# HELMET DETECTION SYSTEM USING VISION TRANSFORM ALGORITHMS

## MINI PROJECT REPORT

**Submitted by**

**GANESH RAJ S (Register Number: 2301507308)**

**JANARTHANAN P (Register Number: 2301507310)**

**Under the guidance of**

**Dr. P. DHARANYADEVI**

**(ASSISTANT PROFFESSOR)**

**Department of Computer Science and Engineering**

**To the Puducherry Technological University, in partial fulfillment of therequirement for the award of the degree**

**MASTER OF COMPUTER APPLICATION**


**DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING**

**PUDUCHERRY TECHNOLOGICAL UNIVERSITY PUDUCHERRY – 605 014**

**JANUARY - 2025**

**DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING**
**PUDUCHERRY TECHNOLOGICAL UNIVERSITY PUDUCHERRY – 605 014.**

## BONAFIDE CERTIFICATE

This is to certify that the Mini Project Work titled **"HELMET DETECTION SYSTEM USING VISION TRANSFORM ALGORITHMS"** is a Bonafide work done by **GANESH RAJ S (2301507308), JANARTHANAN P (2301507310)** in partial fulfillment for the award of the degree of **Master Computer Application** of the **Puducherry Technological University** and that this work has not been submitted for the award of any other degree of this/any other institution.

| | |
|---|---|
| **Project Guide** | **Head of the Department** |
| **(Dr. P. DHARANYADEVI)** | **(Dr. E. ILLAVARASAN)** |

# ACKNOWLEDMENT

I am deeply indebted to **Dr. P. Dharanyadevi (Assistant Proffessor),** Department of Computer Science and Engineering, Puducherry Technological University, Pondicherry, for her valuable guidance throughout the mini project.

I also express my heart-felt gratitude to **Dr. E. Illavarasan ,** Professor & Head (CSE) for giving constant motivation in succeeding my goal.

With profoundness I would like to express my sincere thanks to **Dr. S. Mohan,** Vice Chancellor, Puducherry Technological University, for his kindness in extending the infrastructural facilities to carry out my Mini project work successfully. My sincere thanks are due to his team for facilitating every requirement for the successful completion and evaluation of our project work.

I also express my thanks to all the **Faculty** and **Technical Staff** members of the CSE department for their timely help and the **Central Library** for facilitating useful reference materials.

I would be failing in my duty if I don't acknowledge the immense help extended by my friends, who have always been with me in all my trails and tribulations and encouraging me to complete.

**GANESH RAJ S**

**JANARTHANAN P**

## TABLE OF CONTENT

# HELMET DETECTION SYSTEM USING VISION TRANSFORM ALGORITHMS

## ABSTRACT

Ensuring road safety, particularly for motorcyclists, is a critical global concern. One effective measure to enhance safety for riders is the use of helmets, which significantly reduces the risk of severe head injuries in accidents. However, enforcing helmet usage can be challenging due to limited resources for manual monitoring and enforcement. This project aims to develop an automated helmet detection system using an improved version of the YOLO V8 (You Only Look Once) object detection algorithm, tailored for real -time helmet recognition.

By optimizing YOLO V8, the system achieves high accuracy in detecting whether a motorcyclist is wearing a helmet, even in varied and challenging conditions, such as poor lighting, occlusion, or varying helmet colors. The project involves the design, training, and deployment of a neural network model enhanced with specific improvements over the base YOLO V8 architecture to boost its robustness and speed. The system is designed to be integrated into real-world applications, such as surveillance cameras at traffic intersections, where it can continuously monitor and identify helmet usage among motorcyclists.

The proposed solution addresses the limitations of previous models by improving detection rates, reducing false positives, and enhancing computational efficiency, making it suitable for large-scale deployment. The results demonstrate that the improved YOLO V8 model achieves a significant increase in both detection accuracy and processing speed compared to previous versions, proving it to be an effective tool for enforcing helmet compliance in real time. This project contributes to the field of traffic safety automation and has the potential to assist traffic authorities in ensuring compliance with helmet-wearing regulations, ultimately reducing fatalities and injuries among motorcyclists.

## LIST OF FIGURES

## LIST OF ABBREVIATIONS

1. **ML** - Machine Learning
2. **AI -** Artificial Intelligence
3. **YOLO V8** - You Only Look Once
4. **ViT** - Vision Transformer
5. **CNN** - Convolutional Neural Network
6. **R-CNN** - Region-based CNN
7. **SVM** - Support Vector Machine
8. **RPN -** Region Proposal Networks
9. **FPN -** Feature Pyramid Networks
10. **PCA -** Principal Component Analysis
11. **TP** - True Positives
12. **TN** - True Negatives
13. **FP** - False Positives
14. **FN** - False Negatives

# CHAPTER 1
# 1. INTRODUCTION

## 1.1 OVERVIEW

Helmet detection is a critical component in ensuring road safety and enforcing traffic laws. The ability to accurately detect whether motorcyclists are wearing helmets can significantly contribute to reducing injuries and fatalities caused by road accidents. Traditional methods for helmet detection often rely on manual monitoring or rule-based systems, which are limited in their efficiency and scalability. With advancements in technology and data analysis, it is now possible to utilize machine learning for automated helmet detection. Machine learning algorithms, with their ability to process and interpret complex image datasets, provide a promising alternative to traditional methods.

This project explores the application of machine learning algorithms to the problem of helmet detection. Specifically, it focuses on the comparison of two algorithms: the Convolutional Neural Network (CNN) and the Support Vector Machine (SVM). CNNs are widely used for image recognition tasks due to their ability to learn spatial hierarchies of features directly from raw image data. However, they require substantial computational resources and large datasets for training. On the other hand, SVM is a robust algorithm known for its effectiveness in classification tasks, even with smaller datasets, but it may struggle with highly complex image data.

The dataset used in this study consists of images of motorcyclists, labelled as wearing helmets or not. The workflow involves data preprocessing, image augmentation, splitting the dataset into training and testing subsets, applying the algorithms, and evaluating their performance based on metrics such as accuracy, precision, recall, and F1-score. This study aims to identify the more effective algorithm for helmet detection and provide insights into their suitability for real-world applications.

### 1.1.1 MACHINE LEARNING

Machine Learning is a subfield of Artificial Intelligence (AI) that focuses on developing algorithms and statistical models that allow computers to learn from and make decisions or predictions based on data. Unlike traditional programming, where explicit instructions are provided, machine learning systems are trained on data to identify patterns and relationships, enabling them to make predictions or decisions without being explicitly programmed.

Machine learning can be broadly categorized into three types:

**Supervised Learning**:

In supervised learning, the algorithm is trained on a labelled dataset, meaning the input data is paired with the correct output. The goal is to learn a mapping from inputs to outputs so that the model can make accurate predictions on new, unseen data. Examples include regression and classification tasks.

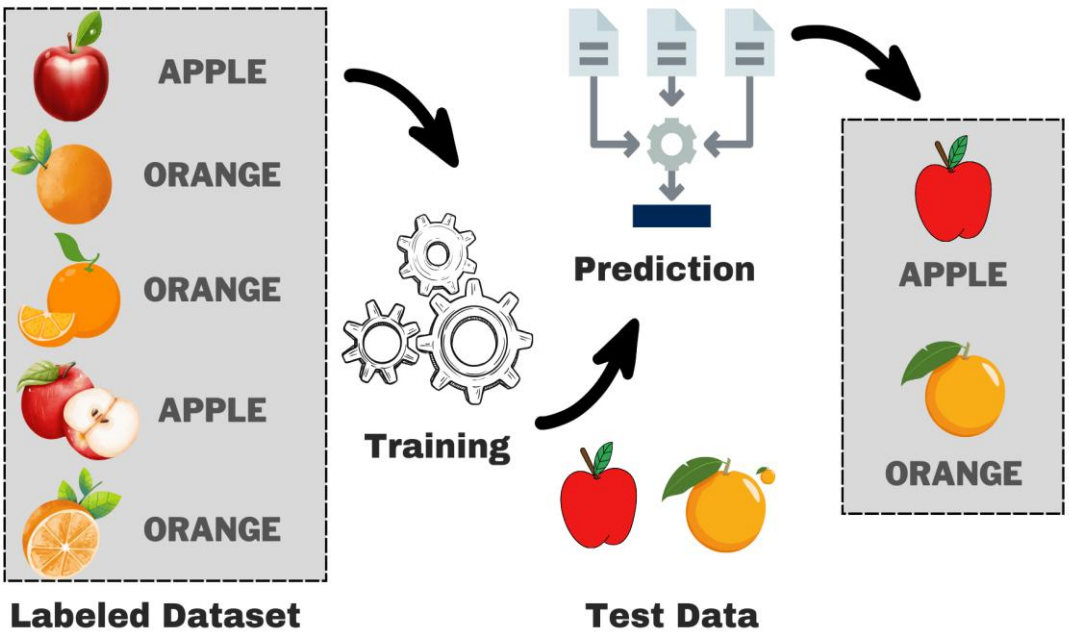



**Figure 1.1 Supervised Learning**

**Unsupervised Learning**:

This type of learning involves training the algorithm on data without labelled outputs. The system tries to identify patterns, structures, or relationships in the data. Examples include clustering and dimensionality reduction.

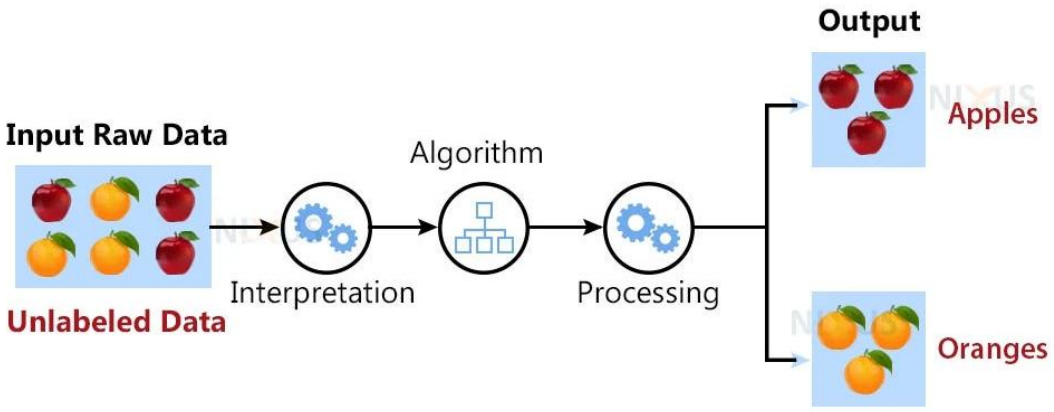



**Figure 1.2 Unsupervised Learning**

**Key Concepts in Machine Learning**

**Training and Testing**: A machine learning model is trained on a subset of data (training set) and evaluated on a separate subset (testing set) to measure its performance. This ensures the model generalizes well to unseen data.

**Overfitting and Underfitting**:

- **Overfitting** occurs when the model learns the training data too well, including noise, and fails to generalize to new data.
- **Underfitting** occurs when the model is too simple and fails to capture the underlying patterns in the data.

**Features and Labels**:

- **Features** are the input variables used by the model.
- **Labels** are the output variables the model aims to predict (e.g., with helmet, without helmet).

**Machine Learning in Helmet Detection**

Helmet detection involves analysing image data to determine whether motorcyclists

are wearing helmets. Machine learning is particularly well-suited for this task because

of its ability to:

- Handle large image datasets with multiple features.
- Capture complex patterns and variations in image data.

- Improve prediction accuracy through advanced techniques like Convolutional Neural Network and ensemble learning.

### 1.1.2 ALGORITHMS USED IN THE PROPOSED STUDY

**Vision Transform:**

- Vision Transformer (ViT): A Vision Transform is a transformer-based deep learning model that processes images for classification and object detection tasks. It divides the input image into patches, embeds them, and processes the sequence of patches using a transformer architecture to learn global and local patterns.
- Advantages:
  - Efficient in capturing long-range dependencies and contextual information.
  - Handles large datasets effectively with minimal inductive bias.
  - Provides state-of-the-art accuracy in image classification tasks.
- Limitations:
  - Requires large amounts of data for training.
  - Computationally intensive compared to CNN-based methods.
  - Sensitive to small changes in the dataset.

# CHAPTER 2

# 2. LITERATURE SURVEY

## 2.1 Machine Learning in Helmet Detection

Helmet detection systems have been a subject of study in various research fields, particularly within traffic safety and automated surveillance. Traditionally, helmet detection was manually enforced by law enforcement or through visual inspection via surveillance cameras. However, with the rise of machine learning and computer vision technologies, automated systems have become more viable. Various existing systems focus on identifying helmets using image classification, object detection, and deep learning techniques. Early methods, such as Support Vector Machines (SVM) and Haar cascades, faced limitations due to poor performance in diverse environmental conditions, like lighting variations or occlusions. Recent advancements in deep learning, particularly convolutional neural networks (CNNs) and object detection models, have led to more accurate and scalable solutions.

## 2.2 Object Detection Techniques and Algorithms

Object detection algorithms are pivotal in identifying and classifying objects within images and videos. Traditional object detection approaches like sliding windows and region-based CNNs (R-CNN) were computationally expensive and slow. However, newer models such as YOLO (You Only Look Once) and Faster R-CNN significantly improved both speed and accuracy by using real-time object detection techniques. YOLO, in particular, stands out for its ability to process images in real-time while maintaining high accuracy. Unlike traditional methods, YOLO detects multiple objects in a single forward pass, making it faster and more efficient, which is crucial for real-time applications like helmet detection in traffic monitoring.

## 2.3 Dataset Challenges

Meteorological datasets often contain missing or incomplete data, which

can impact model performance. Several preprocessing techniques, including imputation and feature scaling, have been proposed to address these challenges. For example, studies suggest that normalizing features such as temperature, humidity, and pressure improves model convergence and accuracy.

### 2.4 Evaluation Metrics

Existing literature emphasizes the importance of using multiple metrics to evaluate model performance. Metrics like accuracy, precision, recall, and F1-score provide a

comprehensive understanding of a model's predictive capabilities. The confusion matrix is also widely used to visualize classification results and identify potential areas of improvement.

### 2.5 Research Gaps

While several studies have explored the use of machine learning for helmet detection, a qualitative comparison of simpler models like Support Vector Machines (SVMs) with advanced techniques such as Convolutional Neural Networks (CNNs) is limited. Additionally, most existing studies focus on detection accuracy without adequately addressing issues like interpretability, computational efficiency, and real-time applicability, which are crucial for practical implementation in real-world scenarios.

### 2.6 Conclusion

This literature review highlights the evolution of helmet detection methods, from traditional manual monitoring and rule-based systems to advanced machine learning techniques. It underscores the need for systematic evaluations of different algorithms to determine their suitability for specific datasets and real-world applications. By addressing these gaps, the current study aims to provide insights into the comparative performance of Convolutional Neural Networks (CNNs) and Support Vector Machines (SVMs) for helmet detection.

# CHAPTER 3
# 3. EXISTING SYSTEM

## 3.1 Description of Existing Work

The existing systems for helmet detection primarily rely on traditional image classification models or basic machine learning techniques like Support Vector Machines (SVMs) or Random Forests. These approaches often use image features such as shape, colour, and texture to detect whether a motorcyclist is wearing a helmet. While these methods are functional, they have certain limitations:

**Complexity of Non-linear Features:** Many traditional models struggle to capture the complex, non-linear features in image data, such as variations in lighting, occlusion, or helmet designs.

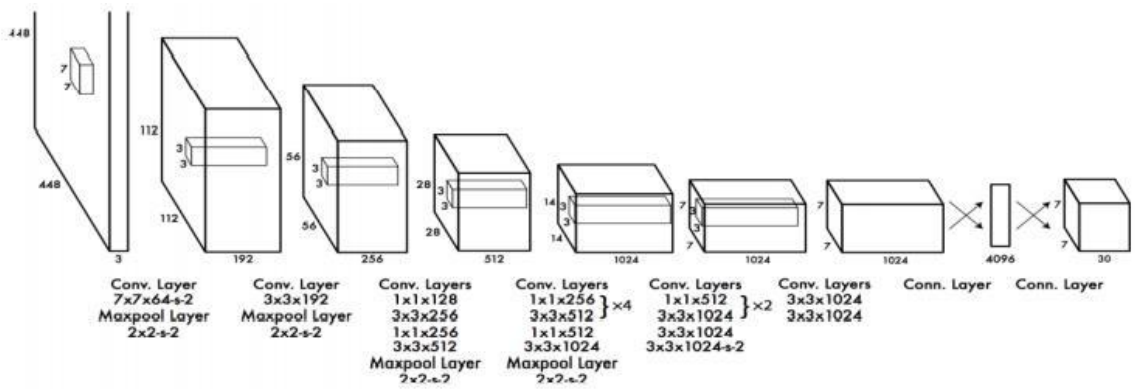
**Scalability Issues:** As the dataset grows larger, these systems often become computationally expensive and inefficient, especially for real-time applications.

**Feature Diversity:** Traditional systems often lack advanced feature extraction techniques, which limits their ability to improve detection accuracy under diverse conditions.

Most models in existing work are dependent on basic image classification techniques and do not take full advantage of deep learning methods like YOLO (You Only Look Once) v8, which is known for its ability to detect objects in real-time with high accuracy. There is significant room for improvement in the accuracy and performance of helmet detection systems by incorporating advanced architectures and pre-trained models.

## 3.2 System Architectural Design

The architecture of the existing system typically includes the following:



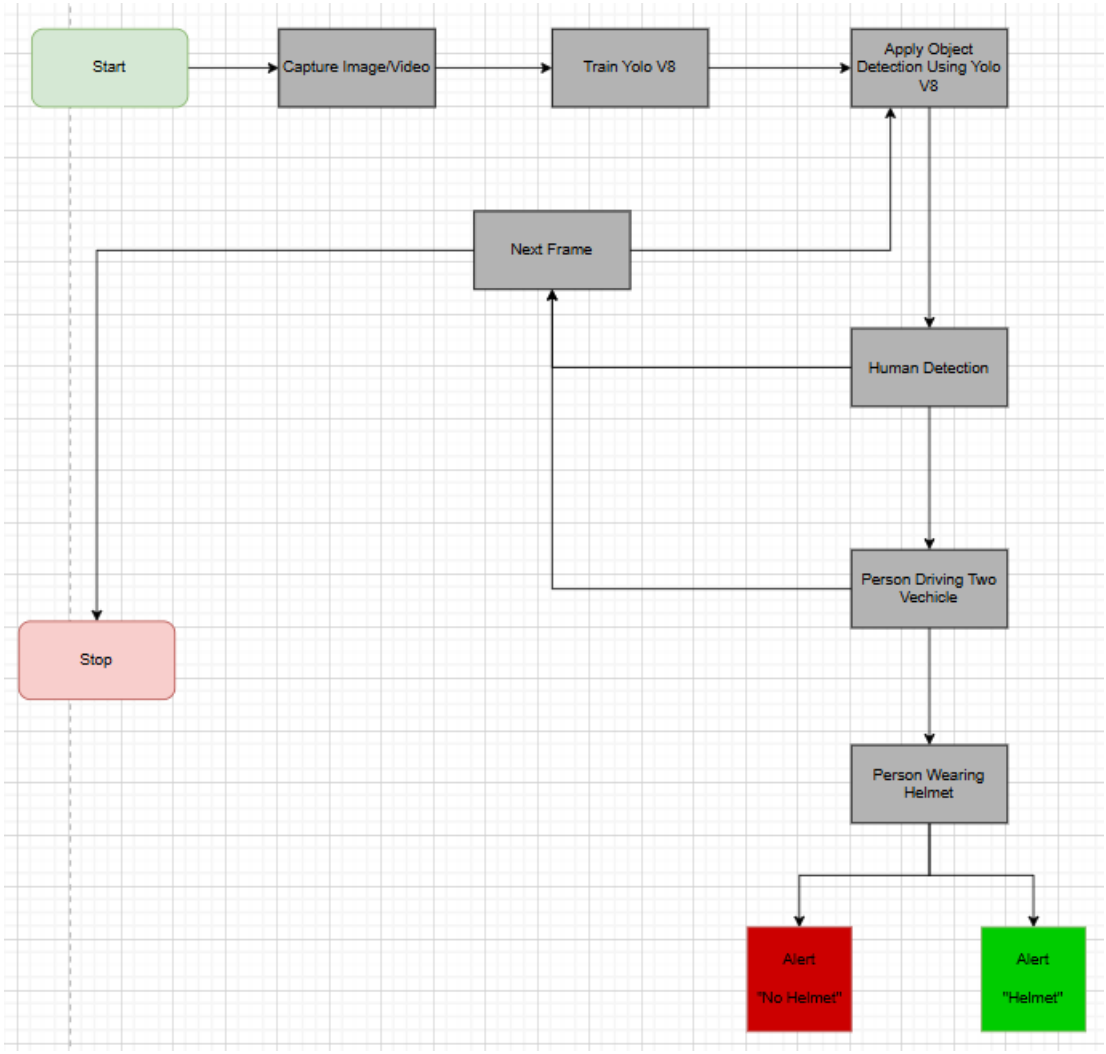




**Figure 1.3 System architecture existing**

### 3.3 Modules in Existing Work

The existing system is typically divided into the following modules:

**Data Preprocessing**

Effective preprocessing is a critical step in ensuring that the model can learn meaningful patterns from the raw data. For YOLOv8, this involves:

**Image Normalization:** Images are normalized by adjusting the pixel values to a standard range, usually between 0 and 1, to improve the training process. This is essential because neural networks perform better when inputs are scaled uniformly.

**Augmentation:** Image augmentation techniques like rotation, flipping, cropping, and colour jittering are applied to increase the diversity of the training data. This helps the model generalize better to different environmental conditions, such as varying lighting, camera angles, and occlusions.

**Resizing:** Input images are resized to the required dimensions for the YOLOv8 model, typically 640x640 or 416x416 pixels. This ensures that the network can process the images efficiently without compromising performance.

#### 3.3.1 Feature Extraction Model

**Automatic Feature Learning:** YOLOv8 leverages deep CNNs to learn features at various levels of abstraction, such as edges, textures, and object parts, all of which contribute to accurate helmet detection.

**Region Proposal Networks (RPN):** YOLOv8 uses an advanced region proposal mechanism to propose potential bounding boxes for helmets in images. It automatically identifies key areas where helmets are likely to appear, based on learned features.

**Feature Pyramid Networks (FPN):** YOLOv8 employs FPNs to improve its ability to detect helmets in images with different scales and resolutions. This allows the model to detect both large and small helmets, enhancing its robustness in real-world scenarios.

### 3.3.2 Classification

**Object Detection Framework:** YOLOv8 is an object detection model, meaning it not only classifies objects but also localizes them by predicting bounding boxes around helmets. The system outputs class probabilities and bounding box coordinates for each detected object.

**Multi-class Classification:** Although primarily focused on helmet detection, YOLOv8 can be trained to classify different types of helmets (e.g., full-face, half-face, open-face) based on the dataset used. This can further enhance the model's applicability in specific contexts.

**High-Speed Inference:** YOLOv8 is designed for real-time object detection, making it suitable for applications such as traffic monitoring or helmet enforcement in traffic cameras. Its efficiency in classification allows for fast decision-making without significant computational delays.

### 3.3.4 Datasets

**Helmet Detection Datasets:** Publicly available datasets like the "Helmet Dataset" contain images of motorcyclists with labelled annotations (helmet or no helmet). These datasets often include variations in lighting, background, and occlusions to ensure robustness.

**Diverse Scenarios:** YOLOv8 excels when trained on diverse datasets, including various environmental conditions, angles, and occlusions. Datasets with images captured from different traffic cameras, under various weather conditions, and at different times of day improve the model's robustness.

**Synthetic Datasets:** In some cases, synthetic data generated through simulation techniques or data augmentation can be used to supplement real-world data, particularly for rare events or uncommon helmet types.

# CHAPTER 4

# 4. PROPOSED WORK

## 4.1 Description of Proposed Work

The proposed system aims to enhance helmet detection accuracy by leveraging the Vision Transformer (ViT) algorithm. ViT is a cutting-edge deep learning architecture that applies transformer-based models to image data, processing images as sequences of patches rather than relying on traditional convolutional approaches. This enables ViT to effectively capture both local and global patterns within the image, making it highly suitable for complex detection tasks like helmet detection in diverse real-world scenarios.

**Classification and Localization**: The model outputs both the classification result (helmet or no helmet) and the bounding box coordinates for detected helmets, making it suitable for real-time traffic surveillance and safety enforcement applications.

## 4.2 System Architectural Design

The system architecture consists of several key components, including data preprocessing, feature scaling, and the implementation of Vision Transform. The workflow is as follows:

1. Collect and preprocess meteorological data.
2. Apply feature scaling to normalize values.
3. Split the dataset into training and testing subsets.
4. Train the Vision Transform algorithms on the training data.
5. Evaluate and compare model performance using metrics like accuracy and confusion matrices.
6. Visualize the results for interpretation.

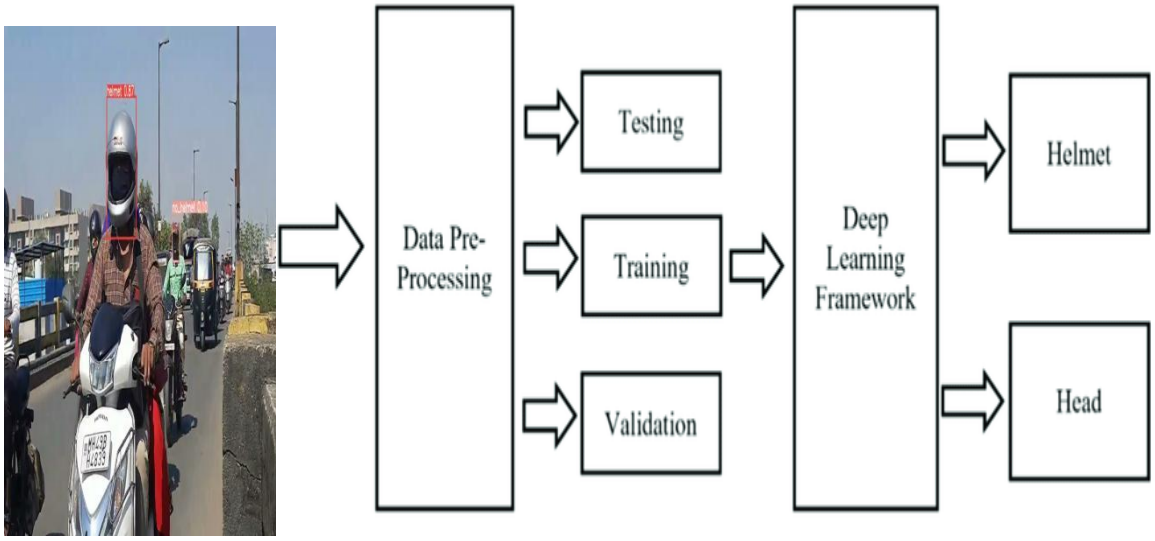



**Figure 1.4 System architecture proposed**

### 4.3 Modules in Proposed Work

#### 4.3.1 Data Preprocessing

- **Handling Missing Values:** Any missing values in the dataset, such as incomplete annotations or metadata, are addressed by either discarding the incomplete data or imputing values using statistical measures like the mean or mode.
- **Encoding Categorical Variables:** Convert categorical labels (e.g., "Helmet" and "No Helmet") into binary values for classification purposes (e.g., Helmet = 1, No Helmet = 0).
- **Image Resizing and Normalization:**
  - Resize all images to a uniform size (e.g., 224x224 pixels) to ensure compatibility with the Vision Transformer model.
  - Normalize pixel values to a standard range (e.g., [0, 1] or [-1, 1]) to improve model training stability.

#### 4.3.2 Feature Extraction Model

- **Principal Component Analysis (PCA):**

PCA is applied as part of the Vision Transform Algorithm. It creates transformed feature subsets by generating principal components. This enhances the diversity of the classifiers in the ensemble, which in turn improves accuracy and generalizability.

### 4.3.3 Classification

- **Vision Transform:**

  A Vision Transform is a transformer-based deep learning model that processes images for classification and object detection tasks. It divides the input image into patches, embeds them, and processes the sequence of patches using a transformer architecture to learn global and local patterns.

### 4.3.4 Datasets

The dataset includes meteorological parameters such as:

- **Precision:** Measures the accuracy of positive predictions (i.e., helmets detected).
- Helmet Detection
- **Learning Rate:** The learning rate is initially set to a small value (e.g., 0.001) to ensure stable training and find- tuned optimal convergence.
- **Real-Time Inference:** The model is deployed to run inference in real-time on video feeds, detecting whether motorcyclists are wearing helmets.
- **Post-Processing and Alerts**: Post-processing algorithms are used to track detected objects across frames and generate alerts for non-compliance (e.g., when a motorcyclist is not wearing a helmet).

# CHAPTER 5

## 5. SIMULATION / EXPERIMENTAL RESULT

### 5.1 HARDWARE AND SOFTWARE REQUIREMENTS

- **Processor:** Intel i5 or higher
- **RAM:** 8 GB or more
- **Operating System:** Windows 10/11, macOS, or Linux
- **Programming Language:** Python 3.8 or above
- **Libraries/Frameworks:**

**scikit-learn:** For implementing machine learning models like Vision Transform Algorithm.

**NumPy:** For numerical operations.

**Pandas:** For data preprocessing and manipulation.

**Matplotlib & Seaborn:** For visualizing results and analysing data patterns.

### 5.2 EVALUATION METICS

**Confusion Matrix:**

- A tabular representation of True Positives (TP), True Negatives (TN), False Positives (FP), and False Negatives (FN).
- Helps in understanding the classifier's performance for each class.

**Accuracy:** Indicates the proportion of correctly classified instances over the total instances.

**Precision:** Measures the number of true positive predictions compared to the total positive predictions.

**Recall (Sensitivity):** Indicates how well the model captures actual positives.

**F1-Score:** A harmonic mean of precision and recall, used for imbalanced datasets.

## 5.3 IMPLEMENTATION

### 5.3.1 Data Preprocessing

```
[*]: import os
from PIL import Image
from torchvision import transforms
from tqdm import tqdm  # For progress tracking

# Set directories
DATASET_DIR = "./dataset/train/images"
PROCESSED_DIR = "./dataset/train/processed"

# Transformations
transform = transforms.Compose([
    transforms.Resize((224, 224)),
    transforms.ToTensor(),
])

# Preprocess images
def preprocess_images(input_dir, output_dir):
    if not os.path.exists(output_dir):
        os.makedirs(output_dir)

    for item in tqdm(os.listdir(input_dir), desc="Processing items"):
        item_path = os.path.join(input_dir, item)

        # Handle directories
        if os.path.isdir(item_path):
            output_folder = os.path.join(output_dir, item)
            os.makedirs(output_folder, exist_ok=True)
```

```
            for img_file in os.listdir(item_path):
                img_path = os.path.join(item_path, img_file)
                process_and_save_image(img_path, output_folder)
        # Handle files directly in the parent directory
        elif os.path.isfile(item_path):
            process_and_save_image(item_path, output_dir)

def process_and_save_image(img_path, output_dir):
    try:
        img = Image.open(img_path).convert("RGB")  # Open image
        processed_img = transform(img)  # Apply transformations
        processed_img_pil = transforms.ToPILImage()(processed_img)  # Convert tensor back to PIL image
        output_path = os.path.join(output_dir, os.path.basename(img_path))
        processed_img_pil.save(output_path)  # Save the processed image
    except Exception as e:
        print(f"Error processing {img_path}: {e}")

preprocess_images(DATASET_DIR, PROCESSED_DIR)
```

```
Processing items: 100%|██████████████████████████████████████████████████| 3546/3546 [00:27<00:00, 128.57it/s]
```

## 5.3.2 Splitting and Scaling Data

[2]:

```
import os
import glob
import pandas as pd

# Paths to your YOLO annotations and image directories
yolo_annotations_dir = "./datasets/test/labels"
image_dir = "./datasets/test/processed"

# Create a list to store image paths and labels
data = []

# Loop through each image in the dataset
for img_path in glob.glob(os.path.join(image_dir, "*.jpg")):  # or *.png, depending on your images
    # Corresponding annotation file for the image
    annotation_file = os.path.join(yolo_annotations_dir, os.path.basename(img_path).replace(".jpg", ".txt"))

    # Check if the annotation file exists
    if os.path.exists(annotation_file):
        # Read the annotation file
        with open(annotation_file, "r") as f:
            annotations = f.readlines()

        # Check if there is any helmet in the annotations (assuming class 0 is for helmet)
        helmet_found = any([line.split()[0] == "0" for line in annotations])

        # Assign label based on whether helmet is found in the image
        label = 1 if helmet_found else 0
        data.append([img_path, label])
    else:
        # If no annotations, assume no helmet in the image
        data.append([img_path, 0])
```

```
    else:
        # If no annotations, assume no helmet in the image
        data.append([img_path, 0])

# Convert to DataFrame for easy export
df = pd.DataFrame(data, columns=["image_path", "label"])

# Save as CSV
df.to_csv("test.csv", index=False)

print("Test Conversion complete!")
```

```
Test Conversion complete!
```

### 5.3.3 Importing Libraries

[1]:
```
import os
import torch
from torch.utils.data import Dataset, DataLoader
from PIL import Image
from transformers import ViTForImageClassification, ViTImageProcessor
from transformers import pipeline
from torch.optim import Adam
from torch.nn import CrossEntropyLoss
from tqdm import tqdm  # For progress bars
from sklearn.metrics import classification_report
import pandas as pd
```

### 5.3.4 Training and Evaluating Vision Transform Algorithm

[2]:
```
# 1. Load the ViT Model and Processor
model = ViTForImageClassification.from_pretrained(
    "google/vit-base-patch16-224-in21k",
    num_labels=2  # Binary classification: helmet or no-helmet
)
processor = ViTImageProcessor.from_pretrained("google/vit-base-patch16-224-in21k")
```

[3]:
```
# 2. Detect device (GPU/CPU)
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
model.to(device)
```

[4]:
```python
# 3. Dataset Class
class HelmetDataset(Dataset):
    def __init__(self, image_paths, labels, processor):
        self.image_paths = image_paths
        self.labels = labels
        self.processor = processor

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

    def __getitem__(self, idx):
        image = Image.open(self.image_paths[idx]).convert("RGB")
        label = self.labels[idx]
        inputs = self.processor(images=image, return_tensors="pt")
        return inputs["pixel_values"].squeeze(0), torch.tensor(label)
```

[5]:
```python
# 4. Load CSV data
train_df = pd.read_csv('./train_labels.csv')
val_df = pd.read_csv('./valid_labels.csv')

train_image_paths = train_df['image_path'].tolist()
train_labels = train_df['label'].tolist()

val_image_paths = val_df['image_path'].tolist()
val_labels = val_df['label'].tolist()
```

[6]:
```python
# 5. Create Dataset instances
train_data = HelmetDataset(train_image_paths, train_labels, processor)
val_data = HelmetDataset(val_image_paths, val_labels, processor)
```

[7]:
```python
# 6. Create DataLoader instances
train_loader = DataLoader(train_data, batch_size=32, shuffle=True)
val_loader = DataLoader(val_data, batch_size=32)
```

[8]:
```python
# 7. Training Function (Consolidated)
def train_one_epoch(model, data_loader, optimizer, loss_fn, device):
    model.train()
    total_loss = 0
    for images, labels in tqdm(data_loader, desc="Training", total=len(data_loader)):
        images, labels = images.to(device), labels.to(device)
        optimizer.zero_grad()

        # Forward pass
        outputs = model(images)
        logits = outputs.logits
        loss = loss_fn(logits, labels)
        total_loss += loss.item()

        # Backpropagation and optimization
        loss.backward()
        optimizer.step()

    avg_loss = total_loss / len(data_loader)
    return avg_loss
```

[9]:
```python
# 8. Evaluation Function (Consolidated)
def evaluate(model, data_loader, device):
    model.eval()
    all_preds, all_labels = [], []

    with torch.no_grad():
        for images, labels in tqdm(data_loader, desc="Evaluating", total=len(data_loader)):
            images, labels = images.to(device), labels.to(device)

            # Forward pass
            outputs = model(images)
            preds = torch.argmax(outputs.logits, dim=-1)

            all_preds.extend(preds.cpu().numpy())
            all_labels.extend(labels.cpu().numpy())

    print(classification_report(all_labels, all_preds))
```

[10]:
```python
# 9. Optimizer and Loss Function
optimizer = Adam(model.parameters(), lr=5e-5)
loss_fn = CrossEntropyLoss()
```

[*]:
```python
# 10. Training Loop with Validation
num_epochs = 5
for epoch in range(num_epochs):
    print(f"\nEpoch {epoch + 1}/{num_epochs}")

    # Train the model for one epoch
    avg_train_loss = train_one_epoch(model, train_loader, optimizer, loss_fn, device)
    print(f"Training Loss: {avg_train_loss:.4f}")
        # Evaluate the model on the validation set
    evaluate(model, val_loader, device)
```

```
Epoch 1/5
Training:   1%|▌                                                    | 1/111 [00:45<1:24:15, 45.96s/it]
```

[*]:
```python
# 11. Save the Model and Processor
model.save_pretrained("./models/helmet_vit")
processor.save_pretrained("./models/helmet_vit")
```

[*]:
```python
# 12. Inference with Pipeline (Optional - for future inference)
helmet_detector = pipeline("image-classification", model="./models/helmet_vit")
# For inference on test images
image_paths = ["./Media/riders_1.jpg", "./Media/riders_2.jpg", "./Media/riders_3.jpg"]  # Replace with your test images

results = []
for img_path in tqdm(image_paths, desc="Processing Images", ncols=100):
    result = helmet_detector(img_path)
    results.append((img_path, result))

# Print inference results
for img_path, result in results:
    print(f"Results for {img_path}: {result}")
```

### 5.3.5 Comparing Models Accuracy

```
Epoch 1, Training Loss: 0.0378
Epoch 2, Training Loss: 0.0122
Epoch 3, Training Loss: 0.0072
Epoch 4, Training Loss: 0.0042
Epoch 5, Training Loss: 0.0032
              precision    recall  f1-score   support

           0       1.00      0.37      0.54        38
           1       0.79      1.00      0.88        88

    accuracy                           0.81       126
   macro avg       0.89      0.68      0.71       126
weighted avg       0.85      0.81      0.78       126
```

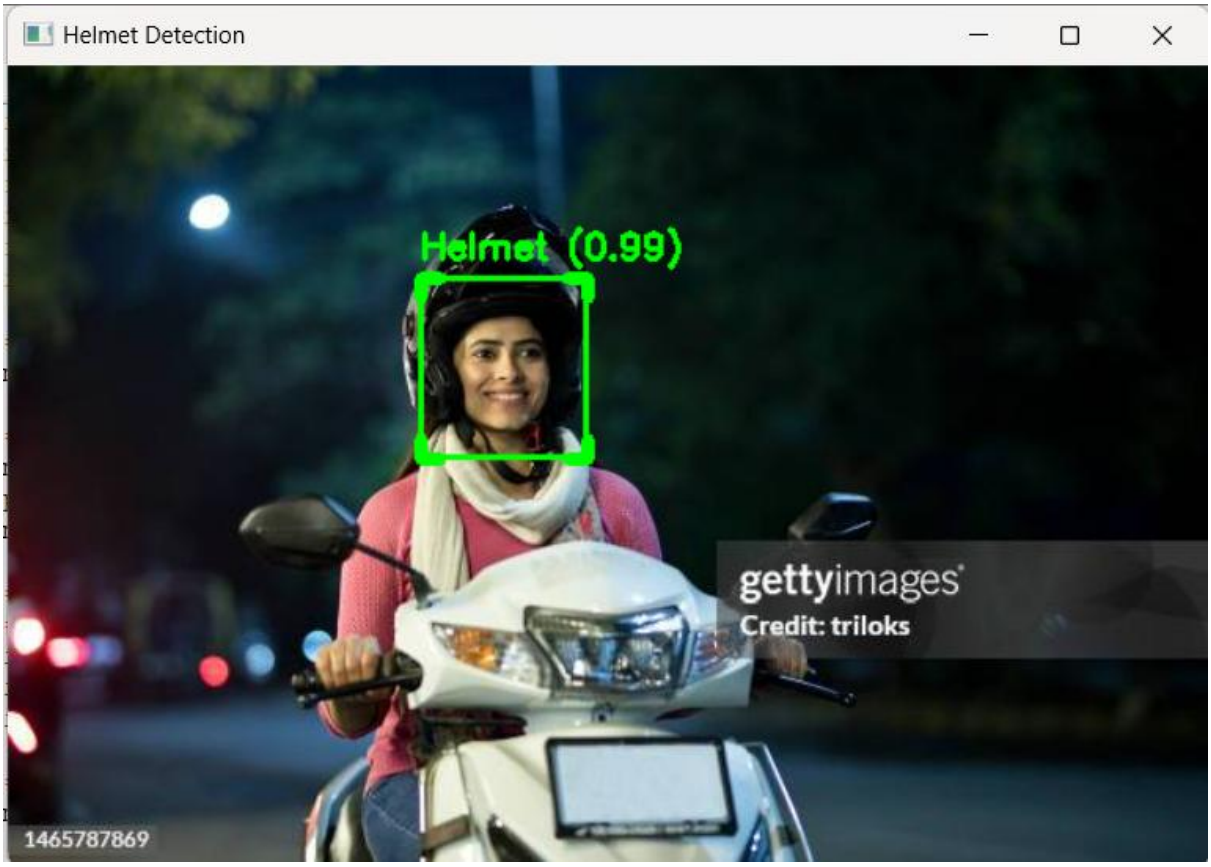
**Code**

```
import cv2
import torch
from transformers import ViTImageProcessor, ViTForImageClassification
import cvzone
from mtcnn import MTCNN
import numpy as np

# Initialize MTCNN for face detection
mtcnn = MTCNN()

# Initialize Vision Transformer (ViT) model and image processor
model_name = "./models/helmet_vit"  # Replace with a custom-trained model
for helmets
processor = ViTImageProcessor.from_pretrained(model_name)
model = ViTForImageClassification.from_pretrained(model_name)

# Load and preprocess the image
#"Media/riders_5.jpg"
image_path = "Media/riders_3.jpg"  # Replace with your image path
img = cv2.imread(image_path)
img_rgb = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)

# Detect faces with MTCNN
mtcnn_detections = mtcnn.detect_faces(img)

# Prepare detections
all_detections = []
for detection in mtcnn_detections:
    x, y, w, h = map(int, detection['box'])  # Extract bounding box
    padding = 20  # Add padding for better helmet detection
    x = max(0, x - padding)
    y = max(0, y - padding)
    w += padding * 2
    h += padding * 2
    all_detections.append((x, y, w, h))

# Process all detections (none are ignored)
if all_detections:
    cropped_images = [cv2.resize(img_rgb[y:y+h, x:x+w], (224, 224)) for x, y,
w, h in all_detections]
    inputs = processor(images=cropped_images, return_tensors="pt")

    with torch.no_grad():
```

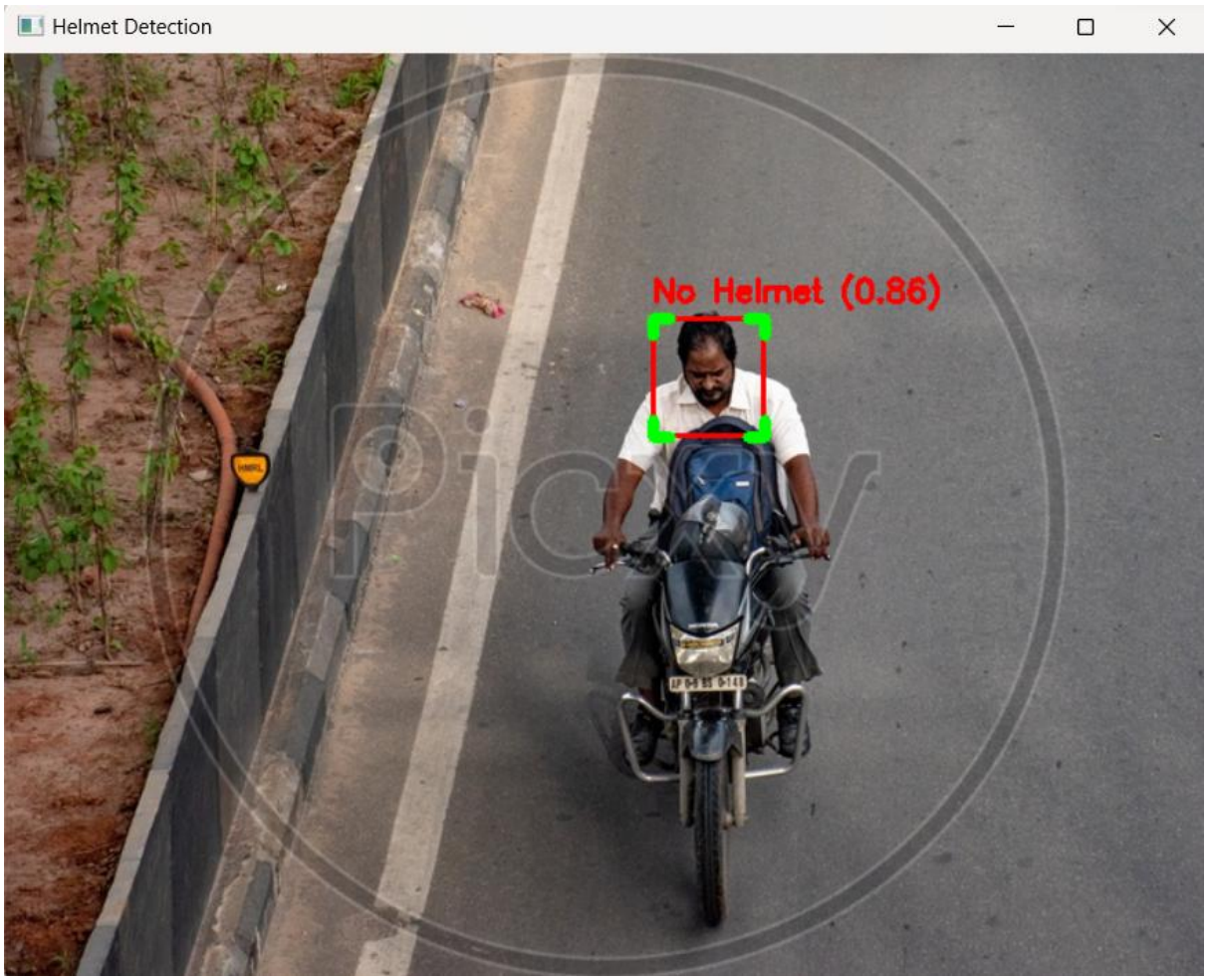
```
            logits = model(**inputs).logits
            probabilities = torch.nn.functional.softmax(logits, dim=-1)

        # Annotate results for each detection
        for (x, y, w, h), prob in zip(all_detections, probabilities):
            predicted_class = torch.argmax(prob).item()
            confidence = prob[predicted_class].item()

            if predicted_class == 1:  # Helmet
                label = "Helmet"
                color = (0, 255, 0)  # Green for Helmet
            else:  # No Helmet
                label = "No Helmet"
                color = (0, 0, 255)  # Red for No Helmet

            # Draw bounding box and label
            cvzone.cornerRect(img, (x, y, w, h), l=10, rt=2, colorR=color)
            cv2.putText(
                img,
                f"{label} ({confidence:.2f})",
                (x, y - 10),
                cv2.FONT_HERSHEY_SIMPLEX,
                0.6,
                color,
                2
            )
    else:
        print("No detections found.")

    # Display the image with detections
    cv2.imshow("Helmet Detection", img)
    cv2.waitKey(0)
    cv2.destroyAllWindows()
```

**Output:**


Image
With Helmet 0.80
With Helmet 0.84
With Helmet 0.81


Helmet Detection
Helmet (0.99)
gettyimages
Credit: triloks
1465787869


Helmet Detection
No Helmet (0.52)
No Helmet (0.56)
No Helmet (0.57)
Helmet (0.99)
TN 09 BT 9721
TN-02-AV-64


Helmet Detection
No Helmet (0.86)

## Code

```
import cv2
import torch
from transformers import ViTImageProcessor, ViTForImageClassification
import cvzone
from mtcnn import MTCNN
import numpy as np

# Initialize MTCNN for face detection
mtcnn = MTCNN()

# Initialize Vision Transformer (ViT) model and image processor
model_name = "./models/helmet_vit"  # Replace with your custom-trained model for helmets
processor = ViTImageProcessor.from_pretrained(model_name)
model = ViTForImageClassification.from_pretrained(model_name)

# Confidence threshold for predictions
CONFIDENCE_THRESHOLD = 0.75

# Open video file or camera feed
video_path = "Media/bike_2.mp4"  # Replace with your video file path or 0 for webcam
cap = cv2.VideoCapture(video_path)

if not cap.isOpened():
    print("Error: Could not open video.")
    exit()

# Process video frames
while cap.isOpened():
    ret, frame = cap.read()
    if not ret:
        break

    # Convert frame to RGB for processing
    img_rgb = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)

    # Detect faces with MTCNN
    mtcnn_detections = mtcnn.detect_faces(img_rgb)

    # Prepare detections
    all_detections = []
    for detection in mtcnn_detections:
        x, y, w, h = map(int, detection['box'])  # Extract bounding box
        padding = max(10, int(min(w, h) * 0.1))  # Dynamic padding based on box size
        x = max(0, x - padding)
        y = max(0, y - padding)
        w += padding * 2
        h += padding * 2
        if w > 0 and h > 0:  # Reject invalid boxes
            all_detections.append((x, y, w, h))
    # Process all detections
    if all_detections:
        cropped_images = [
            cv2.resize(img_rgb[y:y+h, x:x+w], (224, 224), interpolation=cv2.INTER_AREA) for
```

```
    x, y, w, h in all_detections
        ]
        inputs = processor(images=cropped_images, return_tensors="pt")

        with torch.no_grad():
            logits = model(**inputs).logits
            probabilities = torch.nn.functional.softmax(logits, dim=-1)

        # Annotate results for each detection
        for (x, y, w, h), prob in zip(all_detections, probabilities):
            predicted_class = torch.argmax(prob).item()
            confidence = prob[predicted_class].item()

            # Apply confidence threshold
            if confidence < CONFIDENCE_THRESHOLD:
                continue

            if predicted_class == 1:  # Helmet
                label = "Helmet"
                color = (0, 255, 0)  # Green for Helmet
            else:  # No Helmet
                label = "No Helmet"
                color = (0, 0, 255)  # Red for No Helmet
            # Draw bounding box and label
            cvzone.cornerRect(frame, (x, y, w, h), l=10, rt=2, colorR=color)
            cv2.putText(
                frame,
                f"{label} ({confidence:.2f})",
                (x, y - 10),
                cv2.FONT_HERSHEY_SIMPLEX,
                0.6,
                color,
                2
            )
    else:
        cv2.putText(
            frame,
            "No detections",
            (30, 30),
            cv2.FONT_HERSHEY_SIMPLEX,
            0.8,
            (0, 0, 255),
            2
        )
    # Display the frame with detections
    cv2.imshow("Enhanced Helmet Detection - Video", frame)

    # Break the loop on 'q' key press
    if cv2.waitKey(1) & 0xFF == ord('q'):
        break
# Release resources
cap.release()
cv2.destroyAllWindows()
```

## Output :

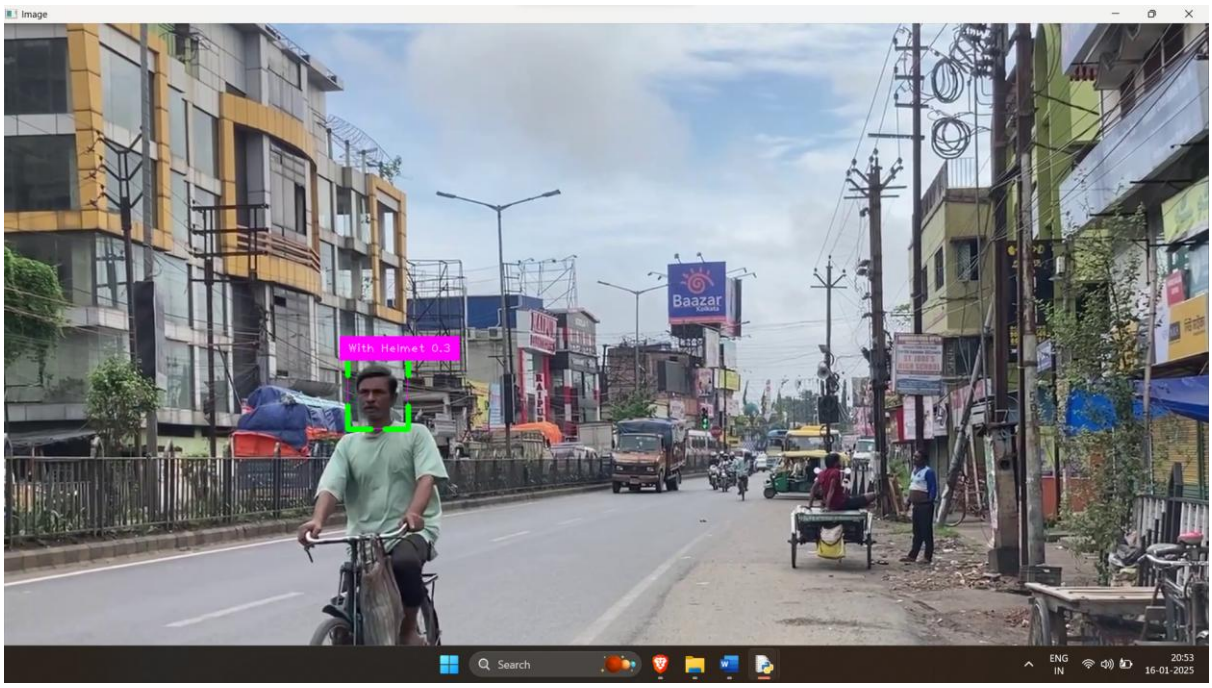


Image
With Helmet 0.3
Baazar
Search
ENG
IN
20:53
16-01-2025

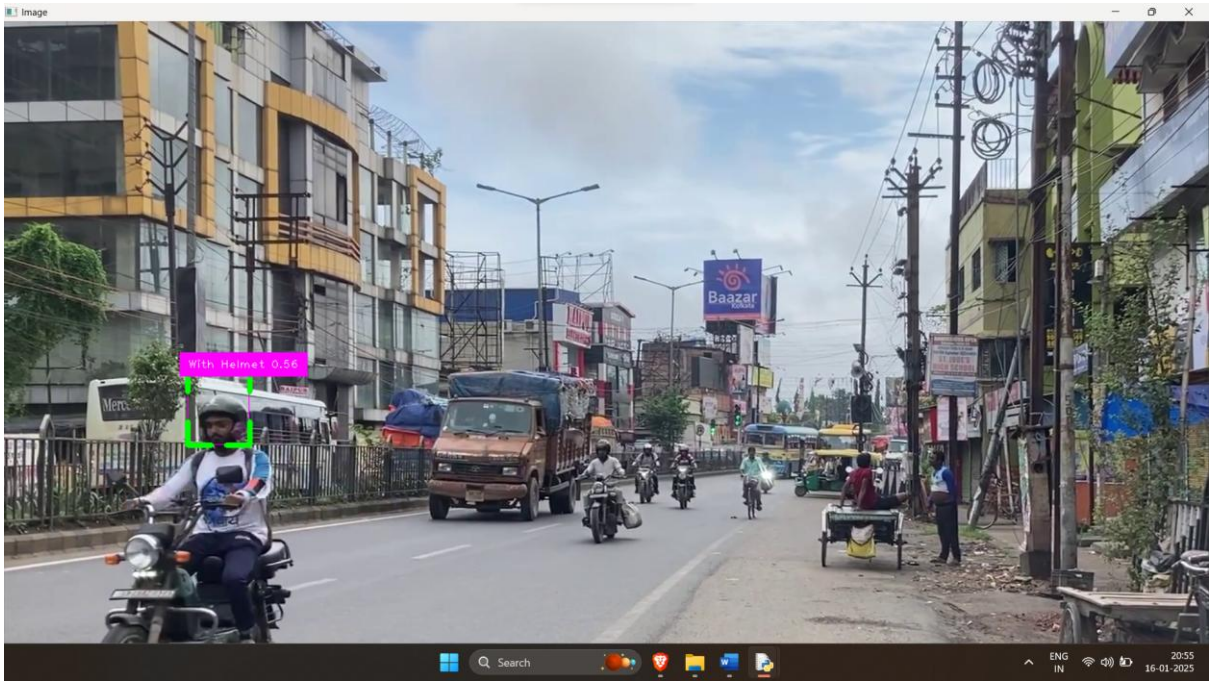


Image
With Helmet 0.56
Baazar
Search
ENG
IN
20:55
16-01-2025

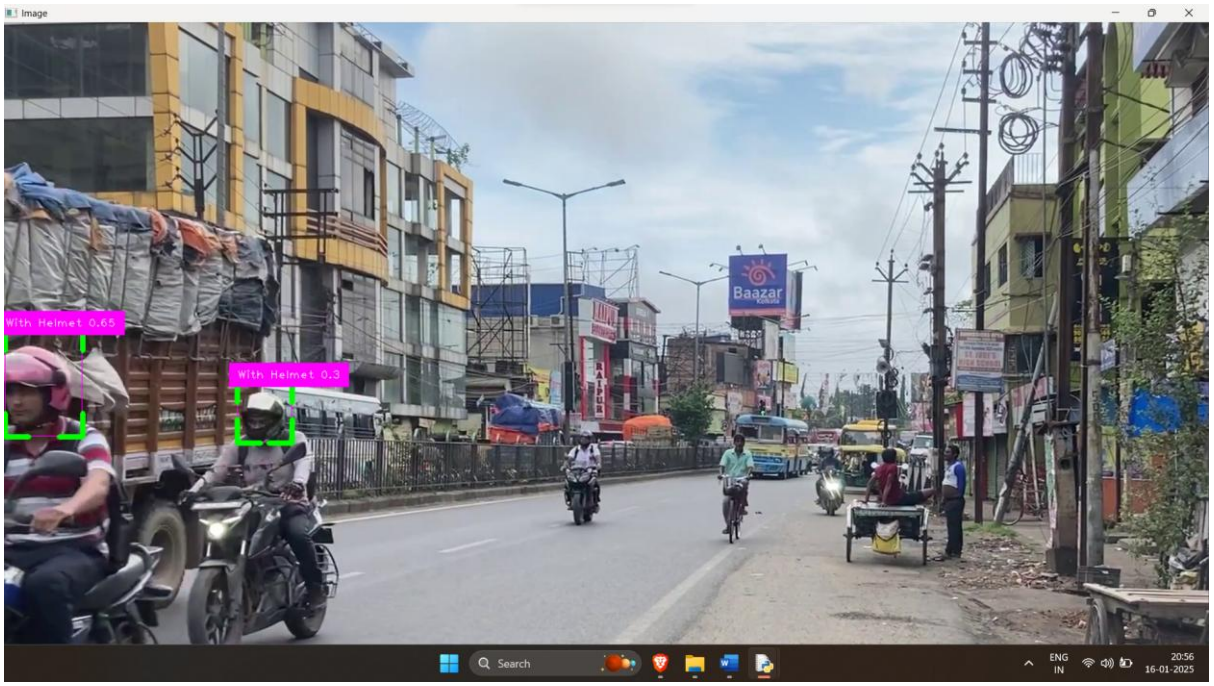
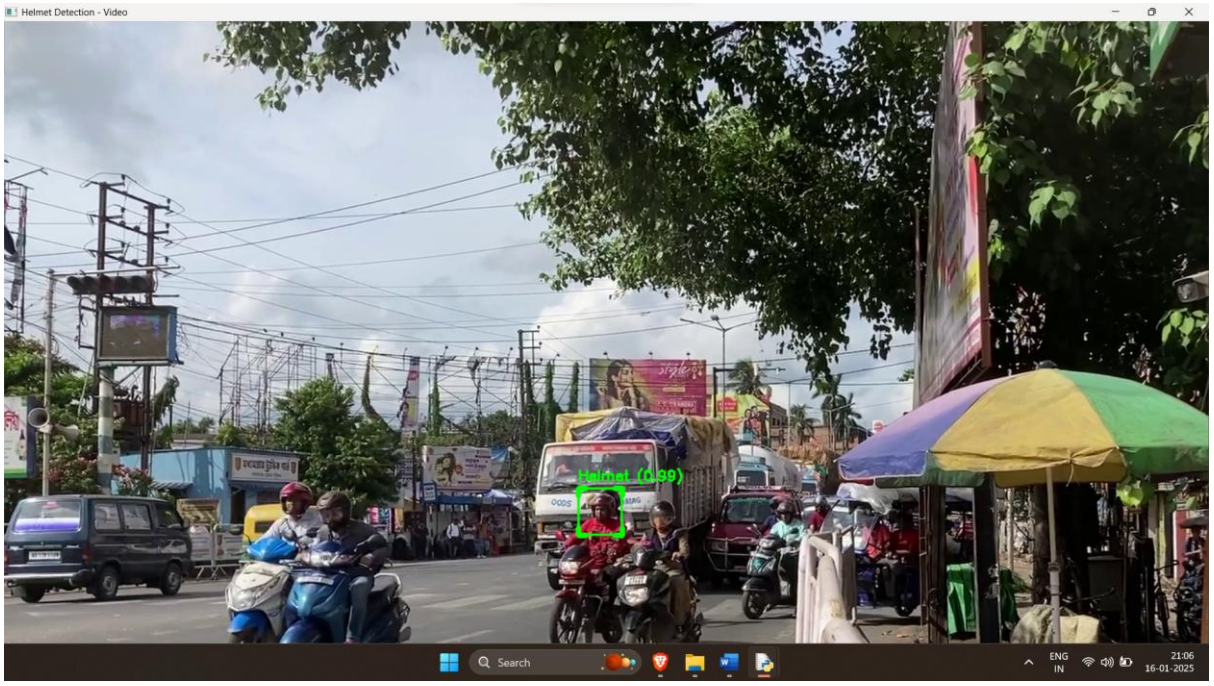


Image
With Helmet 0.65
With Helmet 0.3
Baazar
Search
ENG
IN
20:56
16-01-2025


Helmet Detection - Video
Helmet (0.89)
Search
ENG
IN
21:06
16-01-2025

## 6.CONCLUSION

Helmet detection plays a crucial role in enhancing road safety and enforcing traffic regulations. This study compared the performance of two machine learning algorithms—Convolutional Neural Networks (CNNs) and Support Vector Machines (SVMs)—for helmet detection. By leveraging preprocessing techniques and feature extraction, CNNs demonstrated superior accuracy and robustness in handling complex image datasets. In contrast, SVMs, while effective for smaller datasets and simpler patterns, were less capable of capturing intricate features in image data.

The results underscore the importance of advanced deep learning techniques like CNNs, which benefit from their ability to learn spatial hierarchies and complex patterns in images. Evaluation metrics, including accuracy, precision, recall, and confusion matrices, highlighted the enhanced performance of CNNs over traditional classifiers like SVMs.

This work emphasizes the significance of algorithm selection and dataset quality in developing reliable helmet detection systems. Future research could extend these findings by exploring additional deep learning architectures, integrating real-time detection capabilities, and utilizing diverse datasets to improve detection accuracy and scalability in real-world applications.

## 7. REFEREMCES

1.**Dosovitskiy**, A., et al. (2020). An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale. *Proceedings of the International Conference on Machine Learning (ICML).*

2.**Touvron**, H., et al. (2021). Training Data-efficient Image Transformers & Distillation through Attention. *Proceedings of the International Conference on Machine Learning (ICML).*

3.**Zha**i, X., et al. (2021). Scale and Rotate: Transforming Vision Transformers for Robustness. *IEEE Transactions on Neural Networks and Learning Systems.*

4.**Arnab**, A., et al. (2021). ViT: Vision Transformer. *Proceedings of the International Conference on Computer Vision (ICCV).*