to the higher layers.

3. **SPPF (Spatial Pyramid Pooling Fast):** The SPPF module is strategically placed in the neck to process the deepest feature map from the backbone. By applying parallel max pooling with different kernel sizes (e.g., 5x5 as shown), it extracts multi-scale spatial context without significantly increasing computation. The outputs of these parallel pooling operations are then concatenated, providing a fixed-size output regardless of the input feature map size, and enhancing robustness to object scale variations.
4. **C2f Modules in the Neck:** Just like in the backbone, C2f modules are used within the neck to efficiently process and fuse the feature maps as they are passed through the different pathways of the PANet.

5.1.1.3. Head

The head is where the magic of object detection happens – predicting the bounding boxes, objectness scores, and class probabilities. YOLOv8 employs a **decoupled and anchor-free** approach.

1. **Decoupled Head:** This is a significant departure from earlier YOLO versions that had a coupled head predicting all three outputs from the same set of convolutional layers. In a decoupled head, the feature maps from the neck are fed into separate branches: one for predicting the class probabilities and objectness score, and another for predicting the bounding box coordinates (center x, center y, width, height). This decoupling often leads to improved accuracy as the optimization for classification and localization can happen independently.
2. **Anchor-Free Detection:** Unlike previous YOLO versions that relied on a set of predefined anchor boxes, YOLOv8's anchor-free head directly predicts the bounding box parameters relative to each grid cell in the final feature maps from the neck. This simplifies the architecture and reduces

the number of hyperparameters that need to be tuned. For each grid cell, the head predicts:

a. The probability of an object being present (objectness score).

b. The probability distribution over the different object classes.

c. The offset of the object's center from the top-left corner of the grid cell, and the width and height of the bounding box relative to the size of the grid cell.

3. **Prediction at Multiple Scales (P3, P4, P5):** The head receives feature maps from three different levels of the neck (corresponding to different spatial resolutions: 80x80, 40x40, 20x20 for a 640x640 input). This allows the detector to effectively detect objects of varying sizes. The higher-resolution feature maps (e.g., 80x80) are better at detecting smaller objects, while the lower-resolution maps (e.g., 20x20) are more suitable for larger objects.

5.2 YOU ONLY LOOK ONCE VERSION 11

5.2.1 Working of YOLOv11

YOLOv11 builds upon the foundation of YOLOv8 by introducing refinements aimed at improving both efficiency and accuracy. A key enhancement is the integration of attention mechanisms, allowing the model to selectively focus on the most salient regions within the input, thereby improving its ability to discern and localize objects. This targeted attention contributes to potentially better performance in both object detection and instance segmentation tasks. Ultimately, YOLOv11 represents a step forward in creating a more efficient and perceptive architecture for computer vision.

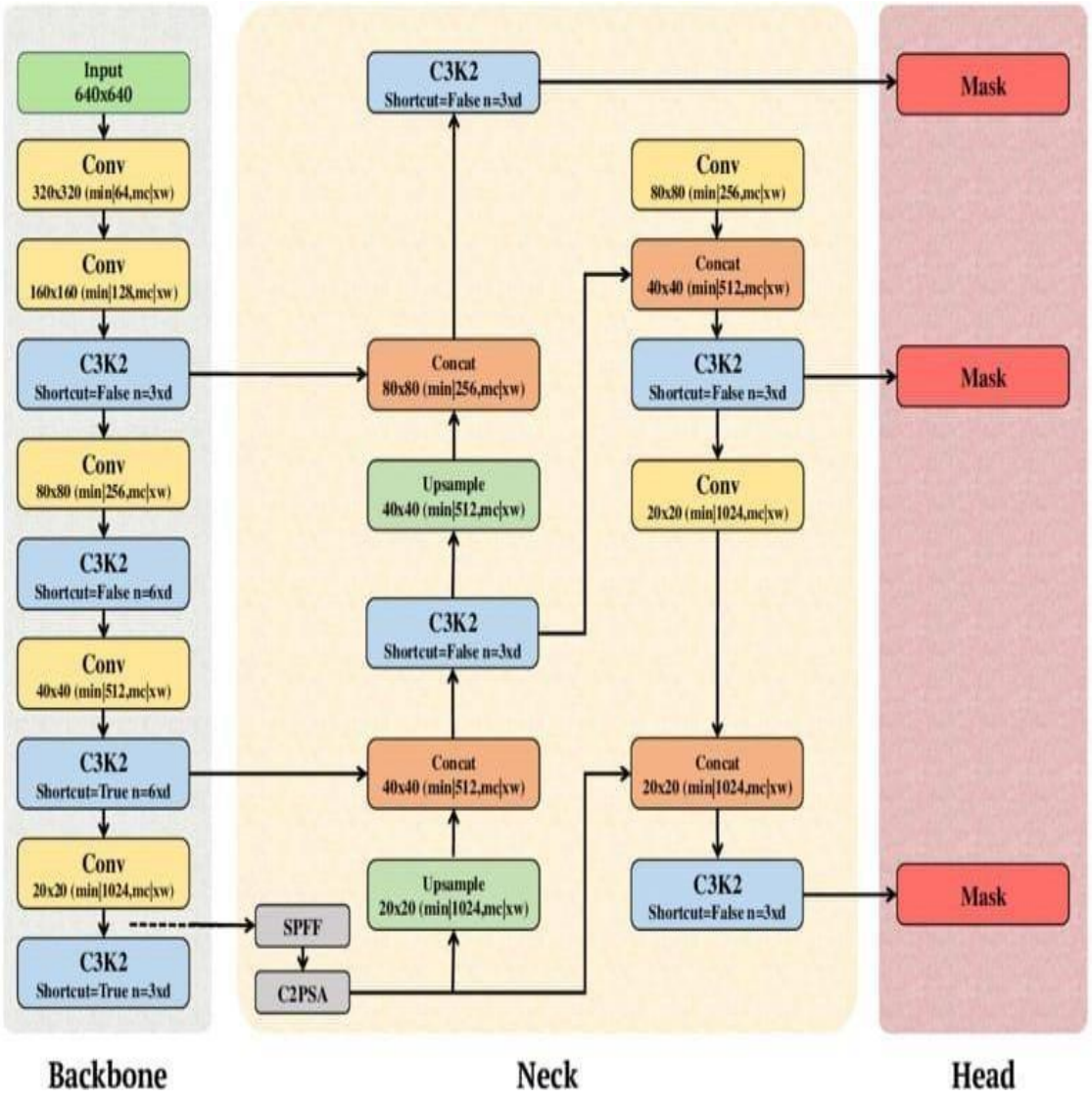

Figure 5.2 System Architecture of YOLOv11

5.2.1.1 Backbone: Efficiency-Focused Feature Extraction

The YOLOv11 backbone retains the hierarchical feature extraction principle but emphasizes efficiency through the use of **C3k2 modules**.

1. **C3k2 Module:** This module replaces the C2f module of YOLOv8. The "k2" in the name likely refers to the use of smaller kernel sizes, potentially 3x3 convolutions arranged in a specific, efficient manner. The internal

structure of C3k2 is designed to achieve similar or better feature extraction capabilities with reduced computational cost compared to C2f. Like C2f, the n parameter controls the number of internal bottleneck layers, and shortcut indicates the presence of residual connections. The focus on smaller kernels can lead to more fine-grained feature learning.

2. **Progressive Downsampling:** Similar to YOLOv8, convolutional layers with a stride of 2 are used to progressively reduce the spatial dimensions, creating the multi-scale feature maps for the neck.

5.2.1.2 Neck: Enhanced Feature Fusion with Attention

The neck in YOLOv11 continues to aggregate multi-scale features, but the key innovation here is the introduction of the **C2PSA module**, which incorporates spatial attention.

1. **SPPF:** The SPPF module is still likely used to provide multi-scale context from the deepest feature map.
2. **C2PSA (Convolutional block with Parallel Spatial Attention):** This is a crucial addition. Spatial attention mechanisms allow the network to learn which spatial regions in the feature maps are the most important for the task at hand (e.g., object detection or segmentation). The "Parallel Spatial Attention" likely refers to a design where spatial attention is computed and applied in parallel branches within the module, potentially capturing different aspects of spatial importance. By focusing on the relevant spatial cues, the network can improve its ability to detect and segment objects, especially in cluttered scenes or when dealing with occlusions.
3. **C3k2 Modules in the Neck:** Consistent with the backbone, C3k2 modules are used for efficient feature processing and fusion within the neck pathways.

4. **PANet Structure:** The underlying PANet architecture for bottom-up and top-down feature aggregation is likely retained or further optimized.

5.2.1.3 Head: Task-Specific Prediction

The head in YOLOv11 is task-dependent, as illustrated by the "Mask" output in your provided diagram, indicating an instance segmentation head.

1. **Shared Feature Processing:** The initial convolutional layers in the head might be shared for both object detection and segmentation tasks.
2. **Segmentation Branch:** For instance segmentation, the head will have a separate branch dedicated to predicting the segmentation masks. This typically involves:
    a. Further convolutional layers to process the features relevant for segmentation.
    b. Upsampling layers to increase the spatial resolution of the mask predictions to match the object's size.
    c. A final convolutional layer (or a series of them) to predict the pixel-wise binary mask for each detected object.
3. **Detection Branch (Implicit):** If YOLOv11 were used for object detection alone, the head would have a "Detect" output similar in concept to YOLOv8, predicting bounding boxes, objectness, and class probabilities, but potentially with optimizations related to the C3k2 modules.

CHAPTER 6

SYSTEM DESCRIPTION

6.1 PYTHON

Python is a popular programming language. It was created by Guido van Rossum, and released in 1991. Python is a high-level, general-purpose programming language. Its design philosophy emphasizes code readability with the use of significant indentation.

Python is dynamically type-checked and garbage-collected. It supports multiple programming paradigms, including structured (particularly procedural), object-oriented and functional programming. It is often described as a "batteries included" language due to its comprehensive standard library.

It is used for:

- web development (server-side),
- software development,
- mathematics,
- system scripting.

6.1.1 History of Python

Python was created in 1980s by Guido van Rossum. During his research at the National Research Institute for Mathematics and Computer Science in the Netherlands, he created Python – a super easy programming language in terms of reading and usage. The first ever version was released in the year 1991 which had only a few built-in data types and basic functionality.

Later, when it gained popularity among scientists for numerical computations and data analysis, in 1994, Python 1.0 was released with extra

features like map, lambda, and filter functions. After that adding new functionalities and releasing newer versions of Python came into fashion. The latest version of Python, Python 3.13 was released in 2024.

Newer functionalities being added to Python makes it more beneficial for developers and improved its performance. In recent years, Python has gained a lot of popularity and is a highly demanding programming language. It has spread its demand in various fields which includes machine learning, artificial intelligence, data analysis, web development.

6.1.2 Syntax and Semantic

Python is meant to be an easily readable language. Its formatting is visually uncluttered and often uses English keywords where other languages use punctuation. Unlike many other languages, it does not use curly brackets to delimit blocks, and semicolons after statements are allowed but rarely used. It has fewer syntactic exceptions and special cases than C or Pascal.

6.1.3 Indentation

Python uses whitespace indentation, rather than curly brackets or keywords, to delimit blocks. An increase in indentation comes after certain statements; a decrease in indentation signifies the end of the current block. Thus, the program's visual structure accurately represents its semantic structure.This feature is sometimes termed the off-side rule. Some other languages use indentation this way; but in most, indentation has no semantic meaning. The recommended indent size is four spaces.

6.1.4 Methods

Methods of objects are functions attached to the object's class; the syntax for normal methods and functions, instance.method(argument), is syntactic sugar for Class.method(instance, argument). Python methods have an explicit

self parameter to access instance data, in contrast to the implicit self (or this) parameter in some object-oriented programming languages (e.g., C++, Java, Objective-C, Ruby). Python also provides methods, often called dunder methods (because their names begin and end with double underscores); these methods allow user-defined classes to modify how they are handled by native operations including length, comparison, arithmetic , and type conversion.

6.1.5 Advantages and Disadvantages of Python

Every programming language comes with benefits and limitations as well. These benefits and limitations can be treated as advantages and disadvantages. Python also has a few disadvantages over many advantages. Let's discuss each here:

Advantages of Python

- Easy to learn, read, and understand
- Versatile and open-source
- Improves productivity
- Supports libraries
- Huge library
- Strong community
- Interpreted language

Disadvantages of Python

- Restrictions in design
- Memory inefficient
- Weak mobile computing

- Runtime errors
- Slow execution speed

6.2 LIBRARIES USED

6.2.1 Cv2 or Opencv

Opencv is a huge open-source library for computer vision, machine learning, and image processing. Now, it plays a major role in real-time operation which is very important in today's systems. By using it, one can process images and videos to identify objects, faces, or even the handwriting of a human.

When it is integrated with various libraries, such as NumPy, python is capable of processing the opencv array structure for analysis. To Identify an image pattern and its various features we use vector space and perform mathematical operations on these features.

The first OpenCV version was 1.0. OpenCV is released under a BSD license and hence it's free for both academic and commercial use. It has C++, C, Python, and Java interfaces and supports Windows, Linux, Mac OS, iOS and Android. When opencv was designed the main focus was real-timeapplications for computational efficiency.

To use these functions at the beginning of the program, you need to connect the opencv library, which is done by the command **import cv2**

OpenCV allows you to perform various operations in the image.

- **Read the Image** : OpenCV helps you to read the image from file or directly from camera to make it accessible for further processing.

- **Image Enhacncement** : You will be able to enhance image by adjusting the brightness , sharpness or contract of the image. This is helpful to visualize quality of the image.
- **Object detection**: As you can see in the below image object can also be detected by using OpenCV , Bracelet , watch , patterns, faces can be detected. This can also include to recognize faces , shapes or even objects .
- **Image Filtering**: You can change image by applying various filters such as blurring or Sharpening.
- **Draw the Image**: OpenCV allows to draw text, lines and any shapes in the images.
- **Saving the Changed Images**: After processing , You can save images that are being modified for future analysis.

6.2.2 Math

Math Module consists of mathematical functions and constants. It is a built-in module made for mathematical tasks.

The math module provides the math functions to deal with basic operations such as addition(+), subtraction(-), multiplication(*), division(/), and advanced operations like trigonometric, logarithmic, and exponential functions. To carry out calculations with real numbers, the Python language contains many additional functions collected in a library (module) called math.

To use these functions at the beginning of the program, you need to connect the math library, which is done by the command **import math**

Python provides various operators for performing basic calculations, such as * for multiplication,% for a module, and / for the division. If you are developing a program in Python to perform certain tasks, you need to work with trigonometric functions, as well as complex numbers. Although you cannot use

these functions directly, you can access them by turning on the math module math, which gives access to hyperbolic, trigonometric and logarithmic functions for real numbers. To use complex numbers, you can use the math module cmath. When comparing math vs numpy, a math library is more lightweight and can be used for extensive computation as well.

The Python Math Library is the foundation for the rest of the math libraries that are written on top of its functionality and functions defined by the C standard.

6.2.3 CVZone

cvzone is a Python package that simplifies the implementation of Computer Vision and AI functions. It is built on top of OpenCV and MediaPipe libraries, making it easier to perform tasks like image processing, hand tracking, face detection, and pose estimation.

Key Features:

- **Simplified Syntax:** cvzone provides a higher-level interface, reducing the amount of code needed to perform complex computer vision tasks.
- **Integration with OpenCV and MediaPipe:** It leverages the power and efficiency of these popular libraries.
- **Modules for Common Tasks:** cvzone includes modules for:
- **Hand Tracking:** Detect and track hands in real-time.
- **Face Detection:** Detect faces in images and videos.
- **Pose Estimation:** Track the pose of the human body.
- **Image Processing:** Provides functions for image manipulation, such as resizing, rotating, and stacking images.
- **Drawing Utilities:** Easily draw shapes, rectangles with rounded corners, and text on images.

- **Object Classification:** Integrate pre-trained models for image classification.
- **Distance Measurement:** Calculate distances between objects or points in an image.
- **Working with Serial Data:** Facilitates communication with hardware like Arduino.
- **Downloading Images:** Provides a convenient function to download images from URLs.
- **Frame Rate (FPS) Handling:** Easily calculate and display the frames per second of a video stream.

CHAPTER 7

SYSTEM IMPLEMENTATION

7.1 WORK FLOW

Step 1: Pre-processing

Step 2: Model selection and customization

Step 3: Training

Step 4: Evaluation

Step 5: Fine-tuning

Step 6: Exporting

Step 7: Deployment

Step 8: Testing and optimization

7.1.1 Preprocessing

In any object detection project, the first and most critical step is collecting and preparing the dataset. This involves gathering a large number of images that contain the objects we want to detect, and carefully annotating these images by marking the locations of each object with bounding boxes. Tools like LabelImg are commonly used for this annotation process. It's important to ensure that the annotations are saved in a format compatible with the TensorFlow Object Detection API, such as COCO, PASCAL VOC, or TFRecord formats, which are widely supported for training models. To make the data preparation process more efficient and organized, we can use MLflow, an open-source platform designed to manage machine learning workflows.

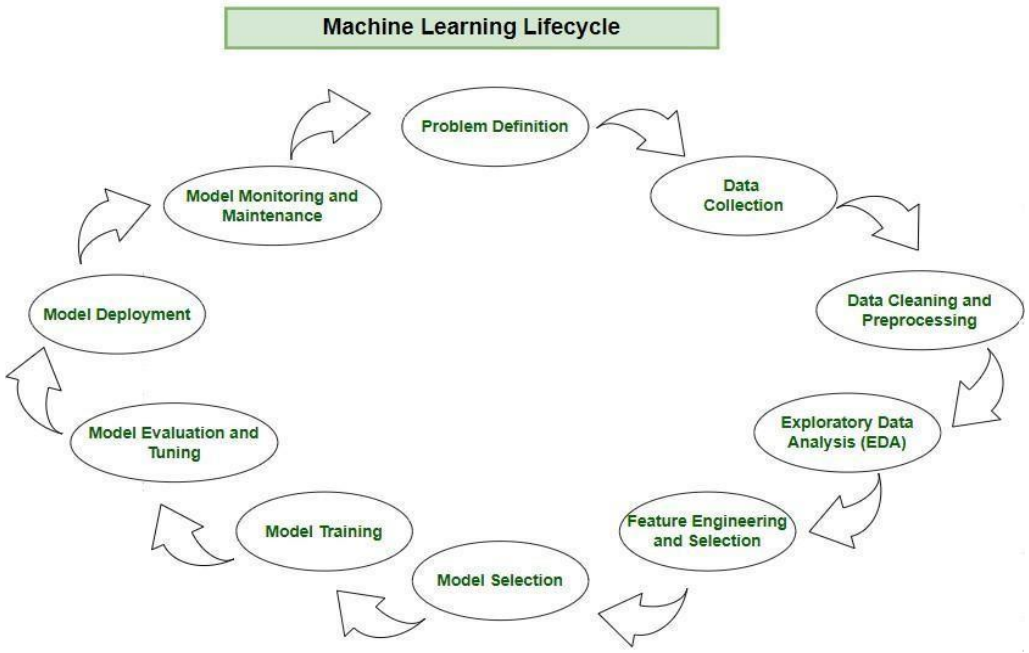

Figure 7.1 Machine Learning Lifecycle

MLflow helps track various aspects of the project, such as the number of images collected, the annotation format used, and how the dataset is split into training and validation sets. By logging this information, MLflow ensures better reproducibility, making it easier to revisit and improve the project later. It also helps maintain consistency when multiple experiments are performed. Overall, integrating MLflow into the data preparation phase provides a structured and trackable approach, helping manage both the data and the overall machine learning lifecycle effectively.

7.1.2. Model Selection and Customization

The TensorFlow Object Detection API offers a variety of pre-trained models like Faster R-CNN, SSD, and YOLO. These models have been trained on large datasets and deliver strong performance in general object detection tasks. However, pre-trained models are not always ideal for every project, especially if we need to detect objects in a specialized domain or with unique characteristics. In such cases, customization is necessary to better suit the specific requirements.

Customization can involve several changes, such as modifying the model's architecture, adjusting hyperparameters, resizing the model for efficiency, changing the number of network layers, or introducing new object categories that the original model was not trained to detect. These adjustments help fine-tune the model for better accuracy and relevance to the target application.

To manage and document this customization process, MLflow can be used. MLflow enables tracking of key information like the choice of model, architecture modifications, hyperparameter settings, and other changes. By recording this data, MLflow ensures that experiments are reproducible and that it is easier to understand what changes led to performance improvements. Overall, MLflow helps in organizing the model development workflow systematically and efficiently.

7.1.3. Training

After preparing the data and customizing the model, the next step is to train the model. The TensorFlow Object Detection API offers built-in tools to simplify this process, including scripts for loading data, training models, and evaluating their performance. However, training object detection models is often very resource- intensive and can take a long time, especially on large datasets. To speed up training, it's recommended to use a GPU, which can handle the heavy computations much faster than a CPU.

A smart approach is to first train the model on a small subset of the dataset. This quick test helps ensure that the model setup is correct and that there are no major errors before committing to full-scale training. Once everything is working properly, we can gradually increase the size of the training dataset and the number of training iterations to improve the model's accuracy.

Using MLflow during training is highly beneficial. MLflow can track important metrics like training and validation loss, learning rate, and the number of iterations completed. Recording this information helps in monitoring model performance over time, identifying issues early, and fine-tuning the training process for better results. It also ensures that experiments are well-documented and reproducible.

7.1.4. Evaluation

Once the model is trained, it is crucial to evaluate its performance using a separate test set. Evaluation helps us understand how well the model can generalize to new, unseen data, which is important for ensuring that it will perform reliably in real-world applications. The TensorFlow Object Detection API provides built-in tools for evaluating object detection models. It calculates important metrics such as mean Average Precision and mean Intersection over Union. These metrics measure the model's accuracy by checking how closely the predicted bounding boxes match the actual ground truth boxes in the images.

Accurate evaluation is key to identifying whether the model is ready for deployment or needs further improvement. Using MLflow during the evaluation phase allows us to systematically track and store evaluation results, including all important metrics. It also enables easy comparison of different models and hyperparameter settings. By recording each experiment's evaluation results, we can better analyze which model performed best under which conditions, making it easier to choose the final model and to fine-tune hyperparameters if necessary. This organized tracking not only improves transparency but also supports reproducibility and more efficient optimization of the object detection pipeline.

7.1.5. Fine-tuning

After evaluating the model, we might find that its performance is not good enough for our needs. When this happens, fine-tuning the model is an effective next step. Fine-tuning involves making adjustments to improve model performance, such as changing hyperparameters (like learning rate or batch size) or using a larger and more diverse dataset to expose the model to more variations. Fine-tuning is a common and powerful technique in deep learning that can lead to significant improvements.

One practical approach is to use a pre-trained model that has already learned useful features from a similar domain. We can then fine-tune this model on our specific dataset, allowing it to adapt better to our particular task without starting from scratch. This method saves time and usually results in faster convergence and better accuracy.

Using MLflow during fine-tuning is extremely helpful. MLflow can track all aspects of the fine-tuning process, including changes to hyperparameters and evaluation metrics. It also allows us to compare the performance of different fine- tuned models in a systematic way. This detailed tracking makes it easier to identify which fine-tuning strategy worked best, ultimately helping us optimize the model for our specific problem domain.

7.1.6. Exporting

After training and fine-tuning the model, the next important step is to export it for inference. The TensorFlow Object Detection API allows exporting the model in different formats, such as TensorFlow Saved Model, TensorFlow Lite, or TensorFlow.js. Each format serves different needs: for example, TensorFlow Lite is optimized for mobile and embedded devices, while TensorFlow.js is designed for web applications. Exporting the model correctly

ensures that it can be deployed easily across various platforms and environments.

It’s important to make sure the exported model remains consistent with the settings used during training and fine-tuning to maintain its performance. Using MLflow during the export process helps by tracking important details such as the export format, version, and any configurations applied. This systematic tracking ensures that the exported model is reliable, reproducible, and easy to deploy, making the overall machine learning workflow much smoother and more organized.

7.1.7. Deployment

After exporting the trained model, the next step is to deploy it in a production environment where it can be used to make real-world predictions. Deployment usually involves setting up an inference pipeline that takes input data (like images or videos), runs it through the object detection model, and outputs the detected objects with their locations. Depending on the application, deployment can become complex, as it may need to address challenges like scalability (handling many users), latency (making fast predictions), and security (protecting the data and model).

Managing the deployment process carefully is important to ensure the model performs reliably in production. Using MLflow can help in this phase by tracking key details such as the deployment environment, configuration settings, and any issues that occur during deployment. This organized tracking makes it easier to troubleshoot problems, maintain the system, and ensure smooth, efficient operation of the deployed model.

7.1.8. Testing and Optimization

After deploying the model, it is essential to thoroughly test it and continuously optimize its performance. Testing ensures that the model works correctly in the real-world environment and meets the application's requirements. This process may include monitoring the model’s outputs, measuring its accuracy in production, and identifying any performance issues, such as slow inference times or incorrect predictions. Gathering feedback from users is also very valuable, as it provides insights into how well the model is meeting user expectations and where it may need improvement.

Optimization might involve fine-tuning the model further, retraining it with new data, or adjusting system resources to improve scalability and speed. This ensures the model remains accurate, reliable, and efficient over time as real-world conditions change. Using MLflow during the testing and optimization phases helps keep a detailed record of the model’s performance, any feedback collected, and all updates or retraining efforts. This tracking makes it easier to analyze trends, troubleshoot issues, and document improvements. Maintaining a clear history of changes and feedback is crucial for continuously enhancing the model and ensuring its long-term success in production environments.

CHAPTER 8

RESULT AND ANALYSIS

This chapter represents the overall result and analysis of Motorcyclist Helmet Detection. Here we have showcased our output for YOLO versions (v8 and v11).

8.1 RESULT FOR YOLOv8

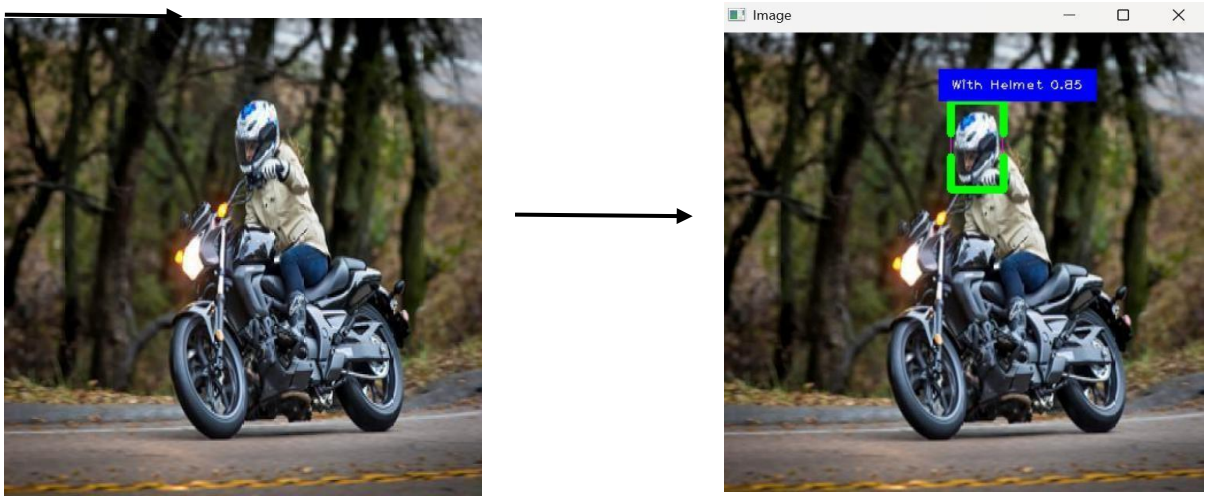

Figure: 8.1 Sample input and Output (With helmet)

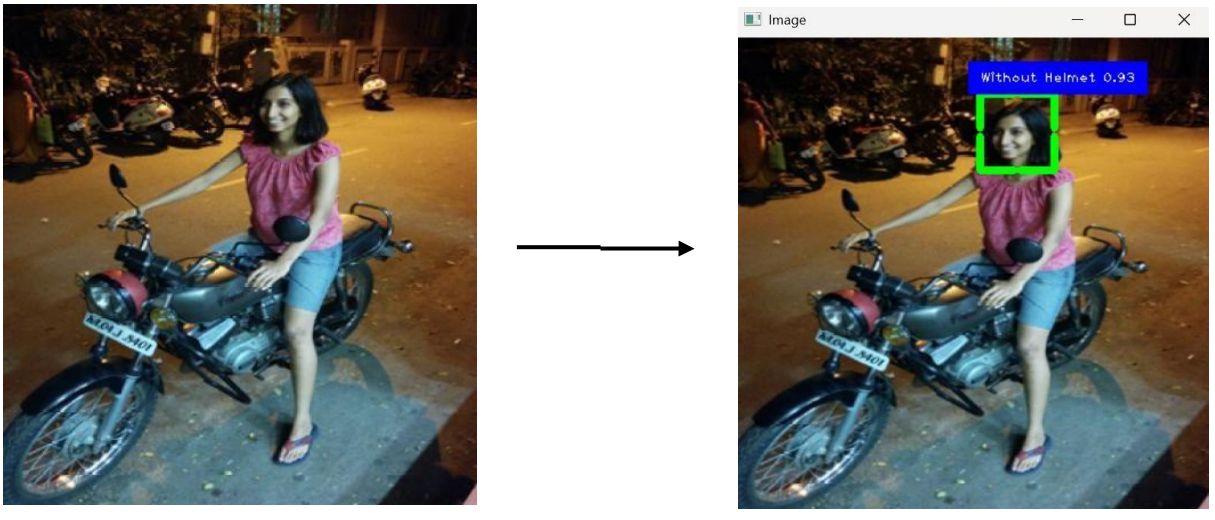

Figure: 8.2 Sample input and Output (Without helmet)

8.2 RESULT FOR YOLOv11

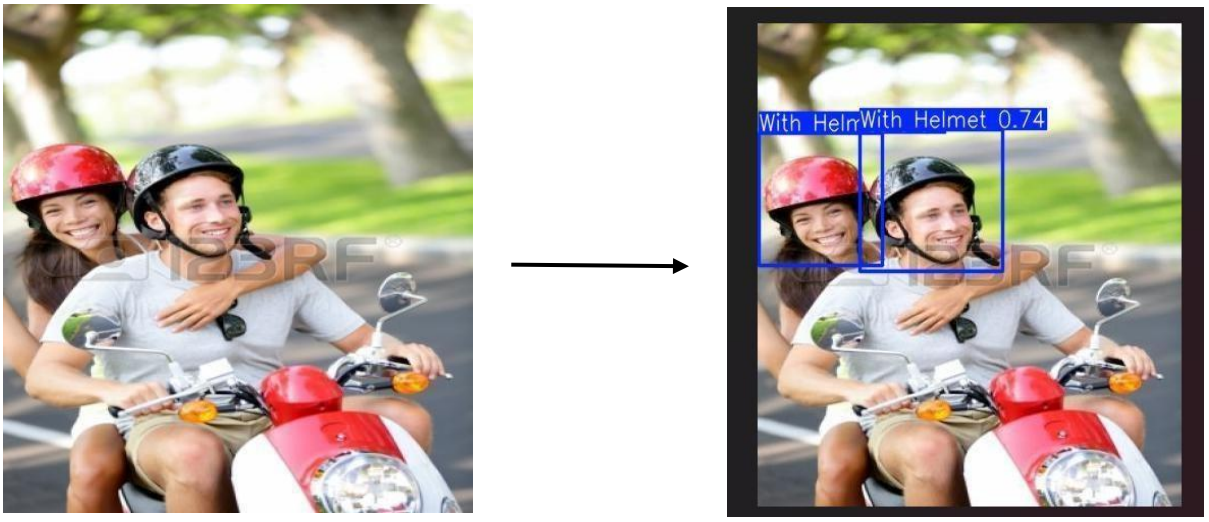

Figure: 8.3 Sample input and Output (With helmet)

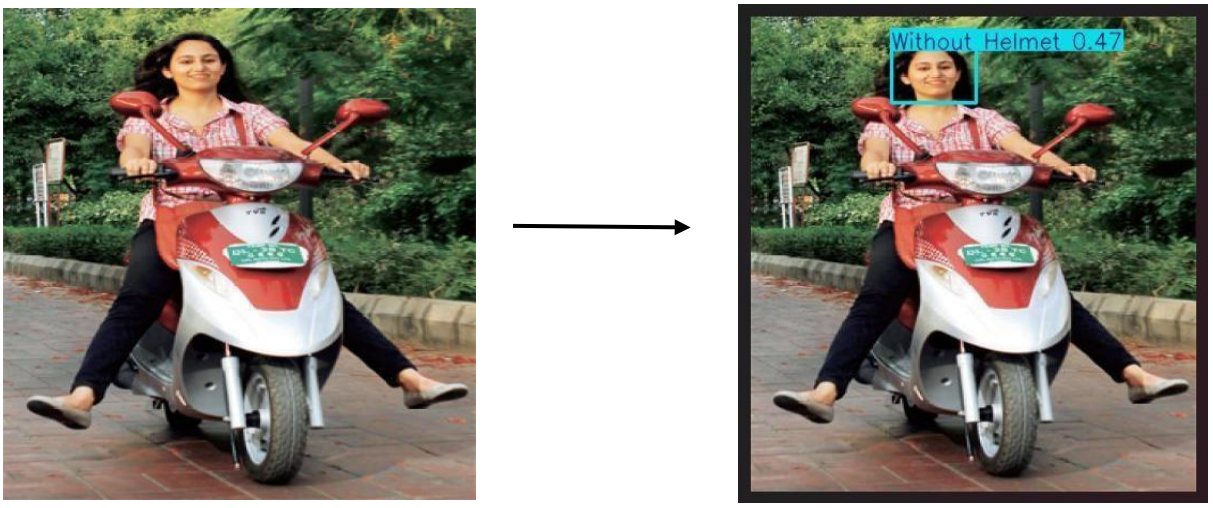

Figure: 8.4 Sample input and Output (Without helmet)

After testing our models, we have performed a confusion matrix to analyse the accuracy of both the versions. We have analyse the performance of the model and plotted graph. Atlast, we have compared both versions and shown the better accuracy.

8.3 METRIX ANALYSIS OF YOLOv8

CONFUSION MATRIX	Predicted Positive	Predicted Negative
Actual Positive	**45**	**1**
Actual Negative	**1**	**13**

Table 8.1 Confusion Matrix of YOLOv8

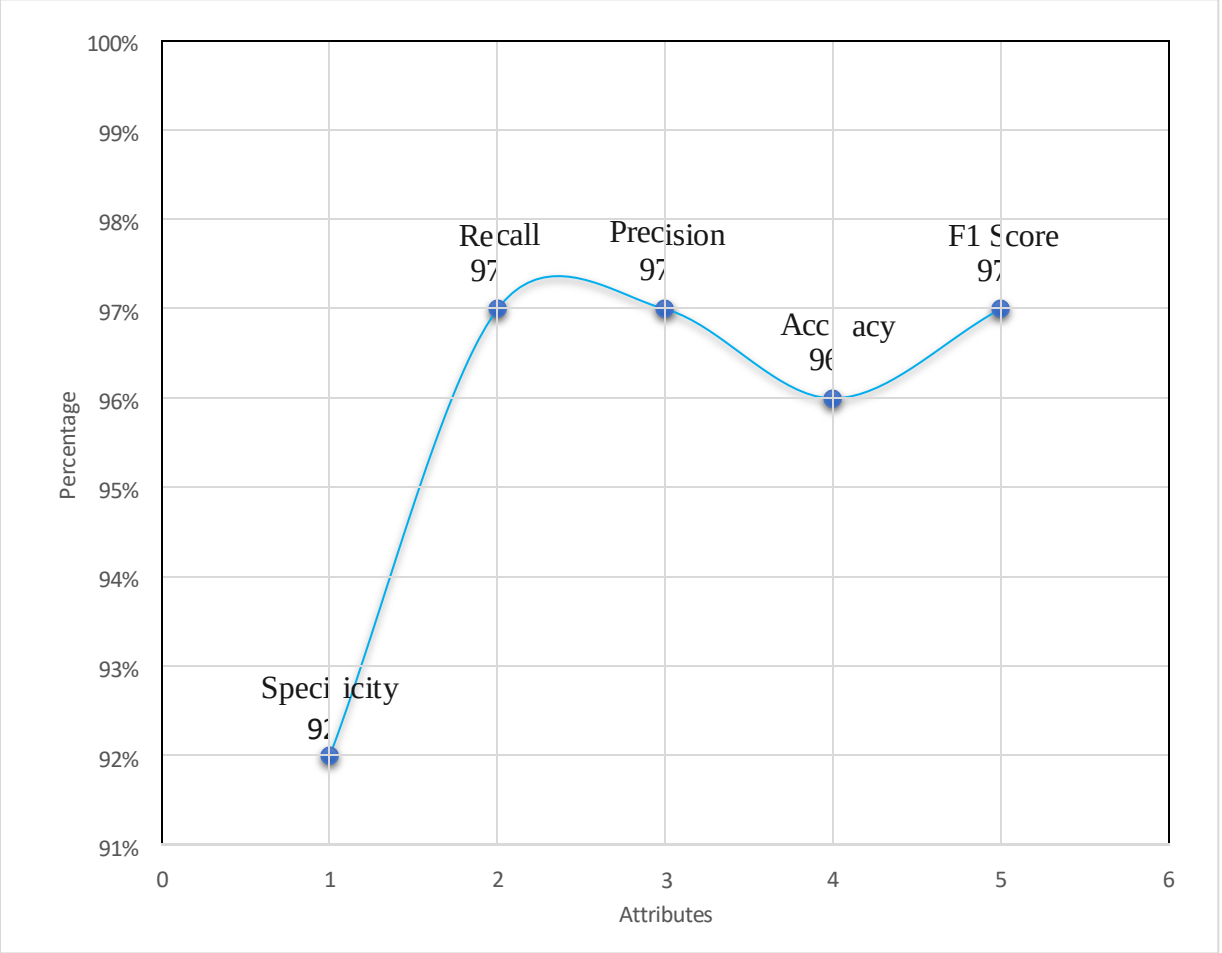

Figure: 8.5 Graph of YOLOv8

8.4 METRIX ANALYSIS OF YOLOv11

CONFUSION MATRIX	Predicted Positive	Predicted Negative
Actual Positive	42	3
Actual Negative	1	17

Table 8.2 Confusion Matrix of YOLOv11

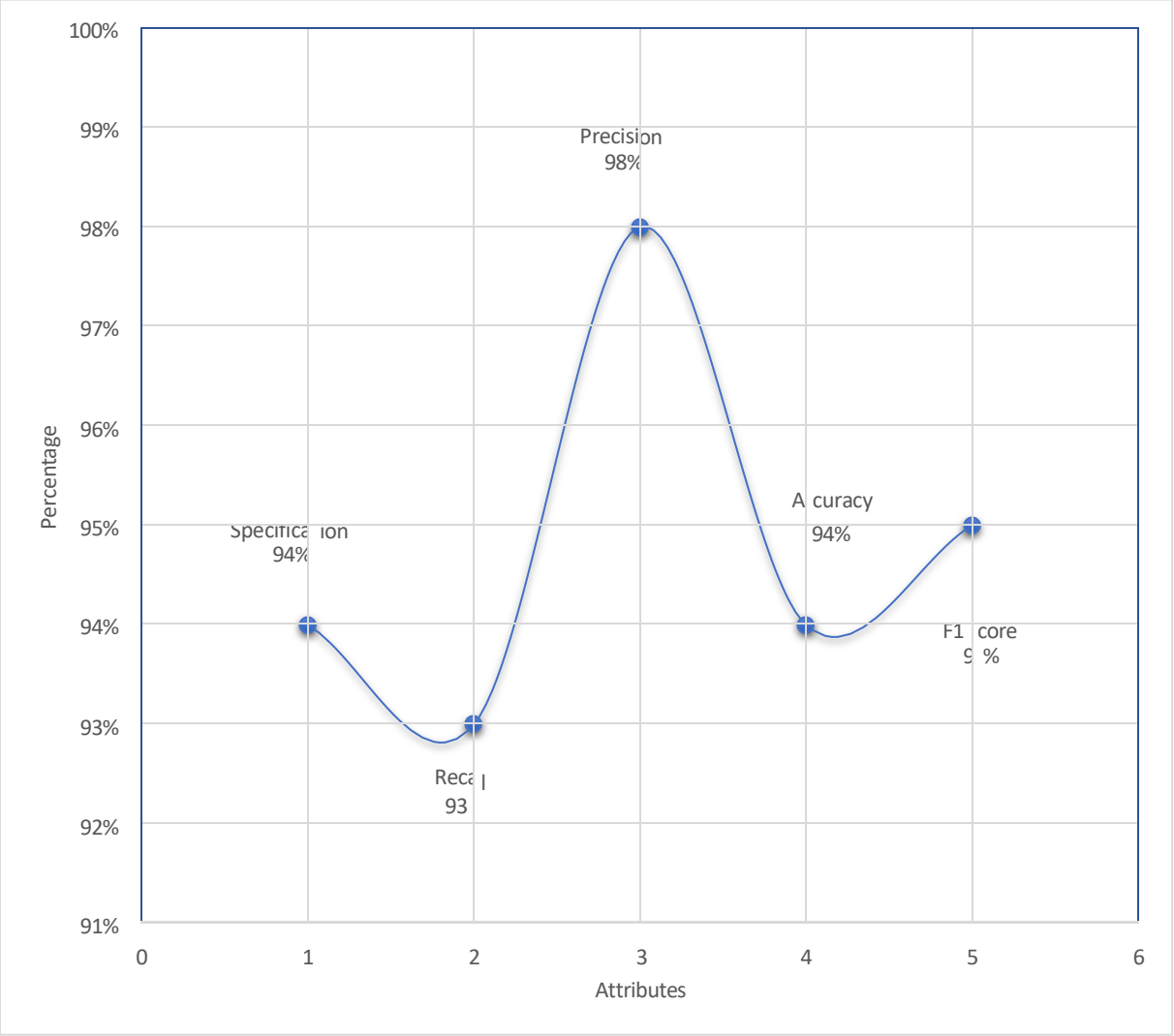

Figure: 8.6 Graph of YOLOv11

8.5 COMPARATIVE ANALYSIS OF YOLOv8 AND YOLOv11

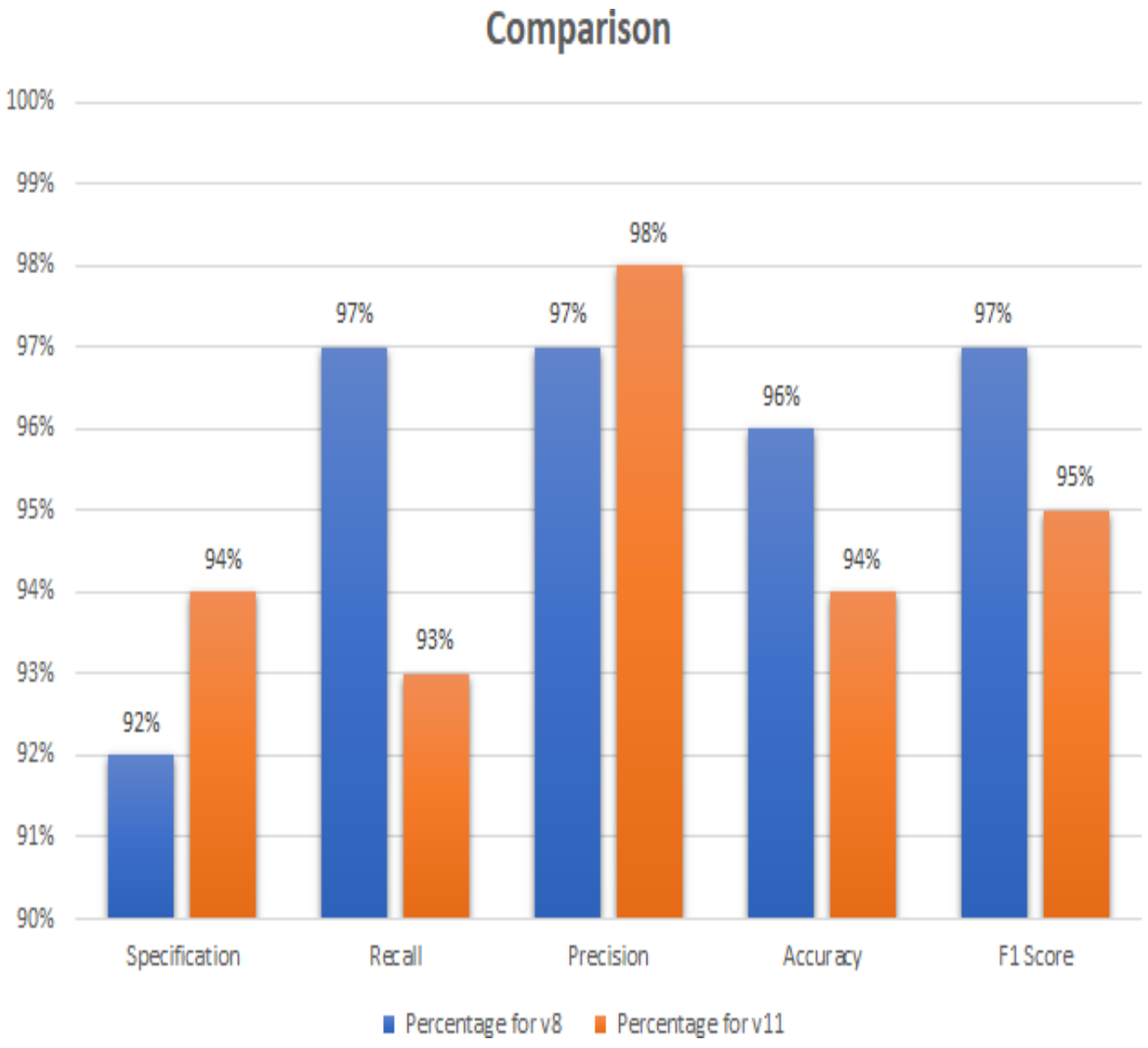

Figure: 8.7 Comparison graph of YOLOv8 and YOLOv11

CHAPTER 9

CONCLUSION AND FUTURE WORK

9.1 CONCLUSION

We have introduced a method utilizing deep learning for automatic helmet detection, employing the YOLO algorithm, specifically YOLOv8 and YOLOv11, to determine which version achieves superior accuracy. This approach enables the automatic identification of motorcycles in images and assesses whether the rider is wearing a helmet. Our dataset was compiled from real-time observations as well as online sources to effectively train the model. We trained both YOLOv8 and YOLOv11 to ascertain which provides the highest accuracy. During the training process, we obtained weights for both models. Upon testing, we found that YOLOv8 achieved an accuracy of 96%, while YOLOv11 reached 94%. This indicates that YOLOv8 outperforms YOLOv11 in terms of accuracy.

9.2 FUTURE WORK

In future work, we can enhance the accuracy and reliability of these systems by tackling issues like variations in lighting, helmet types, and occlusion. Additionally, while we focused on helmet detection during the day, we can expand this to include detection at night using night vision cameras.

APPENDIX

Source code for YOLOv8

```
import cv2

import math

import cvzone

from ultralytics import YOLO

# Load YOLO model with custom weights

yolo_model = YOLO("Weights/best.pt")

# Define class names

class_labels = ['With Helmet', 'Without Helmet']

# Load the image

image_path = "Media/Test_image/BikesHelmets (28).jpg"

img = cv2.imread(image_path)

# Perform object detection

results = yolo_model(img)

# Loop through the detections and draw bounding boxes

for r in results:

    boxes = r.boxes

    for box in boxes:

        x1, y1, x2, y2 = box.xyxy[0]

        x1, y1, x2, y2 = int(x1), int(y1), int(x2), int(y2)
```

```
        w, h = x2 - x1, y2 - y1
        cvzone.cornerRect(img, (x1, y1, w, h))
        conf = math.ceil((box.conf[0] * 100)) / 100
        cls = int(box.cls[0])
        if conf > 0.1:
            cvzone.putTextRect(img, f'{class_labels[cls]} {conf}', (x1, y1 - 10),
scale=0.8, thickness=1, colorR=(255, 0, 0))
        # Display the image with detections
cv2.imshow("Image", img)
# Close window when 'q' button is pressed
while True:
    if cv2.waitKey(1) & 0xFF == ord('q'):
        break
cv2.destroyAllWindows()
cv2.waitKey(1)
```

Source code for YOLOv11

The Following Code is Used To Train The Model

```
from ultralytics import YOLO
# Load a model
model = YOLO("yolo11n.pt")
```

```
# Train the model
train_results = model.train(
    data="D:\YOLO11\Demo_custom_yolo11\Dataset 2/data.yaml", # path to dataset YAML
   epochs=10, # number of training epochs
   imgsz=640, # training image size
    device="cpu", # device to run on, i.e. device=0 or device=0,1,2,3 or device=cpu
)
from ultralytics import YOLO
#Load a model
model = YOLO("runs/detect/train4/weights/best.pt")
#Perform object detection on an image
results = model("test_images/3.jpg" , save = True)
results[0].show()
```

REFERENCES

[1] WHO. Global status report on road safety. 2018 June17. https://www.who.int/publications/i/item/9789241565684 [Accessed: 22 April 2024].

[2] Gadkari N. Road accidents in India 2021. Ministry of Road Transport and Highways 2021.https://morth.nic.in/sites/default/files/RA_2021Compressed.pdf [Accessed:22 April 2024].

[3] Hwang S, Kim Y, Kwon Y, Park J, Won J. Object detection for cargo unloading system based on Fuzzy C means. Computer Mater Contin 2022 May 1;71(2). https://doi.org/10.32604/cmc.2022.023295.

[4] Jian M, Wang G, Zhang M. YOLO-AA: an efficient object detection model via strengthening fusion context information. Multimedia Tools Application Jan 2024;83(4): 10661–76. https://doi.org/10.1007/s11042-023-16063-9.

[5] Chaturvedi P, Lavingia K, Raval G. Detection of traffic rule violation in University campus using deep learning model. Int J Syst Assur Eng Manage Dec 2023;14 (6):2527–45. https://doi.org/10.1007/s13198-023-02107-8.

[6] Qin J, Tan Y, Xiang X, Xiong NN, Zhou Q. Algorithm of helmet wearing detection based on AT-YOLO deep mode. Computer Mater Contin 1 Oct 2021;69(1). https://doi.org/10.32604/cmc.2021.017480.

[7] Anantharaman R, Lee Y, Velazquez M. Utilizing mask R-CNN for detection and segmentation of oral diseases. In: In 2018 IEEE international conference on bioinformatics and biomedicine (BIBM). IEEE; 3 Dec 2018. p. 2197–204. https://doi.org/10.1109/BIBM.2018.8621112.

[8] AbdelSalam H, El-Khoribi R, Mahmoud A, Mohamed S. Object detection using adaptive mask RCNN in optical remote sensing images. Int J Intell Eng Syst Feb 2020;13(1):65–76. https://doi.org/10.22266/ijies2020.0229.07.

[9] Doi Y, Kenjo M, Miura H, Nagata Y, Nakao M, Ohnishi K, Ozawa S. Automatic gas detection in prostate cancer patients during image-guided radiation therapy using a deep convolutional neural network. Phys Med 1 Aug 2019; 64:24–8. https://doi.org/10.1016/j.ejmp.2019.06.009.

[10] Bharati P, Pramanik A. Deep learning techniques—R-CNN to mask R-CNN: a survey. Computational Intelligence in Pattern Recognition: proceedings of CIPR 2019. Advance Intelligent System Computer 2020; 999:657–68. https://doi.org/10.1007/978-981-13-9042-5_56.

[11] B. Anil Kumar, A. Khadhir, and L. D. Vanajakshi, ―Analysis of global positioning system-based bus travel time data and its use for advanced public transportation system applications,‖ Journal of Intelligent Transportation Systems, vol. 25, no. 1, pp. 58–76, 2021.

[12] T. Garg and G. Kaur, ―A systematic review on intelligent transport systems,‖ Journal of Computational and Cognitive Engineering, 2022.

[13] M. Asif, M. B. Ahmad, K. Masood, and M. Rizwan ―Park my ride: your true parking companion,‖ in Proceedings of the International Conference on Intelligent Technologies and Applications, pp. 695–708, Bahawalpur, Pakistan, November 2019.

[14] M. S. Abbas, M. B. Ahmad, M. Asif, A. Hassan, and M. Z. Tariq, ―An automatic accident detection system: a hybrid solution,‖ in Proceedings of the 2019 4th International Conference on Information Systems Engineering (ICISE), pp. 53–57, Shanghai, China, May 2019.

[15] Traffic accidents, ―Traffic accidents,‖ 2012, https://www.pbs.gov.pk/node/1124.

[16] S. Agondeze, C. Ddamulira, S. S. Kizza, and P. Vuzi, ―Occupational hazards among laboratory hub riders in selected health centers in central region

of Uganda,‖ Direct Research Journal of Public Health and Environmental Technology, vol. 6, no. 2, pp. 6–20, 2021.

[17] N. Akhtar and K. Pathak, ―Carbon nanotubes in the treatment of skin cancers: safety and toxicological aspects,‖ Pharmaceutical Nanotechnology, vol. 5, no. 2, pp. 95–110, 2017.

[18] S. Blows, S. Boufous, R. Ivers, B. C. Liu, S. K. Lo, and R. Norton, ―Helmets for Preventing Injury in Motorcycle Riders,‖ Cochrane database of systematic reviews, vol. 23, 2008.

[19] C. S. Olsen, A. M. Omas, M. Singleton et al., ―Motorcycle helmet effectiveness in reducing head, face and brain injuries by state and helmet law,‖ Injury epidemiology, vol. 3, pp. 8–11, 2016.

[20] N. Bellamy, J. K. Hendrikz, R. Le Brocque, C. Willis and C. Wilson, ―Speed Cameras for the Prevention of Road Traffic Injuries and Deaths,‖ Cochrane database of systematic reviews, vol. 6, 2010.

[21] K. Dahiya, C. K. Mohan, and D. Singh``Automatic detection of bike-riders without helmet using surveillance videos in real-time," in *Proc. Int. Joint Conf. Neural Netw. (IJCNN)*, Jul. 2016, pp. 3046_3051.

[22] S. Babu, C. K. Mohan, D. Singh, and C. Vishnu, ``Detection of motorcyclists without helmet in videos using convolutional neural network," in *Proc. Int. Joint Conf. Neural Netw. (IJCNN)*, May 2017, pp. 3036_3041.

[23] N. Boonsirisumpun, W. Puarungroj, and P. Wairotchanaphuttha, ``Automatic detector for bikers with no helmet using deep learning," in *Proc. 22nd Int. Comput. Sci. Eng. Conf. (ICSEC)*, Nov. 2018, pp. 1_4.

[24] C. V. Jiji, and L. Shine ``Automated detection of helmet on motorcyclists from traffic surveillance videos: A comparative analysis using hand-crafted

features and CNN," *Multimedia Tools Appl.*, vol. 79, pp. 14179_14199, Feb. 2020.

[25] N. Dalal and B. Triggs, ``Histograms of oriented gradients for human detection," in *Proc. IEEE Comput. Soc. Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2005, pp. 886_893.

[26] P. Dollar, R. Girshick, P. Goyal, K. He, and T.-Y. Lin, ``Focal loss for dense object detection," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 42, no. 2, pp. 318_327, Feb. 2020.

[27] A. Farhadi and J. Redmon, ``YOLO9000: Better, faster, stronger," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jul. 2017, pp. 6517_6525.

[28] S. Belongie, P. Dollár, J. Hays, T.-Y. Lin, M. Maire, P. Perona, D. Ramanan, and C. L. Zitnick, ``Microsoft COCO: Common objects in context," in *Proc. Eur. Conf. Computer. Vis. (ECCV)*, Zurich, Switzerland. Springer, 2014, pp. 740_755.

[29] R. E. Kalman, ``A new approach to linear _ltering and prediction problems," *J. Basic Eng.*, vol. 82, no. 1, pp. 35_45, Mar. 1960.

[30] J. Munkres, ``Algorithms for the assignment and transportation problems," *J. Soc. Ind. Appl. Math.*, vol. 5, no. 1, pp. 32_38, Mar. 1957.