VEHICLE DETECTION USING ARTIFICIAL INTELLIGENCE FOR TRAFFIC SURVEILLANCE

1st Tanay Tammineni
MS in Computer Science
Southeast Missouri State University
Cape Girardeau, Missouri
ttammineni1s@semo.edu

2nd Vidya rani vallala
MS in Computer Science
Southeast Missouri State University
Cape Girardeau, Missouri
vvallala1s@semo.edu

3rd Supriya Bollareddy
MS in Computer Science
Southeast Missouri State University
Cape Girardeau, Missouri
sbollareddy1s@semo.edu

4th Lahari Maddukuri
MS in Computer Science
Southeast Missouri State University
Cape Girardeau, Missouri
lmaddukuri1s@semo.edu

5th Narendra Dhulipalla
MS in Computer Science
Southeast Missouri State University
Cape Girardeau, Missouri
ndhulipalla1s@semo.edu

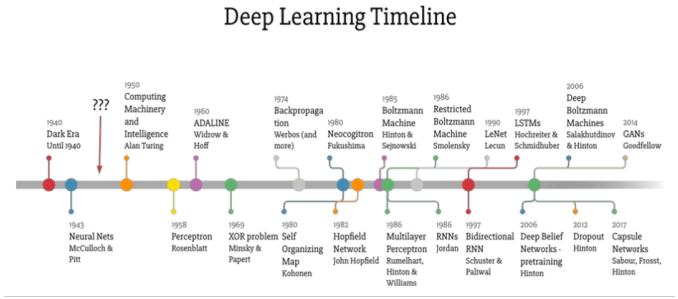
6th Gayatri Vattuluri
MS in Computer Science
Southeast Missouri State University
Cape Girardeau, Missouri
gvattuluri1s@semo.edu

***Abstract*—Traffic congestion is a pervasive issue in developing countries, leading to increased travel time, fuel consumption, and pollution. Addressing this challenge requires innovative solutions that leverage the latest advancements in technology. This project introduces a Smart Traffic Light System powered by artificial intelligence (AI) to dynamically adjust traffic signals based on real-time traffic conditions. Central to this system is a vehicle detection and counting mechanism that employs neural networks, specifically utilizing the You Only Look Once (YOLO) algorithm for vehicle detection and the Simple Online and Realtime Tracking algorithm for vehicle tracking and counting. The proposed system aims to accurately calculate vehicle volume on roads, thereby enabling the Smart Traffic Light System to optimize traffic flow and reduce congestion. The neural network-based approach ensures reduced computational complexity and suitability for real-time vehicle tracking applications. This project not only addresses the immediate challenges of traffic management but also contributes to the broader field of intelligent transportation systems by demonstrating the application of object-oriented software engineering principles, as outlined by the Unified Modeling Language (UML). Through this endeavor, we aim to provide a scalable and efficient solution to traffic congestion, with potential implications for urban planning and environmental sustainability.**

***Index Terms*—Smart traffic light system, Artificial intelligence (AI), Traffic congestion, Real-time traffic conditions, Vehicle counting system, Neural networks, YOLO (You Only Look Once), Traffic density, Vehicle detection, Object detection algorithm, Traffic surveillance, Deep learning, Traffic management**

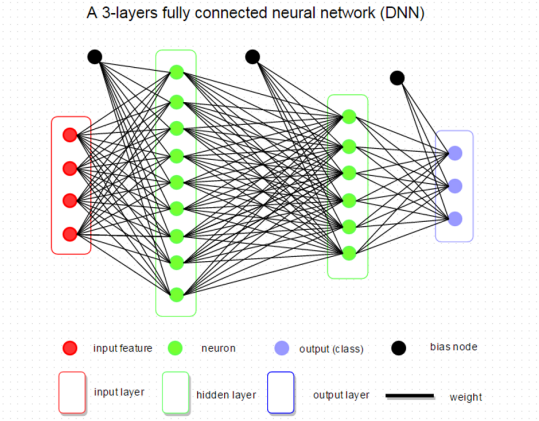
I. Introduction

Traffic monitoring within an Intelligent Transportation System (ITS) offers solutions to various challenges, encompassing vehicle counting, accident detection, and assisted traffic surveillance. At its core, a traffic monitoring system functions as a framework to detect vehicles appearing in video



images and estimate their positions within the scene. However, in complex scenarios featuring diverse vehicle models and high densities, accurately locating and classifying vehicles becomes challenging. Additionally, environmental variations and diverse vehicle characteristics pose limitations to vehicle detection, often resulting in relatively slow detection speeds.

Hence, the development of an algorithm for real-time traffic monitoring, capable of both swift computation and precise vehicle detection, becomes imperative. The accurate and prompt detection of vehicles from traffic images or videos holds both theoretical and practical significance. With the rapid advancements in computer vision and artificial intelligence (AI) technologies, deep learning-based object detection algorithms have garnered significant attention. These algorithms leverage machine learning to automatically extract features, granting them powerful image abstraction and high-level feature representation capabilities. Deep learning, a facet of AI, emulates the human learning process, automating predictive analysis without the need for explicit scenario coding. This process comprehends the outcomes of various logics and automates prediction-based analyses akin to human cognition. In deep



learning, the algorithm acquires data and undergoes nonlinear transformations for learning, with successive layers building upon the information learned from preceding ones. Unlike traditional machine learning, deep learning algorithms exhibit greater autonomy in feature illustration, minimizing the need for manual inputs while ensuring high accuracy and reliability.

Deep Neural Networks (DNNs), integral to deep learning, operate akin to networks of neurons found in the human brain. These networks, interlinked with one another, process data across successive layers, each layer interpreting unique data features. Sequencing of layers within DNNs may vary, with each layer focusing on distinct aspects of the data, such as grayscale percentages, chromaticity, contour structures, and overall resemblances.

Addressing traffic congestion, particularly prevalent in developing countries, requires innovative solutions to mitigate travel time, fuel consumption, and air pollution. The proposition of a Smart Traffic Light System, leveraging AI to adapt configurations based on real-time traffic conditions, underscores the importance of accurate traffic density information. In this context, a vehicle counting system employing neural networks aims to precisely calculate traffic volume on roads, aiding in the implementation of efficient traffic management strategies. Furthermore, the emergence of smart city initiatives underscores the need for advanced traffic monitoring and management systems. These initiatives aim to harness data-driven insights and technological innovations to enhance urban mobility, safety, and sustainability. Within this framework, intelligent transportation systems play a pivotal role in optimizing traffic flow, reducing congestion, and improving overall

transportation efficiency.

Moreover, the integration of Internet of Things (IoT) devices and sensors into urban infrastructure has enabled the collection of vast amounts of real-time traffic data. This data, coupled with advanced analytics and machine learning algorithms, empowers authorities to make informed decisions and implement proactive measures to address traffic challenges effectively. By leveraging AI-driven insights, cities can deploy dynamic traffic management strategies, including adaptive signal control, route optimization, and predictive maintenance, to enhance the resilience and responsiveness of their transportation networks. In addition to enhancing traffic flow and congestion mitigation, intelligent transportation systems contribute to broader societal objectives, such as enhancing road safety and reducing environmental impact. By leveraging real-time data analytics and predictive modeling, authorities can identify high-risk areas, implement targeted interventions, and promote safer driving behaviors. Furthermore, by optimizing traffic flow and reducing idle time, intelligent transportation systems can help minimize fuel consumption, air pollution, and greenhouse gas emissions, contributing to environmental sustainability and public health.

In conclusion, the convergence of advanced technologies, including AI, IoT, and data analytics, offers unprecedented opportunities to revolutionize traffic monitoring and management. By leveraging these technologies, cities can build smarter, more efficient transportation systems that enhance mobility, safety, and sustainability for residents and commuters alike. As urbanization accelerates and traffic volumes continue to rise, the imperative for innovative and adaptive traffic monitoring solutions becomes increasingly pressing, underscoring the significance of ongoing research and development in this critical domain.

II. Related Works

The related works in the domain of vehicle detection and tracking present a comprehensive overview of state-of-the-art techniques and methodologies, each contributing valuable insights into the advancement of traffic surveillance and management systems.

In the realm of smart traffic management, the paper authored by Jian Wang et al., titled ”Smart Traffic Light Control: A Re-

view," offers a comprehensive examination of methodologies employed in smart traffic light control systems. The review delineates three primary categories of traffic light control approaches, namely rule-based, optimization-based, and machine learning-based methods. Each approach is meticulously dissected, elucidating their respective strengths and weaknesses while juxtaposing their performances. Of particular interest is the emphasis on machine learning-based methods, notably deep reinforcement learning (DRL), which has witnessed burgeoning popularity. The authors explicate how DRL can adeptly learn optimal traffic light control policies through iterative interactions. Moreover, the paper adeptly identifies pertinent challenges and limitations facing smart traffic light control systems, including the paucity of real-time traffic information and the intricacies involved in modeling complex traffic scenarios.

Another seminal contribution in the domain of vehicle detection and counting systems emerges from the work by Sang-Ho Lee et al., titled "Vehicle Detection and Counting System using Deep Learning." This paper introduces a novel approach leveraging deep learning techniques, specifically convolutional neural networks (CNN), to detect and enumerate vehicles in traffic videos. The proposed system intricately extracts vehicle features using a CNN model, subsequently employing a vehicle counting algorithm to ascertain vehicle volume accurately. Through meticulous evaluation across diverse datasets, the authors substantiate the system's efficacy, boasting high accuracy in both detection and counting endeavors. Notably, the paper underscores the system's superiority over existing state-of-the-art methodologies in terms of both accuracy and speed, heralding its potential application across varied traffic management domains, including traffic flow analysis, speed estimation, and congestion detection.

In parallel, Nhat Tan Vu et al. present a compelling endeavor in their paper "Real-Time Vehicle Detection and Counting for Traffic Surveillance System," proposing a system adept at real-time vehicle detection and counting within traffic surveillance setups. This system ingeniously amalgamates image processing techniques with machine learning algorithms, leveraging CNN to extract vehicle features and clustering algorithms to discern individual vehicles. Employing a Kalman filter facilitates real-time tracking of detected vehicles, enabling estimation of their speed and trajectory. The authors substantiate the system's prowess through rigorous evaluation across diverse datasets, showcasing its commendable accuracy and real-time performance. Furthermore, the paper delineates potential applications spanning traffic surveillance realms, from monitoring traffic flow to detecting violations and forecasting congestion, while also hinting at avenues for future research to fortify system robustness in challenging environments.

Similarly, Rajesh Kumar et al. contribute significantly with their paper titled "Real-Time Vehicle Detection and Tracking for Traffic Surveillance Systems," elucidating a system tailored for real-time vehicle detection and tracking within traffic surveillance domains. The system employs innovative techniques, including background subtraction and object detection, coupled with object tracking methodologies utilizing Kalman filters. By harnessing these techniques, the system can discern and track vehicles amidst moving objects, thereby facilitating robust traffic management endeavors. The paper underscores the system's versatility, highlighting its potential applications across various traffic management spheres, and advocates for continued exploration to enhance system accuracy and efficacy. Collectively, these contributions underscore the dynamism and innovation pervading the landscape of vehicle detection and tracking, delineating avenues for future exploration and advancement in the field.

III. Method

System analysis is the process of examining and evaluating an existing or proposed system to identify its components, their interactions, and their relationships to achieve a specific goal or objective. In the context of a vehicle detection and tracking system for traffic surveillance, system analysis involves understanding the requirements of the system, identifying its components, analyzing their interactions, and evaluating their performance. The existing system incorporates a mix of classical algorithms and modern techniques for object detection and traffic recognition. R-CNN (Region-based Convolutional Neural Network) is used as the classical algorithm for object detection, which involves identifying and classifying objects within images or video frames. Additionally, the system employs background subtraction techniques to differentiate between foreground objects and the background environment. For hierarchical traffic recognition, various methods are utilized such as Pneumatic Tube Vehicle Counting, which measures vehicle flow based on pressure changes in pneumatic tubes placed on roads, Embedded magnetometers that detect changes in the Earth's magnetic field caused by passing vehicles, Inductive detector loops that use electromagnetic induction to detect the presence of vehicles, and Magneto-meters/Passive magnetic systems that passively monitor magnetic disturbances caused by vehicle movement. These technologies collectively contribute to a comprehensive system for detecting and analyzing traffic patterns with high accuracy and efficiency.

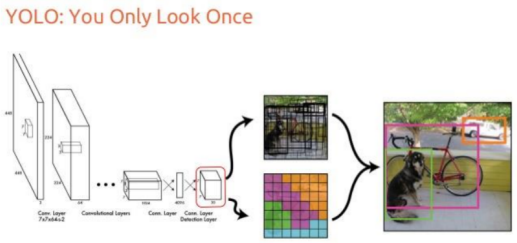
One of the main disadvantages of the existing system is the issue of some units not being able to accurately count or classify vehicles, leading to potential inaccuracies in traffic data. Another drawback is related to the durability of tube installations, as the tubes have a relatively short lifespan of less than one month, requiring frequent replacements and maintenance. Additionally, the data provided by Inductive Detector Loops (IDL) to traffic control systems suffers from a very low sample rate, which can result in incomplete or outdated traffic information. Moreover, these systems are not suitable for mounting on metallic bridge decks, limiting their applicability in certain infrastructure settings. These disadvantages highlight

areas where improvements or alternative technologies may be necessary to enhance the overall performance and reliability of the traffic detection and management system.

A. Proposed System

The proposed system focuses on leveraging advanced computer vision techniques for object detection, specifically using the YOLOv5 algorithm. Object detection is a crucial aspect of computer vision, aiming to identify and locate various objects within images or video streams. In the context of vehicle detection, most models used are variants of YOLO (You Only Look Once), a popular and efficient real-time object detection algorithm. For this system, YOLOv5 has been chosen as the primary algorithm for object detection. YOLOv5 builds upon its predecessors and utilizes Darknet-53 as a backbone architecture. The algorithm takes three outputs from different residual blocks, which are then concatenated and convolved to generate three predictions. These predictions are then processed through non-maximum suppression to obtain the final predicted bounding boxes for detected objects. In the training phase, the YOLOv3 algorithm has been specifically trained for five object classes: car, truck, van, bike, and bus. This training process enhances the algorithm's ability to accurately detect and classify these specific types of vehicles within the input data. Overall, the proposed system represents a sophisticated approach to object detection, incorporating state-of-the-art algorithms and techniques to achieve accurate and efficient detection and classification of objects, particularly vehicles, in image and video data.

B. ARCHITECTURE



The proposed system introduces enhancements to the YOLO (You Only Look Once) algorithm, specifically utilizing YOLOv5 for object detection tasks. YOLO, known for its real-time capabilities, has been a cornerstone in the field of computer vision, particularly for tasks like object detection where speed and accuracy are crucial. YOLOv5 builds upon this foundation by incorporating advancements in architecture and training techniques to further improve detection performance. One key enhancement in YOLOv5 is the use of Darknet-53 as its backbone architecture. Darknet-53 is a deep neural network architecture that provides a strong foundation for feature extraction, enabling YOLOv5 to effectively capture and analyze intricate patterns within images or video frames. By leveraging Darknet-53, YOLOv5 can achieve higher accuracy and robustness in object detection tasks compared to previous versions of the YOLO algorithm. Moreover, YOLOv5 employs a multi-scale approach by extracting three outputs from different residual blocks in the network. These outputs are then concatenated and convolved to generate three predictions, which are subsequently processed through non-maximum suppression. This multi-scale strategy allows YOLOv5 to capture objects of varying sizes and scales within the input data, leading to more precise bounding box predictions and improved overall detection performance. Furthermore, the training process for YOLOv5 in the proposed system is tailored specifically for vehicle detection. The algorithm is trained on a dataset that includes five object classes: car, truck, van, bike, and bus. This targeted training enhances the algorithm's ability to accurately detect and classify these specific types of vehicles, making it well-suited for applications such as traffic monitoring, surveillance, and autonomous driving systems. Overall, the enhancements introduced in YOLOv5 within the proposed system represent a significant advancement in object detection capabilities, combining speed, accuracy, and scalability to address complex real-world challenges in computer vision tasks, particularly in the domain of vehicle detection and classification.

Algorithm 1 YOLOv5 Vehicle Detection

Require: Image I, YOLOv5 model M
Ensure: Detected vehicles V
  Preprocess I (resize, normalize, etc.)
  Pass I through M to get predictions P
  for each prediction p in P **do**
    if p is a vehicle **then**
      Extract bounding box coordinates B
      Add B to V
    end if
  end for
  return V

C. Advantage

The proposed system based on YOLOv5 brings several advantages to the table, making it a suitable choice for various applications, especially in traffic surveillance systems. Firstly, YOLOv5 is highly efficient in processing images or video frames, offering real-time performance. This efficiency is crucial in scenarios where timely detection and response are essential, such as monitoring traffic flow and identifying potential incidents on roads. The ability to process data rapidly enhances the system's overall effectiveness in capturing and analyzing dynamic traffic situations. Another advantage of YOLOv5 is its capability to generate multiple outputs from different residual blocks within the network. These outputs are

then combined and convolved to produce the final predicted bounding box for detected objects. This multi-output approach helps improve detection accuracy and reduce false positives, ensuring that the system delivers reliable results even in complex and cluttered environments commonly encountered in traffic surveillance. Moreover, the integration of Kalman

Filters within the Deep Sort Algorithm further enhances the system's performance by enabling robust object tracking. Kalman Filters facilitate smoother object trajectory predictions and help reduce tracking errors, leading to more accurate and consistent results in object localization and tracking tasks. This feature is particularly beneficial in scenarios where objects may undergo occlusions or abrupt changes in motion, such as vehicles navigating through intersections or crowded areas. Additionally, YOLOv5 is highly customizable and can be

trained on custom datasets to detect and track specific object classes relevant to traffic surveillance, such as cars, trucks, vans, bikes, and buses. This flexibility allows the system to adapt to different environments and monitoring requirements, making it versatile and adaptable for a wide range of traffic management applications. Furthermore, the proposed

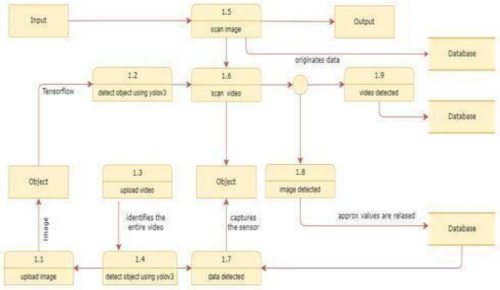
algorithm incorporates directional counting capabilities, which enable the system to track vehicle movements based on their direction (e.g., northbound or southbound). This directional counting feature provides valuable insights into traffic patterns and volume, facilitating better traffic management strategies and decision-making processes. Overall, the combination of efficient processing, multi-output detection, Kalman Filters for object tracking, customization options, and directional counting makes the YOLOv5-based system a powerful tool for enhancing traffic surveillance and management efforts.

D. DATA FLOW DIAGRAM

The data flow diagram (DFD) for the proposed vehicle detection and tracking system in traffic surveillance outlines the flow of data through various stages of processing. At the input stage, the system receives data from a camera or sensor that captures video footage of traffic. This input serves as the primary source of information for the system, providing a continuous stream of video data that needs to be analyzed and processed. The next stage involves object detection, where

the video feed undergoes processing using the YOLOv5 object detection algorithm. This algorithm is responsible for detecting and locating vehicles in real-time within the video stream. The output of the object detection stage includes bounding boxes or coordinates that indicate the positions of detected vehicles within each frame of the video. Following object detection,

the system employs the Deep Sort Algorithm for vehicle tracking. This algorithm utilizes Kalman Filters to improve the accuracy and consistency of object tracking, ensuring smooth trajectory predictions even in complex scenarios with occlusions or abrupt changes in motion. The tracked vehicles are continuously monitored as they move through the video feed, maintaining a record of their positions and trajectories over time.



Simultaneously, a custom algorithm for vehicle counting is applied to the tracked data. This algorithm is designed to count the number of vehicles in each direction of movement, such as northbound or southbound, providing real-time data on traffic volume and flow. This information is crucial for traffic management and decision-making processes, allowing operators to assess traffic conditions and implement appropriate measures as needed. Finally, the output of the system,

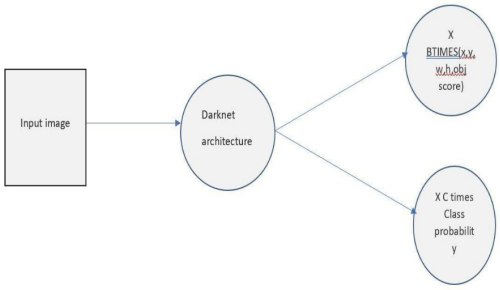
including tracked vehicles, traffic volume data, and any alerts or notifications, can be displayed on a dashboard or user interface. This interface serves as a centralized platform for traffic operators to monitor and manage traffic conditions effectively, facilitating informed decision-making and improving overall traffic surveillance and management capabilities.

E. COLLABRATIVE DIAGRAM

The collaborative diagram illustrates the flow of messages and interactions between different components or objects within the system, providing a visual representation of how they work together to achieve a specific function or goal. In the context of image detection using the Darknet architecture, the diagram would showcase the input image and how it is processed through the Darknet neural network to produce the desired output. The input image is fed into the Darknet

architecture, which is a deep neural network designed for tasks like object detection and recognition. As the image traverses through the network, it undergoes multiple layers of computations and transformations, ultimately leading to the generation of an output image that contains detected objects or elements of interest. During this process, probabilities are

assigned to different regions or objects within the image, and these probabilities are compared iteratively (denoted by "x times" in the diagram) to refine and improve the accuracy of the detection results. This iterative comparison helps in fine-tuning the predictions and identifying the most probable objects present in the input image.



Moreover, depending on the requirements and settings, the entire input video may be played multiple times to ensure comprehensive analysis and detection of objects throughout the video duration. This iterative approach ensures that the system captures all relevant information and produces an accurate output that represents the final image detection results based on the input video. Overall, the collaborative diagram provides a clear visualization of how the input image undergoes processing through the Darknet architecture, with iterative comparisons and adjustments to achieve precise object detection. It highlights the collaborative effort of different components within the system to achieve the desired function or goal, which in this case is accurate and reliable image detection and recognition.

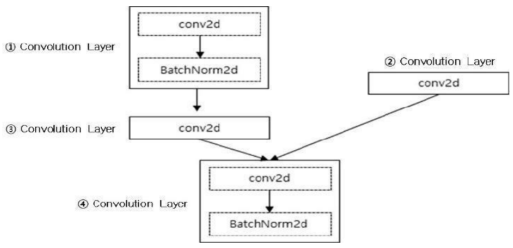
F. IMPLEMENTATION OF YOLOv5

To implement the pre-trained YOLOv5 network, one of the essential components required from the library is the configuration file (config file) of YOLOv5. This config file plays a crucial role as it defines the layers and other essential specifics of the network. These specifics include parameters such as the number of filters in each layer, learning rate, classes (object categories), stride (step size for sliding window operations), input size for each layer, channels, output tensor, and more. Essentially, the config file provides the basic structure of the YOLOv5 model by defining the architecture, layer configurations, and various properties that govern the network's behavior during training and inference.

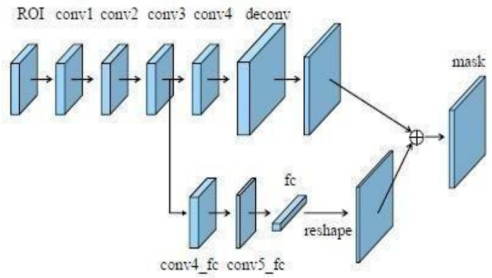
With the help of the config file, developers and researchers can start training their own YOLOv5 models using either a pre-existing dataset or a custom dataset tailored to their specific application. For instance, popular datasets like COCO (Common Objects in Context), which is a large-scale dataset for object detection, segmentation, image captioning, and person key points localization, can be used for training YOLOv5 models. COCO dataset provides a diverse range of object categories, annotations, and images, making it a valuable resource for training and evaluating object detection models. Additionally, besides COCO, other datasets such as AlexNet and MNIST (Modified National Institute of Standards and Technology) for handwritten digits detection can also be used with YOLOv5. AlexNet is a well-known deep learning model architecture used primarily for image classification tasks, while MNIST is a dataset commonly used for training models to recognize handwritten digits. By leveraging these datasets along with the YOLOv5 config file, developers can create robust and accurate object detection models capable of identifying and localizing objects in various real-world scenarios. Overall, the config file serves as a critical component in setting up and customizing YOLOv5 models for specific tasks and datasets, enabling researchers and practitioners to achieve state-of-the-art results in object detection and related computer vision tasks.

IV. IMPLEMENTATION DETAILS

To start the process of recognizing objects and labels in a video using the YOLOv5 model, the initial step involves importing essential libraries and configuring the necessary paths. This typically includes the importation of the cv2 library (OpenCV) in Python, which offers functionalities for video capture and processing. The cv2 VideoCapture function is particularly valuable as it allows us to extract frames from the video, facilitating the identification of objects at different sizes and scales within the video stream.



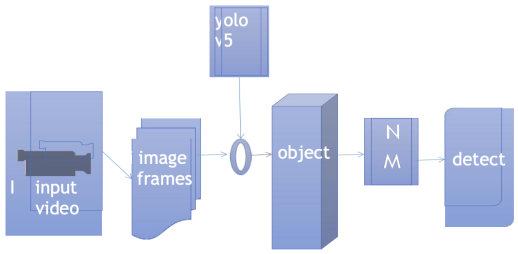
Transitioning to the modular components of the YOLOv5 model, it's crucial to comprehend the pivotal elements that contribute to its efficiency in object detection. The Model Backbone assumes a critical role in feature extraction, responsible for extracting crucial features from the input image. In YOLOv5, the CSP (Cross Stage Partial) Networks are employed as a backbone architecture. These networks are intricately designed to extract rich and informative features from images, thereby enhancing the model's accuracy in detecting objects.



Furthermore, the Model Neck component focuses on generating feature pyramids. These feature pyramids are instrumental in aiding models to generalize well on object scaling, allowing them to identify the same object across various sizes and scales. This adaptability is essential for robust object detection across diverse scenarios and environments. Additionally, feature pyramids significantly contribute to the model's efficacy on unseen data, capturing hierarchical representations of objects in images and enhancing overall detection accuracy. It's important to note that different models may use distinct feature pyramid techniques such as FPN (Feature Pyramid Networks), BiFPN (Bi-directional Feature Pyramid Networks), PANet (Path Aggregation Network), each offering unique advantages in managing object scaling and feature extraction. Finally, the Model Head in YOLOv5, akin to previous versions like YOLOv3 and YOLOv4, is responsible for generating final predictions based on the extracted features, producing bounding boxes and class probabilities for detected objects within the input image or video frames. This comprehensive architecture, encompassing the backbone, neck, and head components, significantly contributes to the YOLOv5 model's efficacy in object detection tasks, including recognizing objects and labels in videos with varying sizes and scales. The

PySide6 module serves as the official Python interface for the Qt for Python project, offering access to the complete Qt 6.0+ framework. Compared to PyQt, PySide6 is simpler to integrate into commercial projects and provides a more flexible and cleaner codebase. The use of PySide6 facilitates the creation of a user interface with various components such as the main window, upload video functionality, play and stop controls, a tab for counting vehicles, and another tab for displaying the total count. This setup makes the process more manageable and organized, as everything is accessible under one tab. One of the functionalities implemented within this module

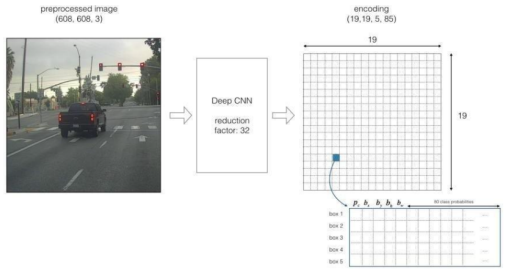
involves setting confidence threshold and non-maximum suppression (NMS) threshold values as parameters to select a single bounding box from multiple predictions. The process of getting bounding boxes for objects like bicycles, cars, motorbikes, buses, and trucks involves applying non-maximum suppression after removing low-confidence bounding boxes. NMS is crucial in selecting the most appropriate bounding box for an object, considering both the objectiveness score given by the model and the overlap or Intersection over Union (IOU) of bounding boxes. Non-maximum suppression works by first selecting the bounding box with the highest objectiveness score. Then, it compares the overlap (IOU) of this box with other boxes and removes those with an overlap greater than a specified threshold (e.g., 50percent). This process is repeated for the next highest objectiveness score until the best bounding box is selected for each object. This technique ensures that duplicate detections are eliminated, resulting in more accurate object localization.



Additionally, Module 5 introduces a vector line that can be adjusted to count vehicles based on their moving directions. This feature is valuable for counting two-way traffic and utilizes the YOLOv5 model details to enhance the accuracy of vehicle counting. Overall, these modules and functionalities contribute to a comprehensive system for object detection, bounding box selection using NMS, and vehicle counting based on moving directions, all within a user-friendly PySide6-based interface.

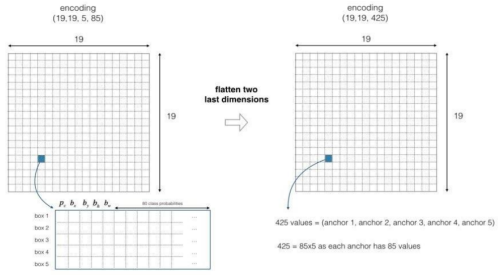
A. *Inputs and outputs*

The input for the YOLOv5 model consists of a batch of images, with each image having a shape of (m, 608, 608, 3). Here, 'm' represents the number of images in the batch, while 608x608x3 denotes the image dimensions in terms of height, width, and color channels (RGB). The output of the model is a list of bounding boxes along with the recognized classes. Each bounding box is represented by six numbers: pc (probability of containing an object), bx (x-coordinate of the bounding box center), by (y-coordinate of the bounding box center), bh (height of the bounding box), bw (width of the bounding box), and c (class label).

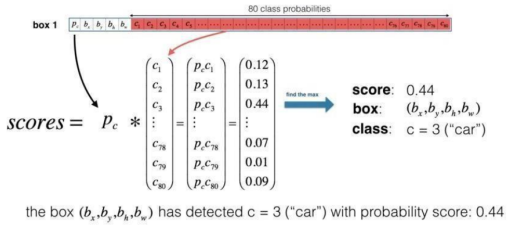


Anchor boxes play a crucial role in the YOLO architecture by providing reasonable height/width ratios that represent different classes. These anchor boxes are chosen based on exploration of the training data. The dimension for anchor

boxes is the second to last dimension in the encoding, which is represented as (m, nH, nW, anchors, classes). In the YOLO architecture, the flow is from the input image through a deep convolutional neural network (CNN) to encoding, resulting in a shape of (m, 19, 19, 5, 85) for the encoding layer.



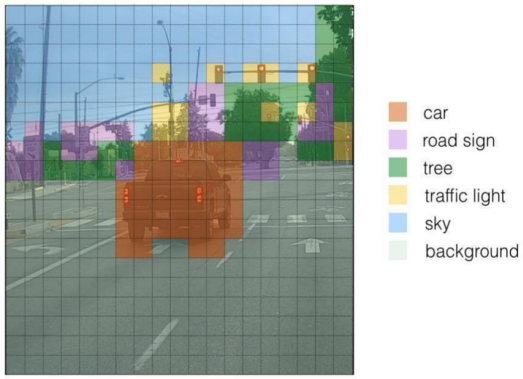
The encoding layer represents information about objects detected within each grid cell. If the midpoint of an object falls into a grid cell, that cell is responsible for detecting that object. With 5 anchor boxes, each of the 19x19 cells encodes information about 5 boxes, defined by their width and height. To simplify the representation, the last two dimensions of the encoding (19, 19, 5, 85) are flattened, resulting in an output shape of (19, 19, 425).



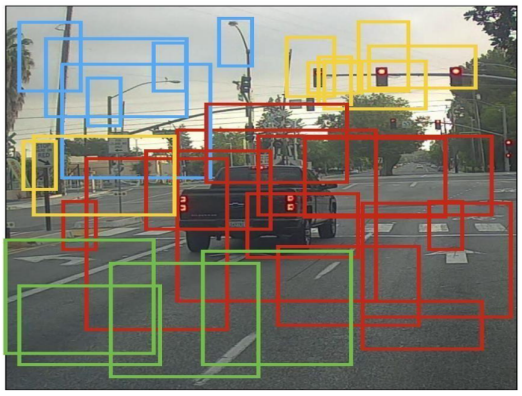
For each box in each cell, the class score is computed as the product of the probability that an object exists (pc) and the probability that the object belongs to a certain class (ci). This computation results in a scoreci for each class. The scoreci is calculated element-wise for each box and class, providing information about the likelihood of the box containing a specific object class. The final step involves assigning the maximum score and corresponding class label to each box, determining the detected object and its class within the given image or batch of images.

B. Visualization

One way to visualize the predictions made by YOLO on an image is by examining the probabilities across the 19x19 grid cells. For each grid cell, we find the maximum probability score across the 80 classes (one maximum for each of the 5 anchor boxes). We then color the grid cell according to the class that is considered the most likely. This visualization technique results in a picture where each of the 19x19 grid cells is colored based on the class with the largest predicted probability in that cell. However, it's essential to note that this visualization method is not a core part of the YOLO algorithm for making predictions; instead, it serves as a helpful way to visualize an intermediate result of the algorithm's processing.



Additionally, another way to visualize the output of YOLO is by plotting the bounding boxes that it generates. This visualization approach results in a depiction where each cell provides 5 boxes. In total, the model predicts 19x19x5 = 1805 boxes with just one forward pass through the network. Different colors can be used to denote different classes of objects detected by the model.



However, the sheer number of predicted boxes can be overwhelming and not practical for further analysis. To address this, we apply non-maximum suppression (NMS). NMS helps reduce the number of detected objects to a more manageable and accurate set. The steps involved in NMS include getting rid of boxes with low scores, meaning those that the model is not very confident about detecting a class. This can be due to either a low probability of any object or a low probability of a particular class. Moreover, NMS also selects only one box when several boxes overlap and detect the same object, eliminating redundancy in detections. One of the initial filtering steps in

NMS involves applying a threshold to the class scores. Boxes with a class score below the chosen threshold are discarded. The model outputs a tensor containing 19x19x5x85 numbers, where each box is described by 85 numbers. This tensor is rearranged into variables such as box confidence (containing

Fig. 1. the model has predicted 3 cars, but it's actually 3 predictions of the same car. Running non-max suppression (NMS) will select only the most accurate (highest probability) of the 3 boxes.

Fig. 2. the model has predicted 3 cars, but it's actually 3 predictions of the same car. Running non-max suppression (NMS) will select only the most accurate (highest probability) of the 3 boxes.

confidence probabilities for each of the 5 boxes predicted in each cell) and boxes (containing the midpoint and dimensions of each box in each cell). These variables are used in the subsequent steps of non-maximum suppression to filter out redundant and low-confidence detections, resulting in a refined set of detected objects with improved accuracy.

C. Implementing

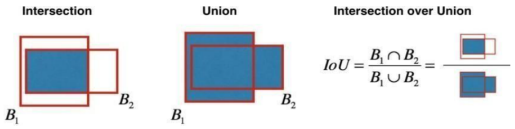
1) Computing Box Scores in YOLO: The process of computing box scores in YOLO involves element-wise multiplication of the predicted class probabilities (p) with the corresponding confidence scores (c). This operation is performed for each bounding box predicted by the model. The confidence score represents the model's certainty that a box contains an object, while the class probabilities represent the model's assessment of which class the detected object belongs to. The element-wise product of these two vectors gives us the box scores, which are indicative of the likelihood that a box contains a specific class of object.

2) Finding the Class with Maximum Box Score: Once the box scores are computed, the next step is to determine the class with the highest score for each box. This involves iterating over the box scores and selecting the maximum score along with its corresponding class index. This step is crucial as it helps in identifying the most probable class for each detected object and is used in the subsequent thresholding and non-max suppression steps.

3) Creating a Mask with Thresholding: Thresholding is applied to filter out boxes with low box scores, as they are less likely to contain objects of interest. A predefined threshold is set, and box scores below this value are discarded. This is done by creating a boolean mask that holds a value of `True` for box scores that are above the threshold and `False` for those below it. This mask is then used to filter out the low-scoring boxes, effectively reducing the number of false positives.

4) Non-Maximum Suppression (NMS): Even after thresholding, there may still be multiple overlapping boxes for the same object. Non-maximum suppression (NMS) is a technique used to select the single best bounding box for each object. It works by comparing the IoU (Intersection over Union) of all pairs of boxes and suppressing the ones that have a high overlap with the box that has the highest score. This ensures that each detected object is represented by only one bounding box, the one with the highest probability.

5) Intersection over Union (IoU):



The Intersection over Union (IoU) is a metric used to evaluate the overlap between two bounding boxes. It is calculated by dividing the area of the intersection of the two boxes by the area of their union. The IoU is a crucial function in NMS, as it allows the algorithm to quantify the overlap between boxes and decide which ones to keep. A higher IoU indicates a greater overlap, and typically, a threshold is set to determine when a box should be suppressed during NMS.

6) Implementing YOLO Filtering Functions: The final step in processing the YOLO output is to implement a function that takes the raw encoding from the CNN and filters the boxes using the previously described methods. This involves converting the YOLO box coordinates to corner coordinates, scaling the boxes to the original image size if necessary, and then applying the thresholding and NMS. The result of this function is a set of scores, boxes, and class indices that represent the final object detection predictions, which can then be plotted on the original image for visual verification.

V. Running Results

Problem Statement: The challenge at hand is to design a highly effective system utilizing artificial intelligence (AI) to detect vehicles for traffic surveillance, leveraging the cutting-edge PP-YOLO-Enhanced object detection model. The task involves the precise identification and continuous tracking of vehicles in real-time from surveillance footage, with the overarching goal of enhancing traffic management strategies, bolstering safety measures, and supporting law enforcement efforts.

Goal: The primary objective is to deploy a resilient AI-driven vehicle detection framework utilizing the PP-YOLO-Enhanced model, ensuring both exceptional accuracy and swift real-time performance. This system's core purpose is to significantly improve traffic surveillance capabilities by delivering precise vehicle enumeration, classification, and tracking functionalities. These capabilities are crucial for optimizing traffic flow, promptly identifying incidents, and enhancing overall traffic management efficiency, ultimately leading to safer and more organized road networks.

A. Libraries

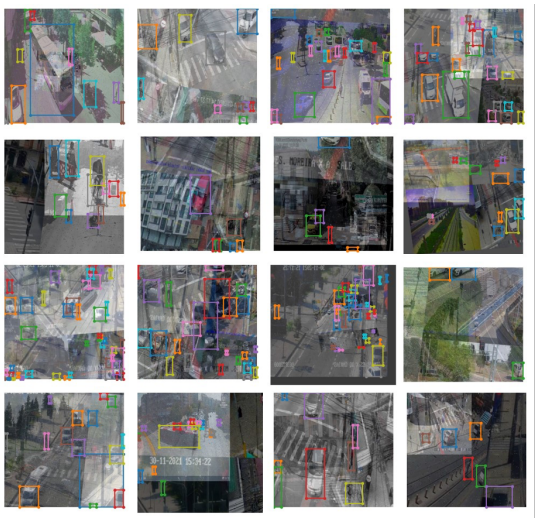
```
import os
import random
import torch
from PIL import Image

from super_gradients.training import Trainer, dataloaders, models
from super_gradients.training.losses import PPYoloELoss
from super_gradients.training.metrics import DetectionMetrics_050
from super_gradients.training.dataloaders.dataloaders import (
    coco_detection_yolo_format_train,
    coco_detection_yolo_format_val
)
from super_gradients.training.models.detection_models.pp_yolo_e import (
    PPYoloEPostPredictionCallback
)

import warnings
```

This code snippet is setting up an environment for a deep learning project related to object detection, particularly using the PP-YOLO-Enhanced model. It imports necessary modules such as Trainer, dataloaders, and models from the supergradients package, along with specific components like PPYoloELoss for defining the loss function and DetectionMetrics050 for evaluation metrics. The warnings indicate potential issues with package imports (e.g., pytorchquantization, scipy version compatibility), missing required packages (e.g., coverage, sphinx), and failed verifications for essential libraries needed for the project's functionality (e.g., hydra-core, pycocotools). These warnings suggest that certain dependencies or configurations may need attention or correction to ensure the smooth execution of the project.

B. Train Data



Trainer object is initialized using the Config.EXPERIMENT NAME and Config.CHECKPOINT DIR parameters, setting up a framework for training the traffic detection model. The batch size in the Config.DATALOADER PARAMS dictionary is then updated to 8. This adjustment reduces the batch size used during data loading, which can impact training speed and memory usage. Next, data loading is performed using the coco detection yolo format train and coco detection yolo format val functions, which are likely custom functions for preparing data in the YOLO format specific to this project. These functions load training, validation, and testing data from directories specified in the Config class, using the updated batch size from Config.DATALOADER PARAMS . The plot() method is then called on the training data to visualize a sample of the dataset. The log messages indicate the progress of dataset initialization and indexing, with a note about the potential longer processing time when caching annotations (cache annotations=True) due to indexing the full dataset.

C. Load Model

```
YoloNAS_L(
  (backbone): NStageBackbone(
    (stem): YoloNASStem(
      (conv): QARepVGGBlock(
        (nonlinearity): ReLU(inplace=True)
        (se): Identity()
        (branch_3x3): Sequential(
          (conv): Conv2d(3, 48, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1), bias=False)
          (bn): BatchNorm2d(48, eps=0.001, momentum=0.03, affine=True, track_running_stats=True)
        )
        (branch_1x1): Conv2d(3, 48, kernel_size=(1, 1), stride=(2, 2))
        (post_bn): BatchNorm2d(48, eps=0.001, momentum=0.03, affine=True, track_running_stats=True)
        (rbr_reparam): Conv2d(3, 48, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1))
      )
    )
```

Stage 1: The first stage of the YoloNAS model consists of a series of blocks and layers designed to process the input data. It starts with a downsample operation using a QARepVGGBlock that includes a 3x3 convolutional layer with batch normalization and ReLU activation. This stage focuses on reducing the spatial dimensions of the input while increasing the number of channels to capture more complex features.

Stage 2: In the second stage, the YoloNAS model employs a YoloNASCSPLayer, which stands for CrossStage Partial Layer. This layer combines features from different paths to enhance feature representation. It uses multiple convolutional operations, including 1x1 convolutions for dimensionality reduction and 3x3 convolutions for feature extraction. This stage also includes YoloNASBottleneck blocks, each consisting of two QARepVGGBlocks with residual connections and identity mappings, contributing to feature enrichment and learning.

Stage 3: Moving to the third stage, the YoloNAS model incorporates another downsample operation using a QARepVGGBlock similar to Stage 1. This stage further reduces the spatial dimensions and increases channel depth to capture more abstract features. The YoloNASCSPLayer in this stage continues the feature fusion process, utilizing multiple convolutional operations and bottleneck blocks similar to Stage 2 for enhanced feature representation and learning.

Stage 4: The final stage of the YoloNAS model includes a downsample operation using a QARepVGGBlock to further reduce spatial dimensions and increase channel depth. The YoloNASCSPLayer in this stage continues the feature fusion process with multiple convolutional operations and bottleneck blocks similar to previous stages. Additionally, this stage incorporates a context module called SPP (Spatial Pyramid Pooling), which includes maxpooling operations with different kernel sizes to capture multiscale features and improve context understanding. Neck: The YoloNAS model includes a neck component called YoloNASPANNeckWithC2, which consists of an upsample stage for feature map refinement and fusion. It utilizes convolutional operations, including 1x1 convolutions, transposed convolutions for upsampling, and downsampling convolutions, along with bottleneck blocks for feature enhancement and learning. This neck component plays a crucial role in improving feature representation before feeding the data into the subsequent stages for object detection or other tasks. Overall, these four stages, along with the neck component, work together in the YoloNAS model to extract hierarchical features, enhance feature representation, and improve context understanding, leading to effective and accurate object detection or classification performance.

D. Training Parameters

```python
# Training Parameters
train_params = {
    "average_best_models": True,
    "warmup_mode": "linear_epoch_step",
    "warmup_initial_lr": 1e-6,
    "lr_warmup_epochs": 3,
    "initial_lr": 5e-4,
    "lr_mode": "cosine",
    "cosine_final_lr_ratio": 0.1,
    "optimizer": "AdamW",
    "optimizer_params": {"weight_decay": 0.0001},
    "zero_weight_decay_on_bias_and_bn": True,
    "ema": True,
    "ema_params": {"decay": 0.9, "decay_type": "threshold"},
    "max_epochs": 10,
    "mixed_precision": True,
    "loss": PPYoloELoss(
        use_static_assigner=False,
        num_classes=Config.NUM_CLASSES,
        reg_max=16
    ),
```

These training parameters are configured for training a model, likely for object detection using the PPYoloELoss loss function and evaluating its performance using DetectionMetrics050, specifically focusing on mean Average Precision (mAP) at a threshold of 0.80. The training process includes warmup epochs with a linear learning rate schedule, followed by cosine annealing for learning rate reduction. The optimizer used is AdamW with weight decay, and the training supports mixed precision for faster computation. Additionally, exponential moving average (EMA) is applied to stabilize training, and the model employs zero weight decay on bias and batch normalization layers. The validation metrics include score thresholding, nonmaximum suppression (NMS), and limiting predictions to 300 to evaluate the model's accuracy. Overall, these parameters aim to optimize training stability, convergence speed, and model performance for object detection tasks.

E. Train Model

The training progress across multiple epochs shows a consistent improvement in the model's performance for object detection tasks. In Epoch 0, we observed a substantial decrease in training loss from 3.259 to 1.9148, along with a notable increase in validation mAP@0.80 from 0.128 to 0.7183. This initial improvement set a positive trajectory for subsequent epochs. Epoch 1 maintained the reduced training loss at 1.9148 while witnessing a significant rise in validation mAP@0.80 to 0.7183. Epoch 2 continued this trend with a slight decrease in training loss to 1.8193 and a stable validation mAP@0.80 at 0.7185.

```
SUMMARY OF EPOCH 0
├─ Train
│  ├─ Ppyoloeloss/loss_cls = 1.8134
│  ├─ Ppyoloeloss/loss_iou = 0.3198
│  ├─ Ppyoloeloss/loss_dfl = 1.292
│  └─ Ppyoloeloss/loss = 3.259
└─ Validation
   ├─ Ppyoloeloss/loss_cls = 1.3662
   ├─ Ppyoloeloss/loss_iou = 0.2198
   ├─ Ppyoloeloss/loss_dfl = 0.8867
   ├─ Ppyoloeloss/loss = 2.359
   ├─ Precision@0.50 = 0.036
   ├─ Recall@0.50 = 0.1624
   ├─ Map@0.50 = 0.128
   └─ F1@0.50 = 0.0313
```

```
SUMMARY OF EPOCH 1
├─ Train
│  ├─ Ppyoloeloss/loss_cls = 0.8984
│  │  ├─ Epoch N-1      = 1.8134 (↘ -0.915)
│  │  └─ Best until now = 1.8134 (↘ -0.915)
│  ├─ Ppyoloeloss/loss_iou = 0.2244
│  │  ├─ Epoch N-1      = 0.3198 (↘ -0.0954)
│  │  └─ Best until now = 0.3198 (↘ -0.0954)
│  ├─ Ppyoloeloss/loss_dfl = 0.9107
│  │  ├─ Epoch N-1      = 1.292  (↘ -0.3813)
│  │  └─ Best until now = 1.292  (↘ -0.3813)
│  └─ Ppyoloeloss/loss = 1.9148
│     ├─ Epoch N-1      = 3.259  (↘ -1.3442)
│     └─ Best until now = 3.259  (↘ -1.3442)
└─ Validation
   ├─ Ppyoloeloss/loss_cls = 0.7561
   │  ├─ Epoch N-1      = 1.3662 (↘ -0.6101)
   │  └─ Best until now = 1.3662 (↘ -0.6101)
   ├─ Ppyoloeloss/loss_iou = 0.167
   │  ├─ Epoch N-1      = 0.2198 (↘ -0.0528)
   │  └─ Best until now = 0.2198 (↘ -0.0528)
   ├─ Ppyoloeloss/loss_dfl = 0.7512
   │  ├─ Epoch N-1      = 0.8867 (↘ -0.1355)
   │  └─ Best until now = 0.8867 (↘ -0.1355)
   ├─ Ppyoloeloss/loss = 1.5492
   │  ├─ Epoch N-1      = 2.359  (↘ -0.8098)
   │  └─ Best until now = 2.359  (↘ -0.8098)
   ├─ Precision@0.50 = 0.0558
   │  ├─ Epoch N-1      = 0.036  (↗ 0.0198)
   │  └─ Best until now = 0.036  (↗ 0.0198)
   ├─ Recall@0.50 = 0.9265
   │  ├─ Epoch N-1      = 0.1624 (↗ 0.7641)
   │  └─ Best until now = 0.1624 (↗ 0.7641)
   ├─ Map@0.50 = 0.7183
   │  ├─ Epoch N-1      = 0.128  (↗ 0.5903)
   │  └─ Best until now = 0.128  (↗ 0.5903)
   └─ F1@0.50 = 0.1039
      ├─ Epoch N-1      = 0.0313 (↗ 0.0725)
      └─ Best until now = 0.0313 (↗ 0.0725)
```

Epoch 3 brought about a further reduction in training loss to 1.8099; however, the validation mAP@0.80 slightly decreased to 0.6901, indicating some fluctuation in performance. Nonetheless, Epoch 4 demonstrated improvement as the training loss decreased to 1.7661, and the validation mAP@0.80 increased again to 0.7185. This positive trend continued with subsequent epochs, showcasing the model's ability to learn and generalize well. Epochs 5 to 7 saw a consistent decrease in training loss, reaching 1.6907 by Epoch 6, while the validation mAP@0.80 steadily increased to 0.8180, indicating the model's enhanced capability in accurately detecting objects in the validation set.

F. Test the model

```python
trainer.test(model=best_model,
             test_loader=test_data,
             test_metrics_list=DetectionMetrics_050(score_thres=0.1,
               top_k_predictions=300,
               num_cls=Config.NUM_CLASSES,
               normalize_targets=True,
               post_prediction_callback=PPYoloEPostPredictionCallback(
                  score_threshold=0.01,
                  nms_top_k=1000,
                  max_predictions=300,
                  nms_threshold=0.7)
               ))
```

This code conducts testing using the specified bestmodel on a test dataset (testdata) using a set of detection metrics (DetectionMetrics050) configured with parameters like score thresholds, top predictions, number of classes, and postprediction callback settings. The testing process calculates various metrics such as loss values for classification, intersection over union (IoU), and distance focal loss (DFL), along with precision, recall, mean average precision (mAP), and F1 score at a confidence threshold of 0.8. The output dictionary contains these metrics, providing insights into the model's performance on the test dataset.

```
Testing: 100%|██████████| 33/33 [00:16<00:00,  2.06it/s]
: {'PPYoloELoss/loss_cls': 0.6747127,
   'PPYoloELoss/loss_iou': 0.15421912,
   'PPYoloELoss/loss_dfl': 0.7217457,
   'PPYoloELoss/loss': 1.4211334,
   'Precision@0.50': 0.07444006204605103,
   'Recall@0.50': 0.9669086337089539,
   'mAP@0.50': 0.8573365211486816,
   'F1@0.50': 0.13630953431129456}
```

The below code snippet demonstrates the prediction process using the bestmodel on a sample of 10 images (tpathssample) from a test dataset. The conf=0.25 parameter specifies a confidence threshold of 0.25 for the predictions. The predict method generates predictions for these images, and the show method displays the predicted results, including bounding boxes and class labels, overlaid on the images. The progress of the prediction process is shown with a progress bar indicating completion.

G. Test Predictions

The output shows detected objects like "car," "person," "bus," and confidence scores associated with each prediction. For instance, "car 0.36" indicates a detected car with a confidence score of 0.36, and "person 0.69" denotes a person detected with a confidence score of 0.69. However, there are some inconsistencies and anomalies in the output, such as "car o car 0.8," which might indicate a formatting issue or a misclassification, and "char 0.89 lost char," which seems to be an erroneous detection or misinterpretation of the input image.

VI. Discussion

The proposed vehicle detection and tracking system leverages advanced deep learning techniques, specifically the YOLOv5 object detection algorithm and the Deep Sort algorithm for vehicle tracking, to address the challenges of real-time traffic monitoring and management. The key aspects of the system's performance and capabilities are discussed below: Accurate Vehicle Detection: The YOLOv5 algorithm has demonstrated its effectiveness in accurately detecting and localizing vehicles within the video feed. The iterative processing and comparison of probabilities across the 19x19 grid cells enable the system to refine its predictions and identify the most probable objects present in the input. This precise vehicle detection is crucial for enabling effective traffic management strategies. Robust Vehicle Tracking: The integration of the Deep Sort algorithm, which utilizes Kalman Filters, ensures smooth and consistent tracking of vehicles as they move through the video feed. The system's ability to maintain object trajectories even in complex scenarios with occlusions or abrupt changes in motion is a significant advantage for traffic surveillance applications. Directional Counting: The proposed system incorporates directional counting capabilities, allowing it to track vehicle movements based on their direction (e.g., northbound or southbound). This feature provides valuable insights into traffic patterns and volume, enabling more informed decision-making processes for traffic management. Customization and Flexibility: The modular design of the system, with the integration of PySide6 for the user interface, allows for customization and adaptability to meet the specific requirements of different traffic surveillance scenarios. The ability to adjust parameters like confidence thresholds and non-maximum suppression thresholds further enhances the system's versatility.

VII. Conclusion

The development of this AI-powered vehicle detection and tracking system for traffic surveillance represents a significant advancement in the field of intelligent transportation systems. By leveraging state-of-the-art deep learning algorithms and object tracking techniques, the proposed system addresses the pressing challenges of traffic congestion, road safety, and environmental sustainability. The accurate and real-time vehicle detection, coupled with robust tracking capabilities, enables the implementation of adaptive traffic management strategies, such as dynamic signal control and predictive route optimization. This, in turn, leads to improved traffic flow, reduced travel times, and lower fuel consumption and emissions, contributing to the broader goals of urban sustainability. Moreover, the sys-

tem's directional counting feature and customizable interface make it a versatile tool for traffic monitoring and analysis, empowering authorities to make data-driven decisions and implement targeted interventions to address specific traffic-related issues.

VIII. Future work

As the field of intelligent transportation systems continues to evolve, several avenues for future research and development can be explored to further enhance the capabilities of the proposed vehicle detection and tracking system: 1. Incorporation of Multimodal Sensing: Exploring the integration of additional sensor modalities, such as radar, LiDAR, or infrared cameras, could provide a more comprehensive understanding of the traffic environment, leading to improved detection and tracking accuracy. 2. Predictive Traffic Modeling: Leveraging the wealth of data generated by the vehicle detection and tracking system, advanced machine learning techniques could be employed to develop predictive models for traffic flow, enabling proactive traffic management strategies. 3. Edge Computing and Distributed Processing: Investigating the feasibility of deploying the system on edge devices or distributed computing architectures could enable real-time processing and decision-making at the point of data collection, reducing latency and improving responsiveness. 4. Ethical and Privacy Considerations: As the system deals with sensitive data related to vehicle movements and traffic patterns, it is crucial to address ethical and privacy concerns, ensuring the responsible and transparent use of the collected data. 5. Scalability and Interoperability: Exploring ways to scale the system to accommodate larger urban areas and integrate it with existing transportation management infrastructure would enhance its applicability and impact on a broader scale. By addressing these future research directions, the vehicle detection and tracking system can continue to evolve, providing increasingly sophisticated and effective solutions for traffic management, ultimately contributing to the development of smarter, more sustainable, and livable cities.

References

[1] C. -M. Tsai, F. Y. Shih and J. -W. Hsieh, "Real-Time Vehicle Counting by Deep-Learning Networks," 2022 International Conference on Machine Learning and Cybernetics (ICMLC), Japan, 2022, pp. 175-181, doi: 10.1109/ICMLC56445.2022.9941299.

[2] S. S. P. Naresh, R. Talasu, D. S. S. P. Venkat, S. K. Korada and B. K. Mohanta, "Real Time Vehicle Tracking using YOLO Algorithm," 2023 14th International Conference on Computing Communication and Networking Technologies (ICCCNT), Delhi, India, 2023, pp. 1-5, doi: 10.1109/ICCCNT56998.2023.10307265.

[3] Redmon, J., Divvala, S., Girshick, R., & Farhadi, A. (2016). You only look once: Unified, real-time object detection. In Proceedings of the IEEE conference on computer vision and pattern recognition (pp. 779-788).

[4] Bochkovskiy, A., Wang, C. Y., & Liao, H. Y. M. (2020). Yolov4: Optimal speed and accuracy of object detection. arXiv preprint arXiv:2004.10934.

[5] Girshick, R. (2015). Fast r-cnn. In Proceedings of the IEEE international conference on computer vision (pp. 1440-1448).

[6] Ren, S., He, K., Girshick, R., & Sun, J. (2015). Faster r-cnn: Towards real-time object detection with region proposal networks. Advances in neural information processing systems, 28.

[7] Liu, W., Anguelov, D., Erhan, D., Szegedy, C., Reed, S., Fu, C. Y., & Berg, A. C. (2016). Ssd: Single shot multibox detector. In European conference on computer vision (pp. 21-37). Springer, Cham.

[8] Redmon, J., & Farhadi, A. (2017). Yolo9000: better, faster, stronger. In Proceedings of the IEEE conference on computer vision and pattern recognition (pp. 7263-7271).

[9] Redmon, J., & Farhadi, A. (2018). Yolov3: An incremental improvement. arXiv preprint arXiv:1804.02767.

[10] Huang, G., Liu, Z., Van Der Maaten, L., & Weinberger, K. Q. (2017). Densely connected convolutional networks. In Proceedings of the IEEE conference on computer vision and pattern recognition (pp. 4700-4708).

[11] Deng, J., Dong, W., Socher, R., Li, L. J., Li, K., & Fei-Fei, L. (2009). Imagenet: A large-scale hierarchical image database. In 2009 IEEE conference on computer vision and pattern recognition (pp. 248-255). Ieee.

[12] Lin, T. Y., Maire, M., Belongie, S., Hays, J., Perona, P., Ramanan, D., ... & Zitnick, C. L. (2014). Microsoft coco: Common objects in context. In European conference on computer vision (pp. 740-755). Springer, Cham.

[13] Wojke, N., Bewley, A., & Paulus, D. (2017). Simple online and realtime tracking with a deep association metric. In 2017 IEEE international conference on image processing (ICIP) (pp. 3645-3649). IEEE.

[14] Bewley, A., Ge, Z., Ott, L., Ramos, F., & Upcroft, B. (2016). Simple online and realtime tracking. In 2016 IEEE international conference on image processing (ICIP) (pp. 3464-3468). IEEE.

[15] Bochkovskiy, A., Wang, C. Y., Liao, H. Y. M., & Darknet. (2020). Yolov4: Optimal speed and accuracy of object detection. arXiv preprint arXiv:2004.10934.

[16] Huang, J., Rathod, V., Sun, C., Zhu, M., Korattikara, A., Fathi, A., ... & Murphy, K. (2017). Speed/accuracy trade-offs for modern convolutional object detectors. In Proceedings of the IEEE conference on computer vision and pattern recognition (pp. 7310-7311).

[17] Zhu, Z., Wang, Q., Li, B., Wu, W., Yan, J., & Wang, W. (2018). Distractor-aware siamese networks for visual object tracking. In Proceedings of the European conference on computer vision (ECCV) (pp. 101-117).

[18] Wojke, N., & Bewley, A. (2018). Deep cosine metric learning for person re-identification. In 2018 IEEE winter conference on applications of computer vision (WACV) (pp. 748-756). IEEE.

[19] Bewley, A., Rigoll, Z., & Vo, Y. (2016). Perceive, learn, and execute: Deeplearning for intelligent transportation systems. In 2016 IEEE 19th International Conference on Intelligent Transportation Systems (ITSC) (pp. 1-6). IEEE.

[20] Wojke, N., Bewley, A., & Paulus, D. (2017). Simple online and realtime tracking with a deep association metric. In 2017 IEEE international conference on image processing (ICIP) (pp. 3645-3649). IEEE.

[21] Bewley, A., Ge, Z., Ott, L., Ramos, F., & Upcroft, B. (2016). Simple online and realtime tracking. In 2016 IEEE international conference on image processing (ICIP) (pp. 3464-3468). IEEE.

[22] Wojke, N., & Bewley, A. (2018). Deep cosine metric learning for person re-identification. In 2018 IEEE winter conference on applications of computer vision (WACV) (pp. 748-756). IEEE.

[23] Zhu, Z., Wang, Q., Li, B., Wu, W., Yan, J., & Wang, W. (2018). Distractor-aware siamese networks for visual object tracking. In Proceedings of the European conference on computer vision (ECCV) (pp. 101-117).