Politechnika Częstochowska

Wydział Inżynierii Mechanicznej i Informatyki

PRACA DYPLOMOWA MAGISTERSKA

An Intelligent Motion Tracking System Using Raspberry PI

Inteligentny system śledzenia ruchu wykorzystujący Raspberry PI

Zinelabidine Leghelimi

Nr albumu: 135862

Kierunek: Informatyka

Forma studiów: stacjonarne

Poziom studiów : II

Promotor pracy: Prof. dr hab. inż. R. Cierniak

Praca przyjęta dnia:

Podpis promotora:

Częstochowa, April 2023

Copyright Page

Dedication

This dissertation is dedicated to my beloved family…

Acknowledgments

PERSONAL

First, I am grateful for the assistance, availability, and understanding of my supervisor, Prof. Eng R. Cierniek.
I'm also grateful to my parents for their unwavering support, guidance, and encouragement throughout my academic career. And to my sisters for their love and support.
Next, my sincere thanks go also to my friends for their constant encouragement and motivation, especially Issam Eddine, Aymen, and Oussama, as well as to my classmates for their camaraderie and moral support.

INSTITUTIONAL

I appreciate the help and guidance of my professors.
Additionally, I would like to express my deepest gratitude to the international office employees of the Czestochowa University of Technology for their help and assistance throughout my studies. Their support and dedication have been instrumental in my success.

Abstract of

A Motion Tracking System using Raspberry Pi

Computer Vision (CV) applications are undergoing a significant revolution, as they are being applied in previously difficult or impossible areas. This includes object detection and tracking within different environments. The development of CV is closely tied to the advancements in Artificial Intelligence (AI) and Deep Learning (DL), which have enabled the creation of powerful algorithms for object tracking and detection. However, these algorithms require significant computational resources and are typically deployed on powerful servers or the cloud. This can be cost-prohibitive and may not be feasible for certain applications or users.

The proliferation of edge devices, such as Internet of Things (IoT) devices, presents a significant opportunity for the deployment of CV applications. However, deploying these algorithms on edge devices is challenging due to computational power, memory, and storage limitations. To tackle these limitations, recent developments in hardware such as Neural Compute Stick (NCS) accelerator make it possible to increase the inference speed and perform offline computations. Nevertheless, the challenge of deploying deep learning algorithms on edge devices still exists. This thesis proposed a DL-based model that follows the tracking-by-detection approach to track multiple objects. The model is deployable in edge computing thanks to an optimization phase that enables deploying the YOLOv8 and DeepSORT on devices with limited resources, namely Raspberry Pi. The experiments that were made on a custom dataset showed that the model on each both version normal and tiny obtained good results in comparison with other state-of-art traditional algorithms.

Keywords: Computer Vision, Deep Learning, Internet of Things, Neural Compute Stick, Object Tracking, Object Detection, YOLOv8, DeepSORT, Raspberry Pi.

Abstrakt pracy pt.

System śledzenia ruchu przy użyciu Raspberry Pi

Aplikacje Computer Vision (CV) przechodzą znaczącą rewolucję, ponieważ są stosowane w obszarach, które wcześniej były trudne lub niemożliwe. Obejmuje to wykrywanie i śledzenie obiektów w różnych środowiskach. Rozwój CV jest ściśle powiązany z postępami w dziedzinie sztucznej inteligencji (AI) i głębokiego uczenia się (DL), które umożliwiły stworzenie potężnych algorytmów do śledzenia i wykrywania obiektów. Algorytmy te wymagają jednak znacznych zasobów obliczeniowych i są zwykle wdrażane na wydajnych serwerach lub w chmurze. Może to być zbyt kosztowne i może nie być wykonalne dla niektórych aplikacji lub użytkowników. Rozprzestrzenianie się urządzeń brzegowych, takich jak urządzenia Internetu Rzeczy (IoT), stwarza znaczącą szansę na wdrożenie aplikacji CV. Jednak wdrożenie tych algorytmów na urządzeniach brzegowych jest trudne ze względu na ograniczenia mocy obliczeniowej, pamięci i pamięci masowej. Aby poradzić sobie z tymi ograniczeniami, najnowsze rozwiązania sprzętowe, takie jak akcelerator Neural Compute Stick (NCS), umożliwiają zwiększenie szybkości wnioskowania i wykonywanie obliczeń w trybie offline. Niemniej jednak wyzwanie związane z wdrażaniem algorytmów głębokiego uczenia się na urządzeniach brzegowych nadal istnieje.
W tej pracy zaproponowano model oparty na DL, który jest zgodny z podejściem śledzenia przez wykrywanie w celu śledzenia wielu obiektów. Model można wdrożyć w edge computing dzięki fazie optymalizacji, która umożliwia wdrożenie YOLOv8 i DeepSORT na urządzeniach o ograniczonych zasobach, a mianowicie na Raspberry Pi. Eksperymenty przeprowadzone na niestandardowym zbiorze danych wykazały, że model na każdej wersji normalnej i małej uzyskał dobre wyniki w porównaniu z innymi najnowocześniejszymi algorytmami tradycyjnymi.

Słowa kluczowe: Wizja komputerowa, głębokie uczenie się, Internet rzeczy, Neural Compute Stick, śledzenie obiektów, wykrywanie obiektów, YOLOv8, DeepSORT, Raspberry Pi.

Table of Contents

List of Figures

List of Tables

List of Acronyms

AI	Artificial Intelligence
ANN	Artificial Neural Network
CNN	Convolutional Neural Network
CNS	Compute Neural Stick
CPU	Central Processing Unit
CV	Computer Vision
DL	Deep Learning
FC	Fully Connected
GPU	Graphical Processing Unit
IoU	Intersection Over Union
LKOP	Lukas Kanade Optical Flow
ML	Machine Learning
MOT	Multiple Object Tracking
MS	Mean Shift
RCNN	Regions with CNN features
VPU	Vision Processing Unit
YOLO	You Only Look Once
SSD	Single Shot Detector

1

Introduction

Context And Motivation

The rapid growth in the number of Internet of Things (IoT) devices in use globally, which is projected to reach 75 billion by 2025, highlights the challenges that arise from the widespread usage of computer vision tasks in these devices, especially since these devices are mostly trying to simulate the human vision. Many IoT applications, such as autonomous cars, drones, and smart cities, rely heavily on object detection and tracking systems to function properly. However, the widespread usage of these systems in many IoT devices presents significant challenges for the industry, as it requires continuous development and improvement of computer vision and deep learning techniques as well as advancements in hardware capabilities to meet the demands of these systems.

In terms of software solutions, many traditional algorithms have been developed that are easy to use and perform efficiently. However, these algorithms require significant computational power and a large amount of time. Moreover, their performance may be limited in certain scenarios due to their dependence on geometric shapes and comparisons. With the advancements in artificial intelligence and deep learning, modern solutions that rely on AI have gained attention. These solutions are more adaptable and efficient, as they are not limited by geometric shapes, and they can continue to improve with the development of more data. In this regard, we aim to make an analytical comparison between the results of selected traditional algorithms and deep learning algorithms, on the one hand, and to compare the performance of deep learning algorithms in various devices, on the other hand.

In terms of hardware solutions, it is known that the usual local computer vision systems rely on local clouds or servers to train and infer deep learning systems. But with the low power and computation of IoT devices, this lack has merged issues regarding the real-time computer vision algorithms' performance that obliged us to move from cloud to edge to achieve better results and solve these issues. In this regard, there are many hardware solutions available at the edge level, but we chose the Neural Computer Stick by Intel to use in developing the system because it is

identical to the framework that we will use, and also because its size is very suitable for edge applications, and certainly due to the results it showed compared to other devices.

Whereas considering the traditional solutions and comparing them with modern solutions to reach the best accuracy, and then comparing modern solutions and finding the best one to develop our model and use it at the edge level, this is what I saw lacking in the existing solutions, where Most of the proposed solutions are chosen directly without studying and comparing it in various circles, and this is what excludes many solutions that can give better results.

Aims And Objectives

Computer vision systems that need to be real-time (DL) mostly use a traditional cloud-based setup with servers backed by capable parallel computing hardware to process clients' requests from applications. This forces the data collection to be centralized in data centers that are under the control of private service providers. But keeping the data that far has raised serious issues in many quality metrics including latency, privacy, security, and intensive processing power. With the remarkable progress in recent years, many solutions are available in terms of both hardware and software for tuning DL-based applications on devices with low computing power and improving privacy and security issues.

As we mentioned before, the issue that can appear in an edge environment is the low computational available which affects the accuracy and time of real-time motion tracking, so the need for a tiny and simplified model to cooperate with hardware accelerators to achieve better results.

In the following list we are going to mention the aim and objectives of our work:

- Design and implementation of 3 traditional motion tracking algorithms (Mean shift, Lukas Kanade Optical, and Kalman Filter).
- Comparative analysis of these 3 algorithms based on selected metrics and human experience as well.
- Select the best traditional algorithm to be used in modern tracking.
- Implementation of the latest version of YOLO (v8) for object detection with the two variants large and nano.
- Implementation of DeepSort Algorithm as an object tracker and configuring it to work with YOLO.
- Optimization and lightweight of our model using OpenVino and NCS.
- Comparative analysis of our two implemented versions of YOLO on different platforms including CPU, GPU, and VPU.
- Explore the possible cooperation between the traditional methods and the modern ones.

- Based on the comparative analysis we will create and deploy a motion-tracking algorithm (detection based) on the edge and configure it with the NCS to speed up the inference of the model.

Dissertation Outline

The rest of this dissertation is presented in the following manner:

- Chapter 2: **"Background**" provides definitions of key terms related to our area of interest, including computer vision, AI, and IoT.
- Chapter 3: "**Literature Review**", evaluates related work in motion tracking and discusses their benefits and limitations. This evaluation includes traditional algorithms and modern ones.
- Chapter 4: "**Proposed Solution**", in which we present our solution from the start, passing by presenting the final model in detail, the used dataset, and the training phase.
- Chapter 5: "**Obtained results**", presents the work environment including hardware specifications and tools, and programming languages used to implement the models, and also presents the results obtained from the experiments of both traditional motion tracking algorithms and modern ones.
- Chapter 6: "**Conclusions and Perspectives**", Here we will conclude and gives our final inference regarding the model and show the deficiencies that our system may face, and we will also present the perspectives, and we will also give recommendations to those interested in working on something related to our topic.

Part 1. State of the Art

Due to the ongoing evolution of computer vision, in which several approaches and techniques have emerged, some of which are even contradictory, a '**State of the art**' part is essential to provide a comprehensive overview of the theoretical foundation of the field. It clarifies key terms and concepts used in the field, reviews existing applications, and discusses recent findings. This part is organized as follows:

In the "**Background**" chapter, we presented a concise yet comprehensive overview of computer vision terms related to object detection and tracking, followed by a detailed discussion of AI algorithms including the supervised and unsupervised methods. We focused on CNN and its variants such as RCNN, which are commonly used for object tracking. We concluded the chapter by introducing a set of IoT devices that can contribute to solving hardware-related problems, namely Raspberry PI (RPi) and Neural Compute Stick (NCS).

Furthermore, to offer a clear understanding of motion tracking, we present in the second chapter a review of the literature, including traditional and deep learning-based algorithms. We also highlighted their advantages and limitations, which helped us identify gaps that exist in the current state of the art and allowed us to express the problem more accurately.

"Computers are able to see, hear and learn. Welcome to the future."

—Dave Waters

2

Background

INTRODUCTION

The importance of motion-tracking systems lies in their ability to simulate three of the most fundamental human capabilities: vision, inference, and recognition. To achieve this, such systems should be equipped with a camera to capture, intelligence to recognize and track objects, enough energy to process data quickly, and flexibility to adapt to diverse situations.

This chapter aims to provide a thorough understanding of motion-tracking systems. It explores four related domains: *computer vision* and *deep learning*, which enable systems to simulate the three aforementioned capabilities; the *internet of things* and *edge computing*, which provide systems with the required energy and computation speed.

COMPUTER VISION

1.1. Object Detection

Object detection is a computer vision technique that answers the question: "What objects are where?". In other words, it seeks to identify and detect all instances of semantic objects in a picture or video, such as persons, cats, or cars [1, 2]. An example of the output of this technique is shown in the figure below, where rectangular boxes bound the detected objects.

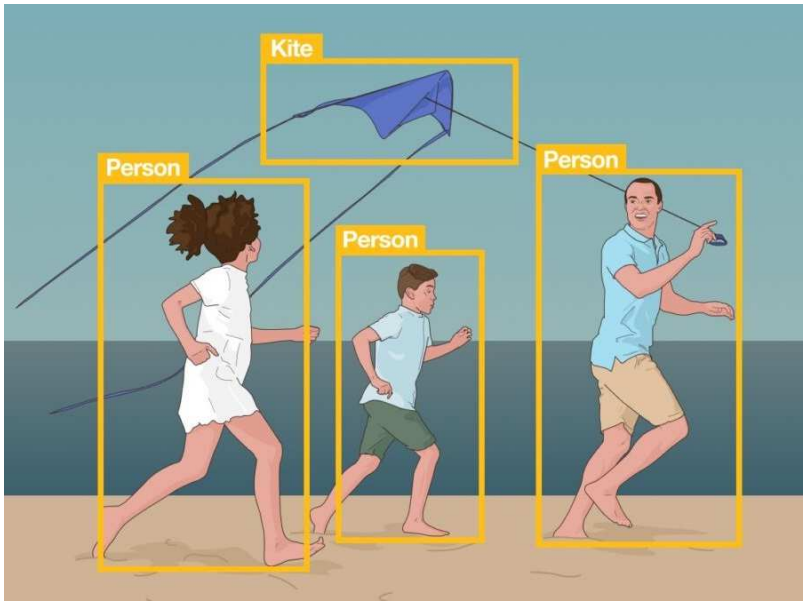


Fig. 2.1 Illustration of an object detection technique [3].

There are several approaches to object detection, and they can be broadly classified into two categories: traditional and deep learning-based methods [4], as schematized in Fig. 2.2. On the one hand, traditional methods typically involve feature extraction followed by a recognition step. Among these methods is the *Viola-Jones* method, which is based on the observation that facial features have a relative relationship to each other. In addition, there are other feature-based descriptors, such as SIFT (*Scale-Invariant Feature Transforms*) and HOG (*Histogram of Oriented Gradients*), which extract features using various image computations and transformations. On the other hand, modern DL-based methods have taken advantage of the strong feature learning and representation capabilities of neural networks. The R-CNN family of algorithms, which are designed to perform well in object localization and recognition tasks (see section 2.2), as well as YOLO (*You Only Look Once*), which is designed to work in real-time, are two examples of such DL-based methods.

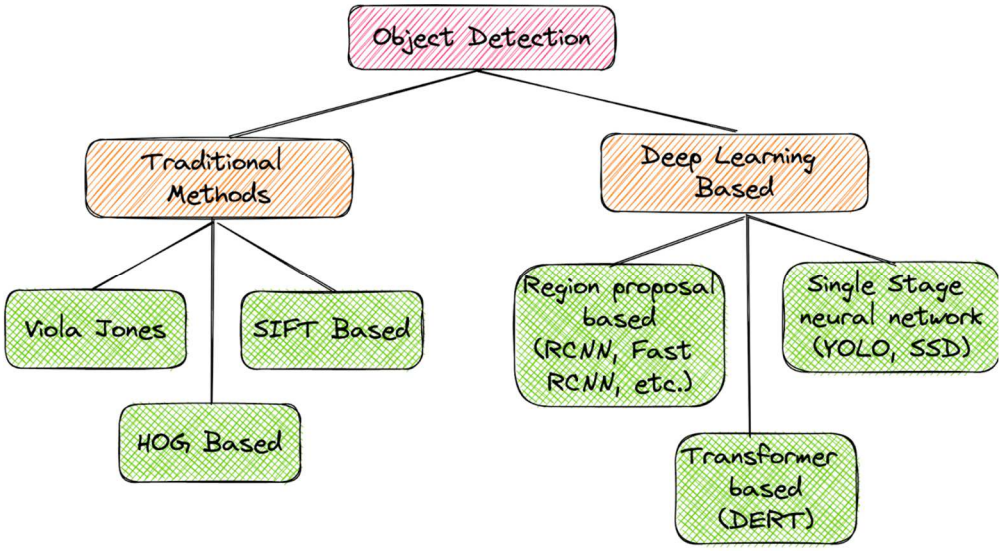


Fig. 2.2 A taxonomy of object detection algorithms [4].

1.2. Motion Tracking

Motion tracking also referred to as "*object tracking*", is the process of identifying and following the movement of a specific object (i.e., single object tracking), or several objects (i.e., multiple object tracking), over the course of a video sequence. A survey paper [5], published in 2022, found that motion tracking has become an important research topic in computer vision. In this thesis, we focused on MOT (*Multiple Object Tracking*) since it is an essential component of many CV applications, including video surveillance (see Fig. **2.3**), sports analysis, traffic monitoring, robotics, and autonomous vehicles.

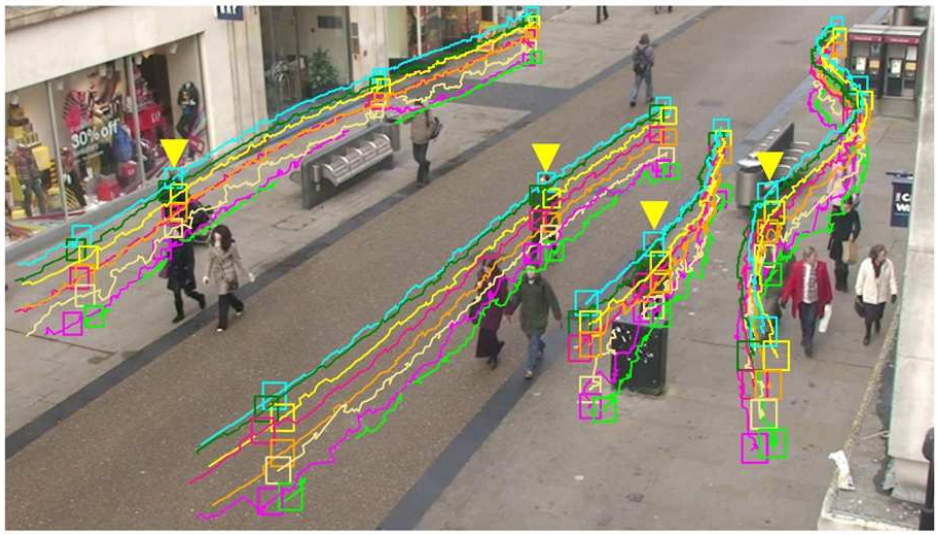


Fig. 2.3 An example of tracking people in public places [6].

Some issues and challenges

Despite significant progress in the field of MOT, several challenges remain. The following are some of the common issues that arise in MOT [7, 8]:

- **Occlusion**: occurs when something we want to see is entirely or partially hidden or occluded by another object in the same frame. It is challenging to handle especially in crowded environments where ground-truth bounding boxes often overlap each other.
- **Object re-identification**: is the process of identifying objects that have disappeared from the camera's field of view and then reappear later. Object re-identification is challenging, as objects can change their appearance due to factors such as lighting conditions and pose.
- **Motion blur**: occurs when objects move quickly, and their motion is captured by a camera's long exposure time, resulting in blurred images. Motion blur can make it challenging to track objects accurately since it alters their appearance.
- **Scale variation**: occurs when objects move closer or further away from the camera, resulting in changes in their apparent size. Scale variation is a significant challenge in MOT as it requires estimating the distance between the camera and the object in order to maintain accurate object localization.

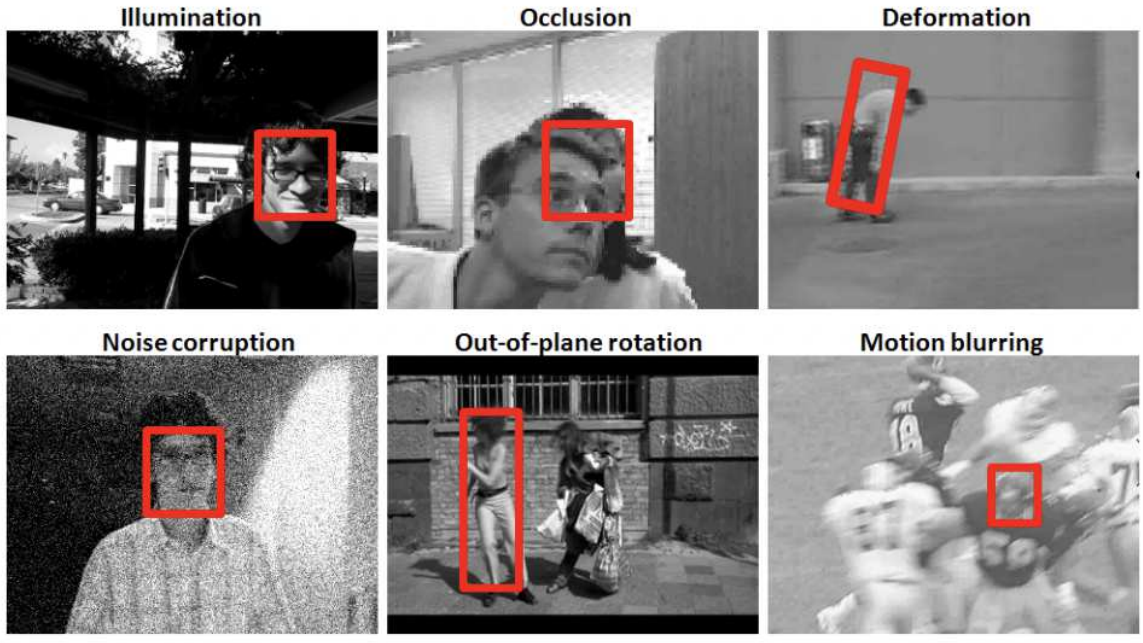


Fig. 2.4 Illustration of some challenging appearance changes in object tracking [9].

To address these challenges, researchers have developed a range of algorithms that combine multiple cues, such as appearance, motion, and geometry, to improve tracking performance.

Common tracking methods

Several techniques have been proposed for object tracking. They are classified into three main categories: *feature-based*, *model-based*, and *learning-based* methods. In this section, we will discuss these different methods in detail.

a. Feature-based methods

Feature-based methods are one of the most widely used methods for motion tracking. They cover all methods for tracking an object's motion over time using image features such as corners, edges, or blobs [10]. The features are detected in the first frame of the video sequence, and their motion is tracked over time. These methods are distinguished by their capability to track objects despite partial occlusions, changes in lighting conditions, and other types of noise, making them suitable for a wide range of applications. Here are a few examples of feature-based tracking algorithms: Lucas-Kanade method, Kanade-Lucas-Tomasi feature tracker, and optical flow.

b. Model-based methods

Methods known as "*model-based*" rely on the use of a mathematical model to represent the object or objects being tracked. The model is then used to estimate the object's position and shape in each frame of the video sequence, allowing the object to be tracked over time [11]. Additionally, these models could be combined with image features to improve tracking accuracy. Examples of model-based motion tracking methods include active contours, particle filters, Kalman filters, and mean shifts.

c. Learning-based methods

Learning-based methods refer to techniques that use ML and DL algorithms to track objects in a video sequence. These methods rely on the ability of deep neural networks to learn features and representations of the object to be tracked, allowing them to handle variations in appearance, occlusion, and lighting conditions [12]. Examples of such methods are Siamese networks, correlation filters, and R-CNNs. Other prominent methods include YOLO, SORT (*Simple Online and Realtime Tracking*), and DeepSORT.

To sum up, object tracking is a complex task that requires the use of different methods to achieve high tracking performance. Hybrid tracking combines multiple methods to improve tracking performance. Interestingly, recent developments in DL have led to the development of reliable motion-tracking algorithms that can effectively address new challenges such as occlusions, background clutter, lighting changes, and large movements or deformations of the tracked objects. Furthermore, these algorithms are capable of being trained on enormous amounts of data to improve their performance.

ARTIFICIAL INTELLIGENCE

Artificial intelligence (AI) is a subfield of computer science and engineering that focuses on building machines that are capable of performing tasks that ordinarily require human intelligence, such as learning, problem-solving, decision-making, and perception [13]. While there have been several definitions of AI over the past decades, we refer to the one found in one of the leading textbooks on the subject:

AI is the ability of a machine to independently replicate processes typical of human cognitive function in *deciding* on an *action* in response to its *perceived environment* to achieve a predetermined *goal* [14].

In today's world, we are surrounded by computer-based AI technologies, including personalized advertising, search engines, speech, and facial recognition, and more. AI has been particularly revolutionizing the field of object tracking and detection, and it is anticipated that it will play a significant role in further advancements in this field.

2.1. Machine Learning

Machine learning (ML) refers to the development of algorithms and statistical models that enable computers to learn from data and make predictions or decisions without explicit instructions [15]. To do so, the algorithms use statistical and probabilistic techniques to understand the characteristics of the data and make predictions or decisions accordingly. These algorithms are designed to continuously improve their performance by adjusting their actions based on feedback and the accuracy of the predictions [16].

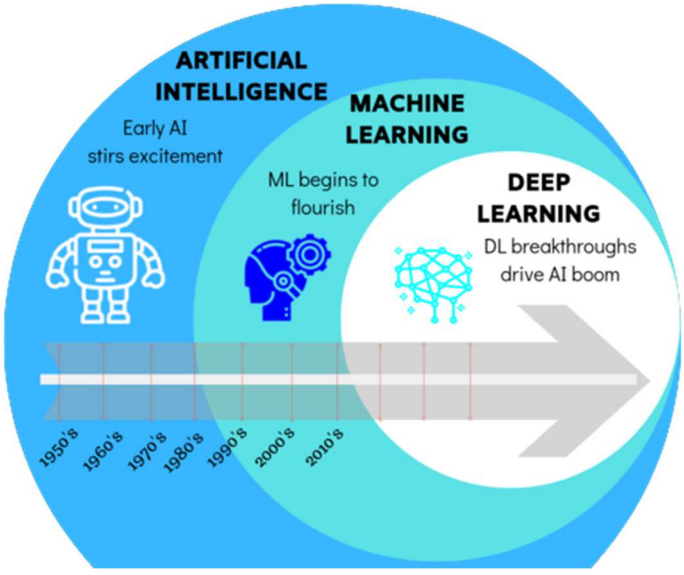


Fig. 2.5 The relationship between AI, ML, and DL (edited from [17]).

ML techniques can be broadly classified according to the type of label and the type of feature [18]. Based on labels, ML can be classified into three paradigms: *supervised*, *unsupervised*, and *reinforcement learning*. Whereas based on features, ML can be classified into *handcrafted* or *non-handcrafted* feature-based techniques.

2.2. Deep Learning

Deep learning is a branch of ML that focuses on neural network models with numerous layers. It is deep in precisely the sense that its models learn many layers of transformations [19]. At its core, DL consists of computing hierarchical features or representations of the observational data, where the higher-level features or factors are defined from lower-level ones. Several unsupervised and supervised feature learning algorithms have been used, along with neural networks, hierarchical probabilistic models, and other techniques [20].

CNN

The Convolutional Neural Network (CNN) is one of the various types of artificial neural networks that has demonstrated outstanding performance in many CV applications. This method is extremely popular due to the abundance of open-source software and research on it, as well as the large number of papers that have been written about it [21]. A CNN consists of four layers, as illustrated in Fig. **2.6**.

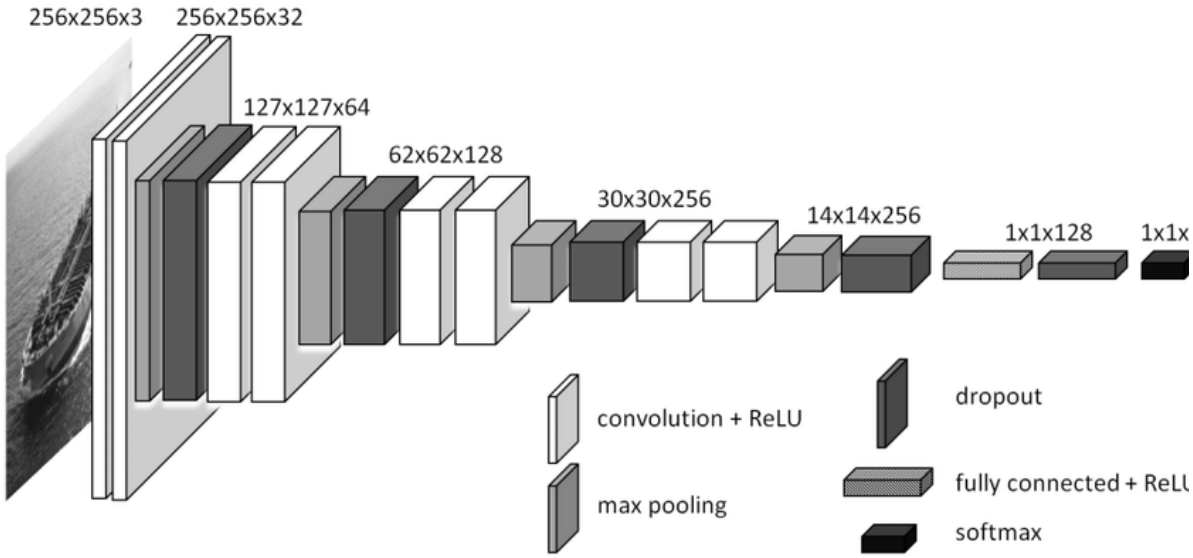


Fig. 2.6 A typical architecture of a CNN [22].

The convolutional layer is the first layer while the fully connected layer is the last one. In the following sections, we will describe each layer and explain its respective role.

a. Convolution layer

This layer is composed of a set of kernels used to extract the features from inputs by sliding over them. The kernel will be generated randomly at the beginning and will be adjusted based on inputs.

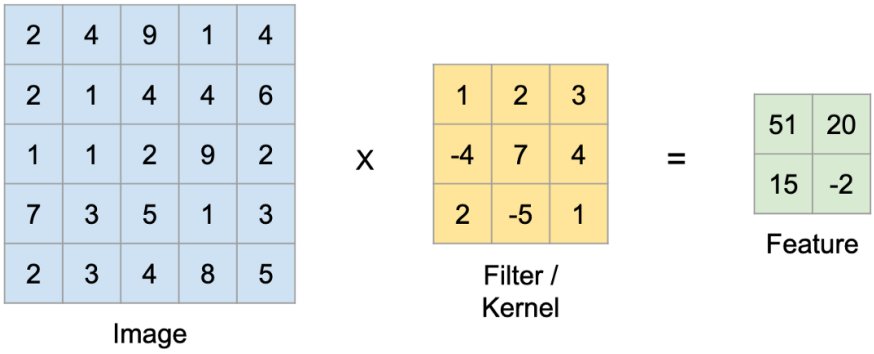


Fig. 2.7 An example of the convolution operation [23].

b. Pooling layer

This layer is usually used just after the convolution layer to reduce the number of parameters and the computation time by taking the max, min, or mean of regions of the inputs. Fig. **2.8** presents an example of a max pooling operation:

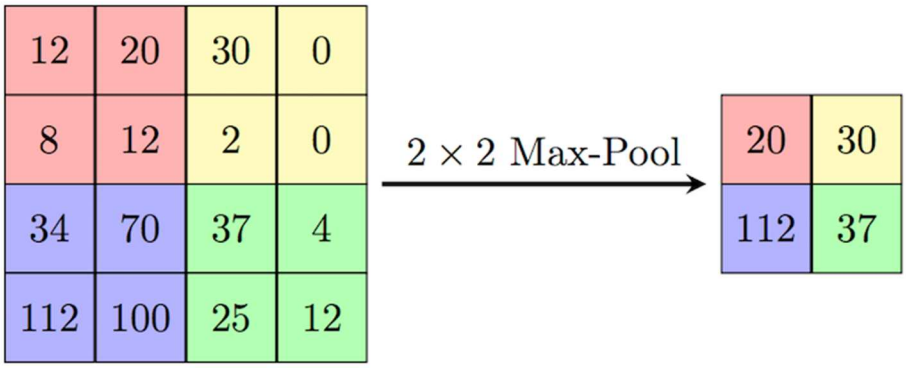


Fig. 2.8 An example of the max pooling operation [24].

c. Relu

An activation function called ReLU (*Rectified Linear Unit*) is utilized in multi-layer neural networks to introduce non-linearity [25]. It only triggers when the input surpasses a specific threshold. In case the input is negative, the function returns zero as output to prevent negative values in images.

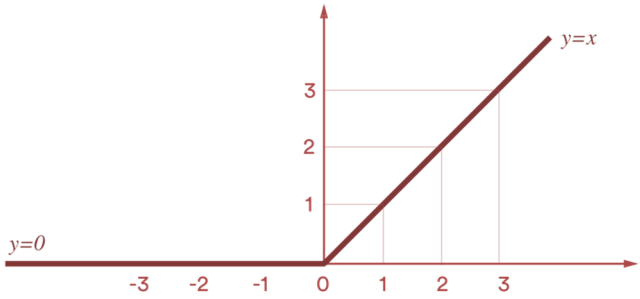


Fig. 2.9 The ReLu activation function [26].

d. Fully connected layer

In a fully connected layer, each neuron receives input from all activations in the previous layer and computes a weighted sum of these inputs [27]. The result is passed through a non-linear activation function to produce the output. The weights and biases in the fully connected layer are learned during the training process, allowing the network to learn complex relationships between the input and output.

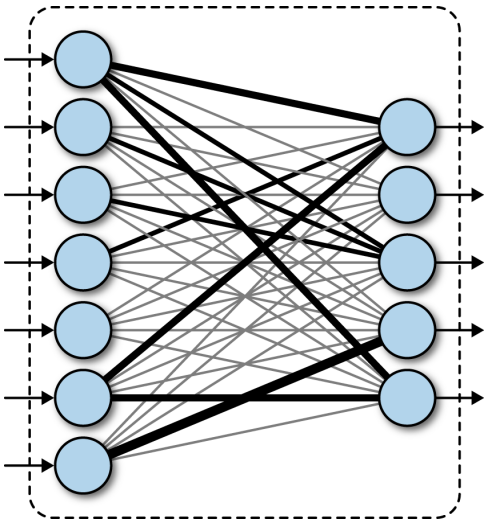


Fig. 2.10 Illustration of a fully connected layer [28].

R-CNN and fast R-CNN

R-CNN (*Regions with CNN features*) was introduced in 2014 by Ross Girshick et al. [29] as a method for object detection that involves generating region proposals and classifying each proposal using a CNN (see Fig. **2.11**).

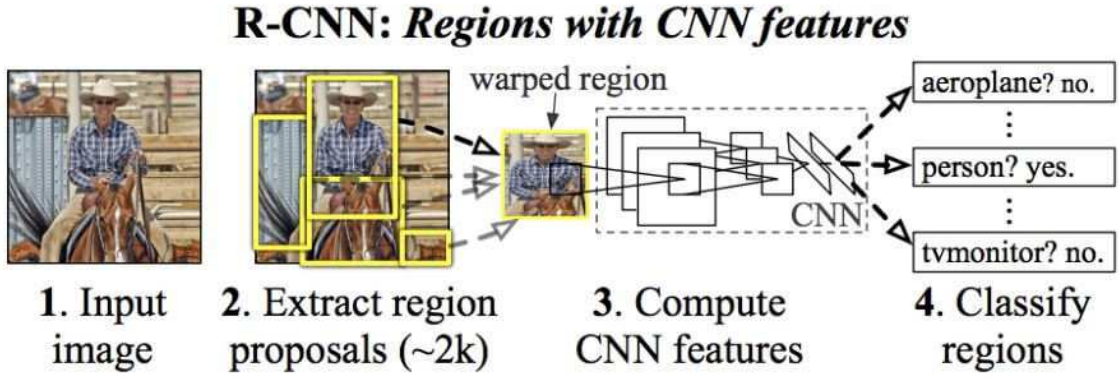


Fig. 2.11 A typical architecture of R-CNN [29].

The basic idea behind R-CNN is to first generate a set of region proposals that may contain objects of interest. These proposals can be generated using methods such as selective search or edge boxes. Then, for each proposal, a CNN is used to extract features from the region, and these features are fed into a separate fully connected layer to classify the proposal and predict a bounding box for the object. However, one major drawback of R-CNN is that it is slow, as it requires a separate CNN forward pass for each region proposal. This can be time-consuming, especially for large images with many proposals.

To address this issue, Girshick et al. [30] introduced Fast R-CNN in 2015, which is an improvement of R-CNN that uses a single CNN to classify all region proposals in a single pass. It does this using an RoI pooling layer that resizes each region proposal to a fixed size and maps it to a fixed-length feature vector that can then be classified using a single fully connected layer. In addition, Fast R-CNN uses bounding box regression to refine the location of each object. Given an initial bounding box for an object, Fast R-CNN uses the predicted bounding box offsets to adjust the box to better fit the object.

Overall, R-CNN and Fast R-CNN are important advances in motion tracking and have been widely used and cited in the CV community. Indeed, object tracking algorithms need to not only precisely detect objects of interest but also operate in the shortest possible time, especially in real-time applications.

2.3. OpenVINO Toolkit

OpenVINO (*Open Visual Inference and Neural Network Optimization*) is an Intel-developed toolkit that aims to accelerate deep learning workloads on a variety of hardware platforms [31]. It includes a range of tools and libraries for optimizing, deploying, and running DL models.

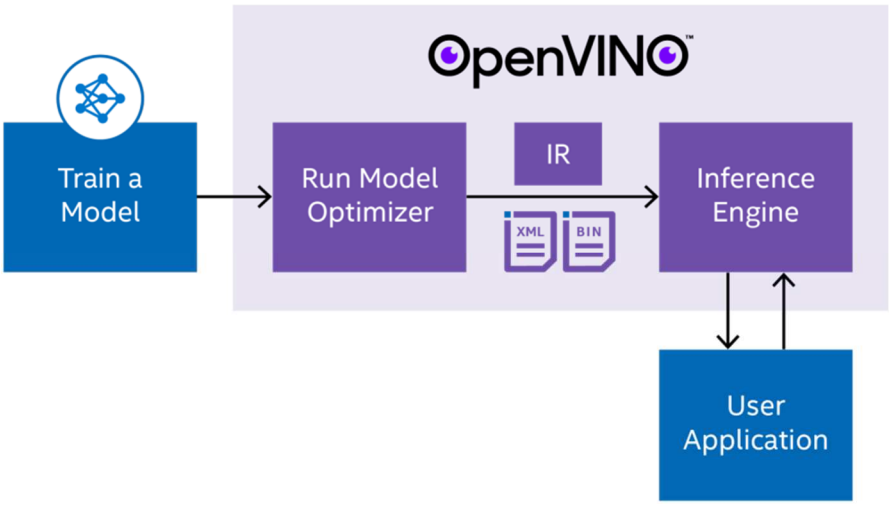


Fig. 2.12 The OpenVINO toolkit workflow [32].

The following are some key features of the OpenVINO toolkit:

- **Model optimization**: OpenVINO includes tools for optimizing deep learning models to work more effectively across a variety of hardware platforms. This can involve techniques such as quantization, pruning, and fusion of layers.
- **Hardware acceleration**: OpenVINO supports the acceleration of deep learning models on a range of hardware platforms, including CPUs, GPUs, and VPUs. Users can then select the hardware that best meets their needs and constraints.
- **Pre-trained models**: OpenVINO includes a set of pre-trained models that can be used for tasks such as object detection and classification [33], [34]. These models can be fine-tuned for specific use cases or used as-is for common tasks.

INTERNET OF THINGS

3.1. Edge Computing

Edge computing is a computing model that supports the storage and processing of data in the same location where it is generated or collected, avoiding the need to send it to data centers [35]. As a result, this immediate processing of data on the edge has solved the ongoing latency issue of the cloud. The following issues have been also taken into consideration:

- **Privacy and security:** since the data does not take a long way on the edge, the possibility of it being hacked or its privacy violated is reduced.
- **Operational costs:** edge computing allows a decrease in operational costs by requiring less internet bandwidth and replacing data centers with edge devices.
- **Speed:** data are processed quickly as a result of a decrease in network congestion and the time required for data transfer.

- **Availability:** unlike cloud-based solutions, which rely on the availability of an internet connection, edge computing solutions allow some functions and data to be stored locally.

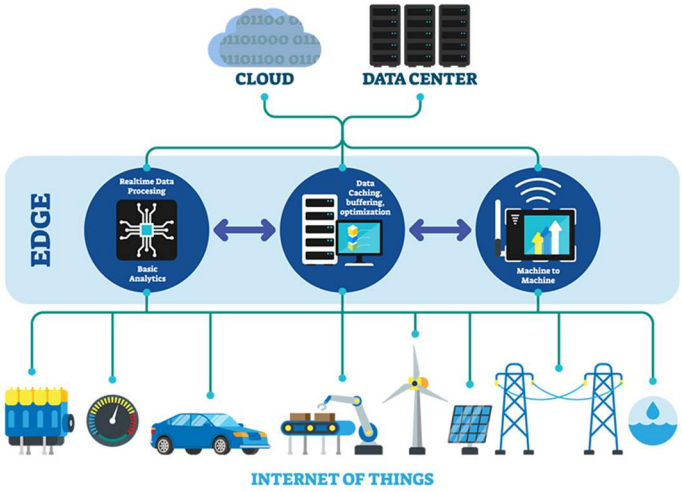


Fig. 2.13 An abstract illustration of edge and cloud computing [36].

Edge computing solutions enable the development of automated processes for object detection and tracking that can quickly generate valuable insights. The ability to operate with speed and agility is critical for various CV applications such as security risk monitoring and compliance enforcement. Edge computing simplifies the deployment of these solutions, resulting in increased accuracy and efficiency.

3.2. Raspberry Pi

The Raspberry Pi (RPi) is a small computer about the size of a credit card that is capable of performing advanced computational operations as well as any other computer [37]. Its latest version, the RPi v4.0 B, can play 4K video at 60 frames per second and boost the Pi's media center capabilities, making it an excellent option for CV-based projects.

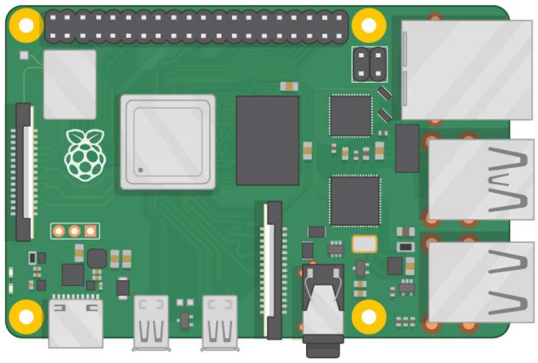


Fig. 2.14 The RPi v4.0 B [38].

3.3. Intel Movidius Neural Compute Stick

Overview

The Neural Compute Stick (NCS) is a modular DL accelerator that fits inside a standard USB 3.0 stick, as shown in Fig. **2.15**. It allows users to access DL interface applications for offline AI prototyping without the need for a network connection. In addition, the Intel Movidius Neural Compute Stick is well-suited for edge computing applications as it does not require a cloud connection, thus improving system security. The stick also supports OpenVINO™, a toolkit that facilitates faster solution development and more efficient deployment.

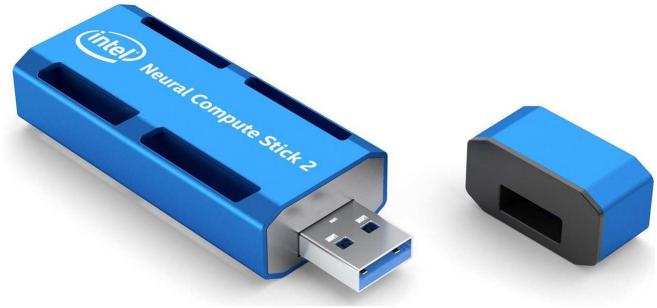


Fig. 2.15 The Intel Movidius NCS v2 [39].

The newer version NCS v2 offers plug-and-play simplicity, support for common frameworks, and out-of-the-box sample applications [40]. It is based on the Intel® Movidius™ Myriad™ X VPU, which contains two neural compute engines and is eight times faster than other previous VPUs in terms of performance (it can perform one trillion operations per second) [39].

The hardware architecture of NCS

The Intel® Movidius™ Myriad™ X VPU is a powerful processor designed specifically for neural network inferencing [41]. It serves as the main processor in the NCS and has a large amount of LPDDR DRAM memory, allowing it to store big neural network models and perform computations quickly. It also includes image and vision accelerators, which are capable of performing various tasks such as image pre-processing, object detection, and image classification.

As depicted in Fig. **2.16**, the aforementioned accelerators are designed to cooperate with the SHAVE (*Streaming Hybrid Architecture Vector Engine*) processors to provide efficient neural network performance. The SHAVE processors are 12 specialized vector processors responsible for performing the necessary calculations to execute neural network operations. They are optimized for running DL algorithms and are capable of performing complex computations at high speeds.

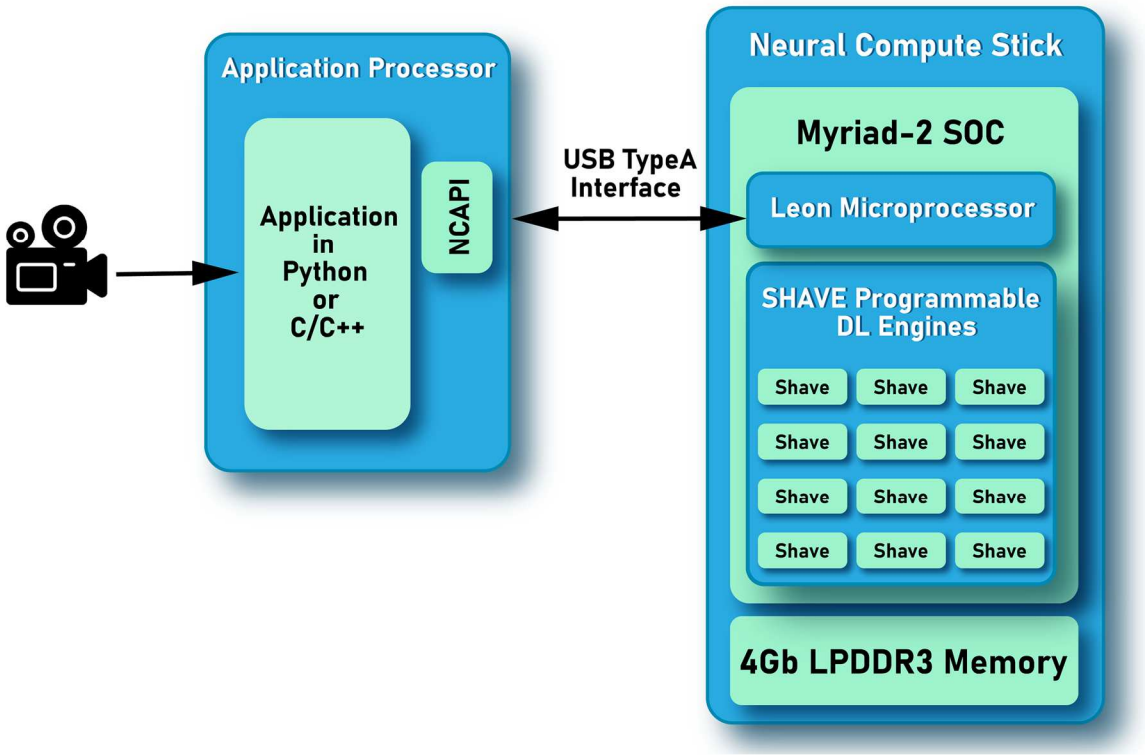


Fig. 2.16 The NCS architecture (edited from [41]).

The software workflow of NCSDK

The RPi can easily add AI capabilities thanks to the NCSDK, which includes tools for converting trained models into a format the NCS can understand and an API for

loading these models onto the NCS at runtime. It is important to note that the NCS is only running the model for inference purposes and not training it in any way.

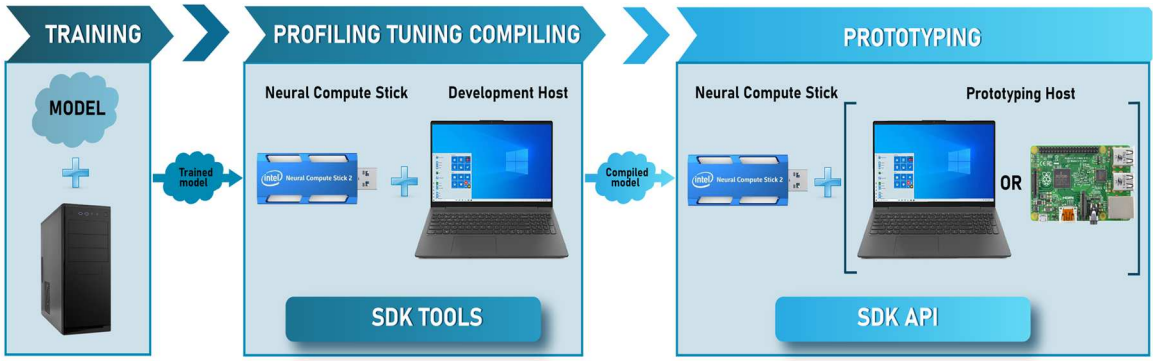


Fig. 2.17 The typical workflow for development with NCSDK (edited from [41]).

As shown in Fig. **2.17**, the workflow for the development using NCSDK is divided into two important phases as follows:

a. Profiling, tuning, and compiling

As illustrated in Fig. **2.18** (a), during this phase a compiler converts the model written in a programming language into a graph that the NCS can process.

b. Prototyping

As illustrated in Fig. **2.18** (b), the essential task in this phase is the loading of the generated model (graph) and the firmware into the RAM of the NCS.

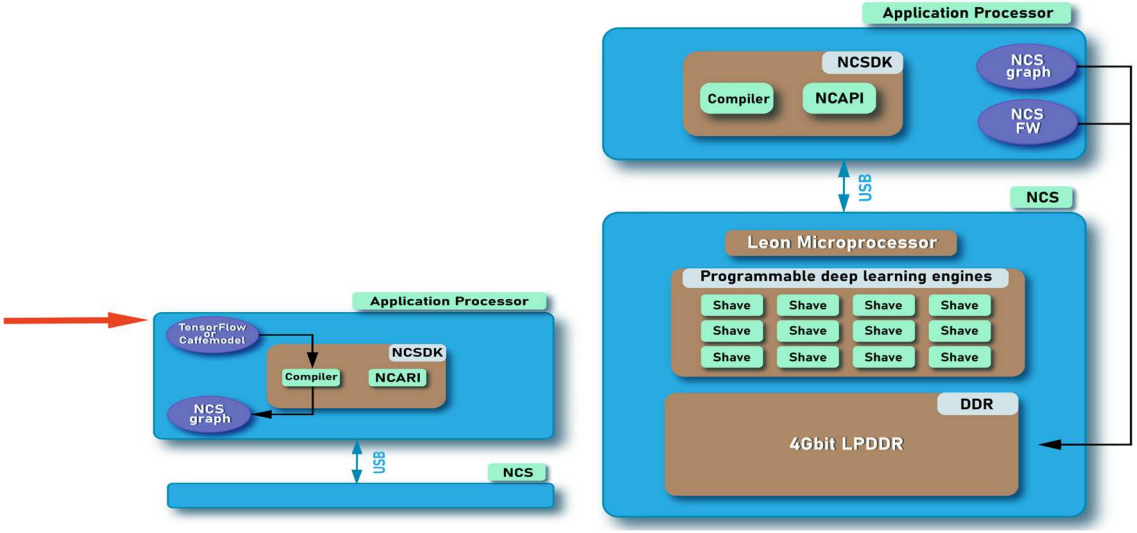


Fig. 2.18 The NCSDK software workflow [41].

(a) in the left is the first phase, whereas (b) in the right is the second phase.

CONCLUSION

In this chapter, we focused on presenting the most important techniques and methods used in developing our proposed system, where we defined CV techniques, including object detection within images and motion tracking in streamlined videos. We also presented AI and ML technologies, with a focus on DL methods since they are the ones we used. In the end, we described the edge devices we used to deploy our model, namely the RPi and Intel NCS.

In the next chapter, we are going to review the literature and present the current motion-tracking solutions, along with a discussion of their advantages and limitations.

"The biggest room in the world is the room for improvement."

– Helmut Schmidt

3

Literature Review

INTRODUCTION

Over the past few years, computer vision and object tracking have made significant development, with researchers investigating various techniques to address object tracking challenges. These methods range from conventional to cutting-edge algorithms. Despite this progress, object-tracking solutions still have significant shortcomings in terms of efficiency and the ability to keep track of objects.

In this chapter, we will examine both traditional and deep learning-based object tracking methods and explore how combining them in hybrid approaches can lead to better results. Furthermore, we will briefly review hardware solutions developed for optimizing object-tracking algorithms for edge deployment.

OBJECT TRACKING ALGORITHMS

4.1. Traditional Motion Tracking Algorithms

This section will introduce three widely used algorithms for object tracking: Mean Shift, Kalman filter, and Lucas-Kanade optical flow. A detailed description of each algorithm will be presented, along with a discussion of their respective strengths and limitations.

Mean shift

In the context of motion tracking, the mean shift algorithm can be used to track the movement of an object by finding the mode of the object's color distribution in each frame of a video [42]. To do this, the algorithm divides the image into a set of bins, each representing a range of color values, and computes the mean value of the color distribution within each bin. It then shifts the bin centers towards this mean value and continues to iterate until the bin centers converge, at which point the mean shift algorithm returns the mode of the distribution as the location of the object being tracked.

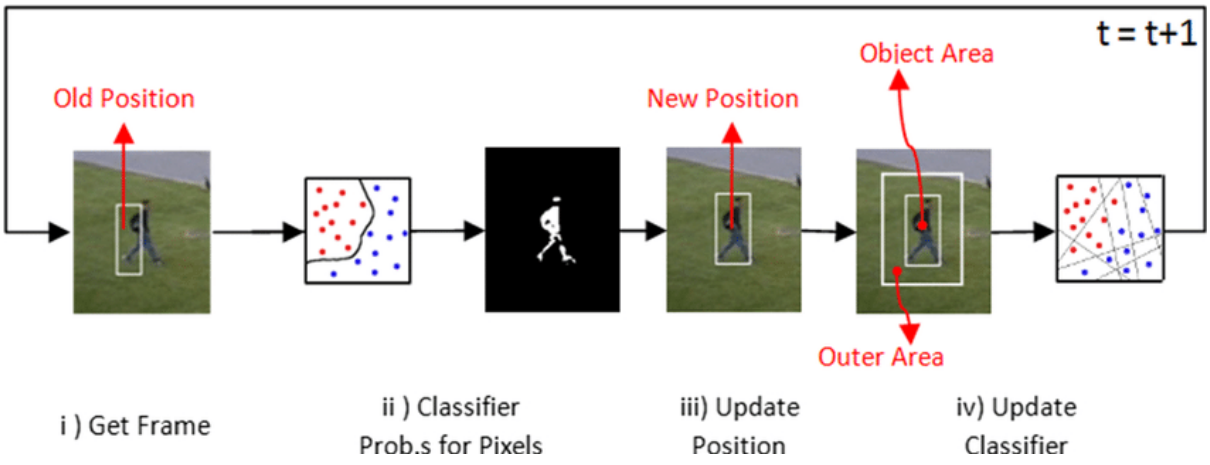


Fig. 3.1 Illustration of a classifier-based mean-shift tracking framework [43].

One advantage of this algorithm is that it does not require any prior knowledge about the object being tracked, such as its shape or size. It can also handle occlusions and changes in lighting fairly well, making it a robust method for motion tracking in challenging environments [44]. However, it can be computationally intensive and may not always produce accurate results in situations where the object being tracked has complex motion or undergoes significant changes in appearance.

Lukas-Kanade optical flow

Lukas-Kanade optical flow is a widely-used algorithm for computing optical flow in computer vision. It estimates the motion of points or small regions between consecutive frames of an image sequence by assuming that the motion is locally smooth and that the image intensity values at corresponding points in the two frames should have a similar gradient [45]. The algorithm

solves a system of linear equations for each point or region to estimate the motion by minimizing the sum of squared differences between the two frames.

Fig. **3.2** illustrates the difference between the standard optical flow (the image on the left) and that of Lukas-Kanade (on the right).

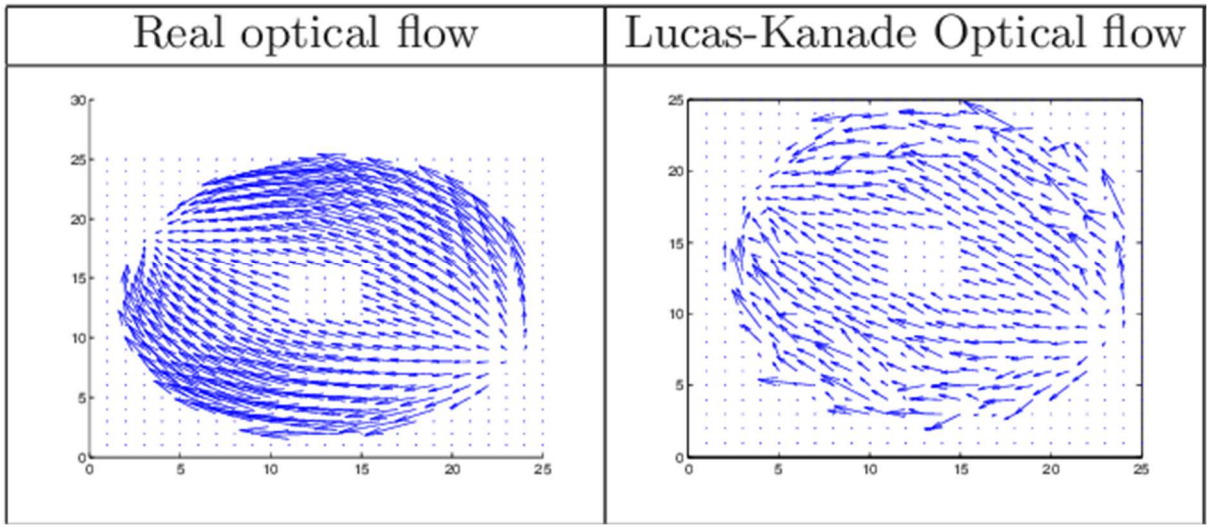


Fig. 3.2 Illustration of standard optical flow and Lukas-Kanade optical flow [46].

Lukas-Kanade optical flow is computationally efficient and robust to small variations in image intensity, making it suitable for real-time applications [47]. However, its accuracy decreases in scenes with complex motion or larger discontinuities. Moreover, the algorithm is sensitive to image noise and can only estimate 2D motion.

Kalman Filter

In the context of motion tracking, the Kalman filter predicts the location of an object in the next frame based on its previous location and velocity and then updates the prediction with the measurement from the current frame [48]. The filter consists of two main steps: the prediction step uses the system model to estimate the current state based on the previous state, while the update step corrects the prediction based on the measurement from the current frame.

Fig. **3.3** demonstrates the usage of the Kalman filter for tracking the ball of a table tennis game played by two kids. The depicted equation is used to estimate the position and velocity of the ball and track its trajectory over time.

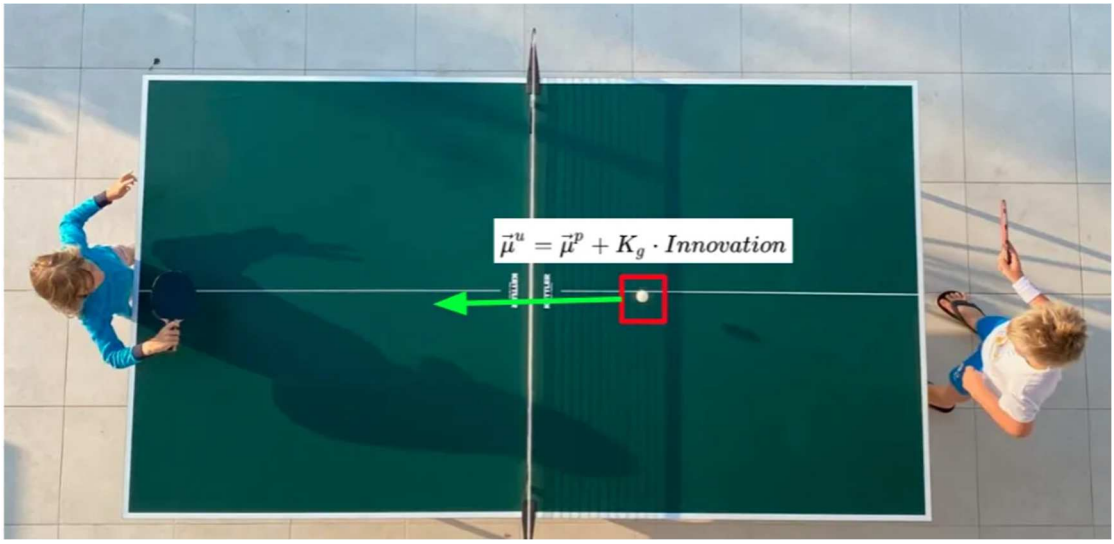


Fig. 3.3 The Kalman filter ray equation [49].

Kalman filter has several advantages such as providing an efficient and reliable estimation of the state variables, handling noise and uncertainty, and being computationally efficient [50]. However, it also has limitations, such as the assumption of linear and Gaussian noise models, the need for a good initial estimate of the state variables, and the complexity of implementation. Moreover, its performance degrades significantly in the presence of nonlinearities and non-Gaussian distributions.

4.2. Deep Learning-based Motion Tracking Algorithms

In this section, we will delve into some of the most popular deep learning-based object detection and tracking algorithms that have revolutionized the field of computer vision. The focus will be on two of the most widely used object detection models, namely YOLO and SSD. In addition, we will examine two cutting-edge DL-based object tracking algorithms, DeepSORT and Siamese, and highlight their key features and benefits.

DL-based detectors

a. YOLO

YOLO (*You Only Look Once*) is a fast object detection algorithm that uses a single CNN to predict the bounding boxes and class probabilities for objects in an image [51]. It works by dividing an image into a grid of cells and using each cell to predict the presence of objects in the image. Each cell is responsible for predicting a fixed number of bounding boxes and class probabilities. YOLO then uses a combination of these predictions to detect and classify objects in the image.

One of the main advantages of YOLO is its high speed, which makes it suitable for use in real-time applications [52]. It is also able to predict multiple objects in an image and can handle a

wide range of object scales, aspect ratios, and locations. However, YOLO has some limitations in terms of accuracy and may not be as precise as some of the other object detection algorithms.

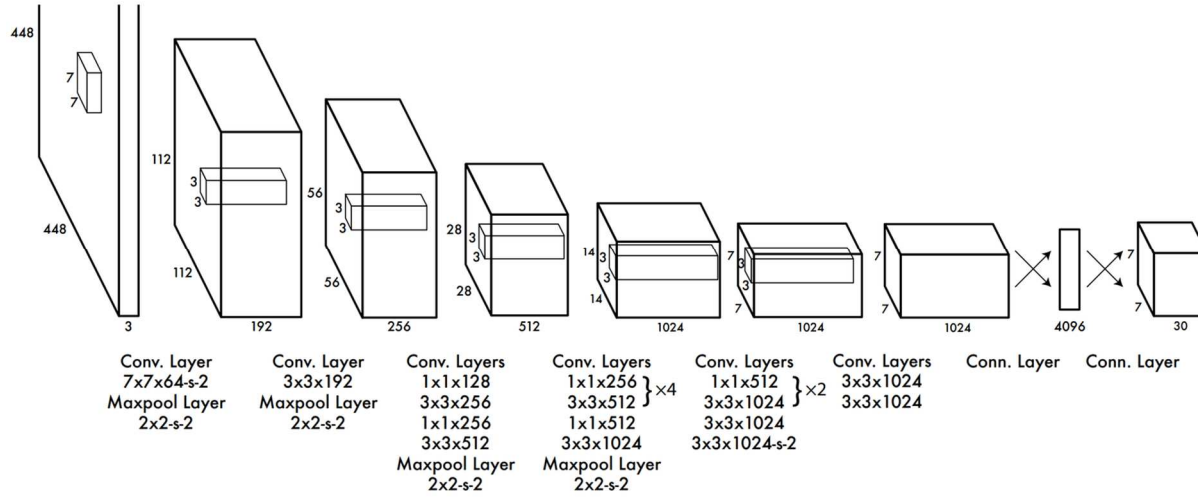


Fig. 3.4 The YOLO architecture [53].

YOLOv8 [54] is the latest version of YOLO and was designed to be more accurate and efficient than previous versions. It was tested on the COCO object detection benchmark and achieved an AP of 52.9% on the test-dev set. To address the issue that YOLO can be computationally expensive, a lightweight version has been developed, named YOLOv8n, in which fewer layers were used to achieve faster inference times and lower memory usage. Interestingly, this lightweight version can still achieve high accuracy, making it suitable for applications where computational resources are limited.

b. SSD

SSD (*Single Shot Multibox Detector*) is a type of object detection algorithm that is fast and accurate. It is a one-stage detector that uses a single deep neural network to predict the bounding boxes and class probabilities for objects in an image [55]. One of the key features of SSD is its ability to predict multiple bounding boxes for each object in an image. It uses anchor boxes, which are predefined bounding boxes with different aspect ratios and scales, to generate these predictions. The network is trained to predict the offsets for each anchor box and the confidence scores for each class [56].

In addition, SSD can be trained end-to-end, meaning that the network can learn to predict bounding boxes and class probabilities directly from the raw input images, without the need for region proposal algorithms or additional post-processing steps. This makes it faster and more efficient than some other object detection methods [57].

Overall, SSD is a popular choice for object detection tasks because of its speed, accuracy, and flexibility. It has been used in a variety of applications, including autonomous vehicles, surveillance systems, and image and video analysis.

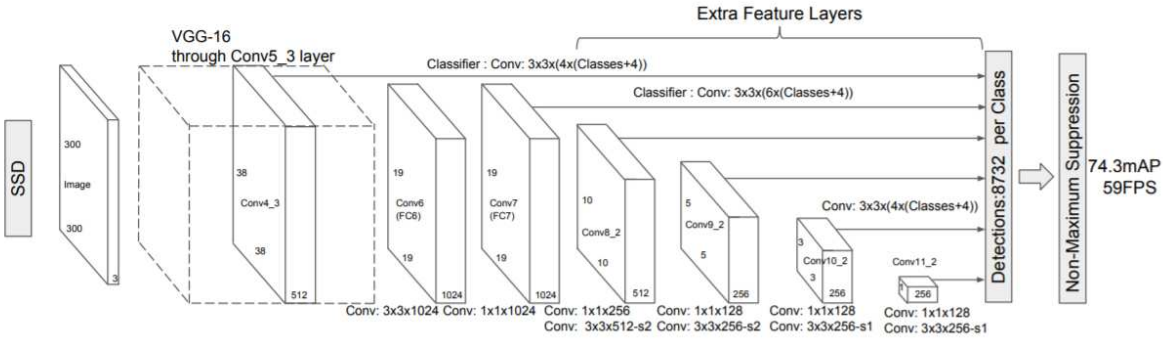


Fig. 3.5 The SSD architecture [56].

DL-based trackers

a. DeepSORT

DeepSORT (*DL-based SORT*) is a real-time object tracking algorithm that combines the popular SORT (*Simple Online and Realtime Tracking*) algorithm with a deep neural network for more accurate and robust object tracking. The SORT algorithm is a simple but effective object-tracking algorithm that uses a combination of motion and appearance features to track objects in a video stream [58]. However, SORT can struggle when objects have occlusions or are moving quickly, leading to inaccurate or lost tracks. In addition to its improved accuracy and robustness, DeepSORT has also been shown to outperform other state-of-the-art object-tracking algorithms in various benchmarks and real-world scenarios [59].

Furthermore, DeepSORT has enabled the development of advanced MOT systems that can be used in a variety of applications, such as crowd analysis, traffic monitoring, and sports analytics [60]. Its effectiveness in these applications has made DeepSORT a valuable tool for researchers and practitioners alike. Several studies have demonstrated the effectiveness of DeepSORT in various tracking scenarios. For instance, one study [61] evaluated the performance of DeepSORT on a dataset of crowded pedestrian videos and found that it outperformed other state-of-the-art tracking algorithms in terms of accuracy and robustness. Another study [62] evaluated the performance of DeepSORT in a tracking-by-detection framework for tracking soccer players in broadcast footage and found that it achieved state-of-the-art results.

b. Siamese

A Siamese network is a type of neural network architecture that is named after the *Siamese twins*, who were conjoined twins with identical or nearly identical features [63, 64]. The Siamese network shares a similar concept in that it uses identical subnetworks that share the same weights

and parameters to process different inputs and compare their similarities or differences, as depicted in Fig. **3.6**. The architecture of a Siamese network typically involves feeding the inputs through identical subnetworks, which learn to extract high-level features from the inputs. The output of each subnetwork is then fed into a comparison layer, which computes a similarity score between the inputs. The similarity score can be used to determine whether the inputs are similar or dissimilar.

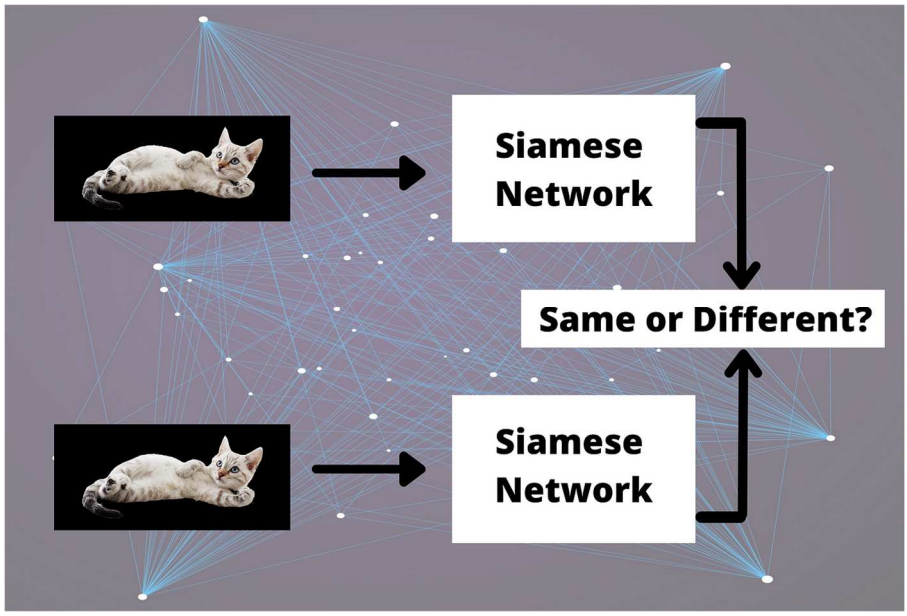


Fig. 3.6 Siamese network architecture [65].

The advantage of using a Siamese network for one-shot image recognition is that it can learn discriminative features from small amounts of training data, and can generalize well to new objects [66]. This is achieved through weight sharing between the two identical subnetworks, which extract useful features from the input images and produce a similarity score that indicates whether the images belong to the same class or not.

RELATED WORKS

5.1. Overview

Deep learning-based visual multi-object tracking techniques have gained significant attention in recent years due to their ability to accurately track multiple objects in real-time video sequences. In this overview, we will cover three popular DL-based tracking algorithms.

Tracking by detection algorithms

One popular approach to MOT is the TBD (*Tracking by Detection*) framework that first detects all objects of interest in each frame of the video using a pre-trained object detector (such as YOLO or Fast R-CNN) and then associates them with the detected objects in the previous frame to form tracks. The association could be made using Kalman filters. TBD is simple,

effective, and widely used in practical applications [7, 67]. However, its performance heavily depends on the accuracy of object detection and linking algorithms.

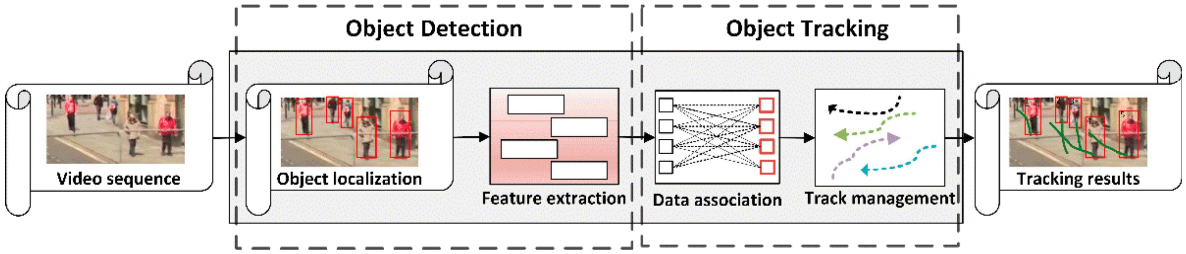


Fig. 3.7 The common workflow of the TBD framework [67].

Joint detection and tracking algorithms

JDT (*Joint Detection and Tracking*) is an end-to-end tracking algorithm that jointly performs object detection and tracking in a single step. The algorithm uses a Siamese network architecture to compare the features of the current frame with the features of the previous frame to estimate the object's position and velocity. The tracked objects are then associated with the detections using a linking algorithm. JDT can take advantage of temporal information to improve both detection and tracking accuracy [7, 67]. However, it requires a large amount of training data and is computationally expensive compared to TBD.

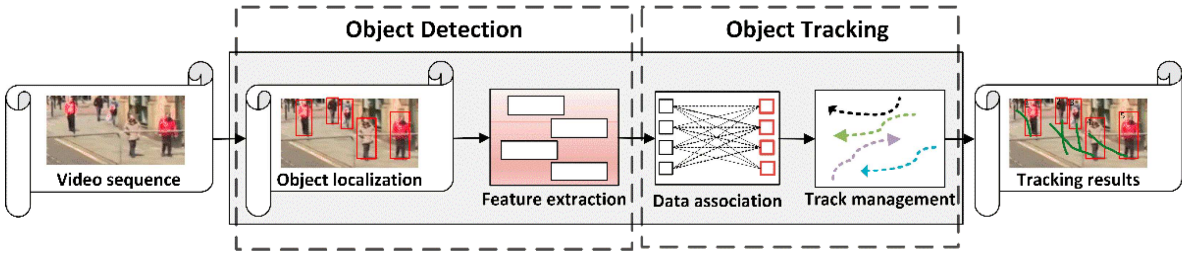


Fig. 3.8 The common workflow of the JDT framework [67].

Transformer-based algorithms

Transform-based tracking algorithms are a group of techniques that use spatial or frequency transforms to represent the target object and track it over time. The algorithm first computes a feature representation of the target object in the first frame and then tracks it by comparing the feature representation in subsequent frames [7, 67]. Transform-based tracking algorithms use encoder-decoder architecture to obtain global and rich contextual interdependencies for tracking, as illustrated in Fig. **3.9**. They are computationally efficient and can handle large-scale tracking tasks. However, the parameters are too large and the algorithm may suffer from accuracy issues when the target object undergoes large-scale deformations.

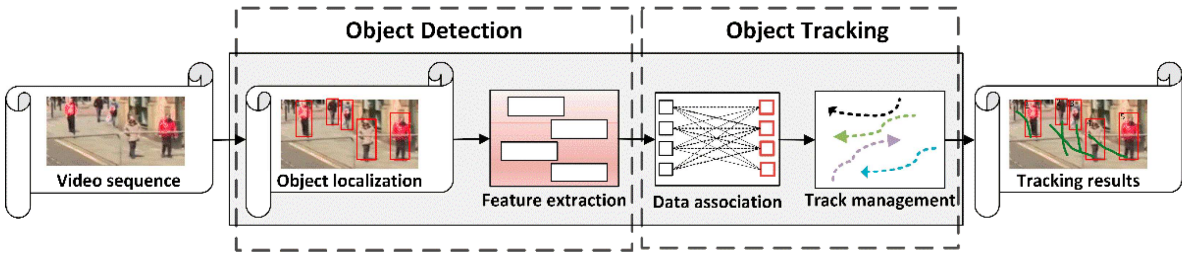


Fig. 3.9 The encoder-decoder transformer architecture [67].

To sum up, DL models are effective in object detection and tracking tasks, leading to many proposed algorithms for MOT. In the next section, we will review and analyze some recent scientific papers published between 2019 and 2022 that propose deep learning models to track multiple objects. We specifically focused on those based on the TBD framework since they are based on the same framework as our proposed model.

5.2. Review And Discussion

As explained above, in TBD algorithms the detection is done using any DL model. The difference between studies is in the algorithm used to keep track of the trajectory of objects of interest. Different DL-based detectors and association algorithms were tested in this regard. The commonly used object detectors were Fast R-CNN, SSD, and YOLO.

The authors in [68], introduced a new multi-scale detector that builds upon the SSD, which can effectively detect small objects by utilizing temporal information. The proposed approach successfully addresses the problem of re-identification of tracked objects when they are lost. Furthermore, the paper highlights the superiority of the CNN-based correlation filter as an affinity model over the key points matching-based approaches used previously. The experiments conducted on KITTI and MOT15 tracking benchmarks produced positive results, indicating the potential of the proposed approach.

In [69], Li et al. proposed an MOT technique that relies on the YOLO algorithm. First, the video stream undergoes multi-object detection using the YOLO algorithm. Once the target size, position, and other information are obtained, the depth feature extraction takes place, and noise data from irrelevant regions in the image is removed. This results in a reduction in the calculation complexity and time of feature extraction. The proposed algorithm was tested on the MOT-16 and MSR datasets, and the results demonstrate that it is effective for MOT.

In 2021, Chanho et al. [70] solved the problem of simultaneously considering all tracks during memory updating, with only a small spatial overhead, via a novel multi-track pooling module. They used the bilinear LSTM tracking framework to achieve online tracking performance.

Other researchers have also used DL-based trackers for MOT, including DeepSORT. According to Lin et al. [71], the TBD algorithm heavily depends on appearance information. In response, they proposed the HTA (*Hybrid Track Association*) algorithm that uses an incremental Gaussian mixture model (IGMM) to track the historical appearance distances of a target and includes this statistical data in the calculation of the detection-to-track association cost. The researchers tested the HTA algorithm on three MOT benchmarks and found that it effectively enhances target identification performance with only a minor impact on tracking speed.

In [72], the authors proposed a technique called NMS (*Non-Maximum Suppression*) to track occluded targets more efficiently. The approach involves keeping additional potential boxes that might contain the obscured target for trajectory-matching purposes while getting rid of unrelated boxes based on the IOU (*Intersection over Union*) measure between the predicted and identified boxes. The results of experiments conducted on the MOT17 and MOT20 datasets demonstrate the effectiveness of the approach in tracking occluded targets.

Overall, the reviewed studies indicate that DL models have been successful in MOT for autonomous driving. However, some limitations need to be addressed. One of the main limitations is the high computational cost of the proposed models, which may not be suitable for real-time systems. Another limitation is the dependence of some models on object size and the lack of object association between frames. This limitation may reduce the accuracy of the models in crowded

CONCLUSION

In this chapter, we presented some existing solutions in the literature of object detection and motion tracking, both traditional and modern algorithms with a brief description of the mechanism that they are based on and the results that researchers have obtained. We aimed from that to find the gaps and differences between these solutions and have a clear scope of the added value that we will present in the next chapters.

Part 2. Contribution

To effectively present our solution, we have created a chapter "**Proposed Solution**". In this chapter, we will provide an in-depth explanation of the theoretical side of our model, outlining the key steps involved in building it from training and dataset to optimization. By doing so, we hope to provide valuable context for our audience and help them understand the underlying principles behind our solution.

In the "**Obtained Results**" chapter, we will provide details on the programming languages and frameworks used in our solution, along with their respective versions. Additionally, we will present the results of both traditional and deep learning algorithms and compare their performance. We will also include information on the environment used for our experiments. Through this comprehensive analysis, we can evaluate the feasibility of our optimization efforts and provide a thorough assessment of our solution.

"Everything can be improved."

—Clarence W. Barron

4

Proposed Solution

INTRODUCTION

In the field of computer vision, motion tracking is an essential and challenging task with widespread applications in areas such as surveillance, robotics, and autonomous vehicles. As outlined in the previous chapters, existing motion-tracking models often struggle to perform accurately and efficiently on edge-based devices, such as the Intel Movidius NCS. To address this issue, we present a novel motion-tracking model that combines two powerful deep-learning models: the YOLOv8 object detector and the DeepSORT object tracker. Our model has been optimized to run efficiently on edge-based devices, while also being trained on a custom dataset that we have improved and annotated for this purpose. This chapter will introduce our proposed model and provide a comprehensive overview of its composite steps. Furthermore, we will delve into the model's training phase and elaborate on the optimization process.

6.1. Model Architecture

Fig. **4.1** presents the system architecture of the proposed motion-tracking model, which is based on the tracking-by-detection approach. Hence, the proposed approach consists of two phases: object detection and object tracking. In the first phase, the input video stream is processed by the YOLOv8 to detect objects in each frame of the video stream. Once objects have been detected, they are passed to the DeepSORT tracker, which assigns unique IDs to each object and tracks their positions across multiple frames. The tracker also incorporates motion prediction and Kalman filtering to improve the accuracy and stability of the tracks.

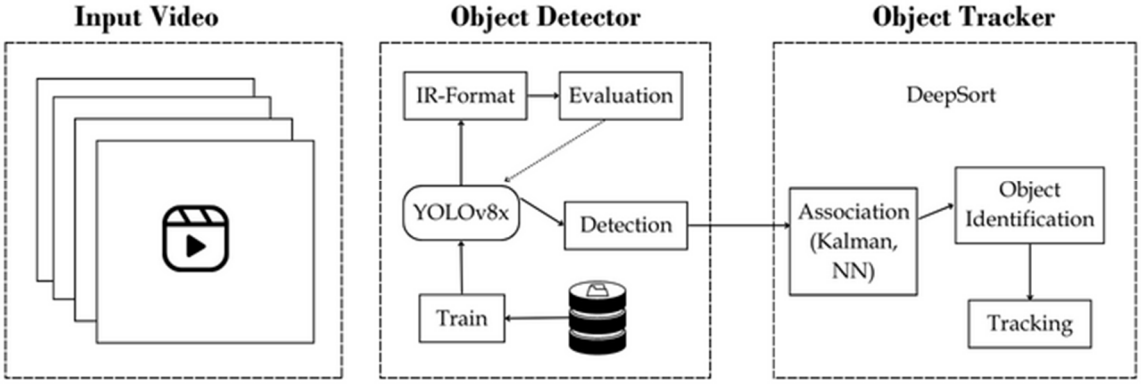


Fig. 4.1 The model architecture.

It is worth noting that the proposed model was trained on a custom dataset that was collected and annotated for autonomous driving. The dataset is derived from the Udacity Self-Driving Car dataset after applying several data cleaning and augmentation algorithms to include more challenging scenes such as occlusions and rapid motion. Importantly, the proposed architecture enables the model to track multiple objects in real-time with high accuracy and can be applied to various practical scenarios.

Phase 1: Object Detection

In the detection phase, we opted to use the latest version of YOLO: the eight version. As illustrated in Fig. **4.2**, YOLOv8 is built on CNNs and introduces a new backbone network, Darknet-53, which is faster and more accurate than earlier versions. Additionally, YOLOv8 uses feature pyramid networks to better recognize objects of different sizes. Interestingly, YOLOv8 also has a user-friendly API that allows users to easily implement the model in their applications.

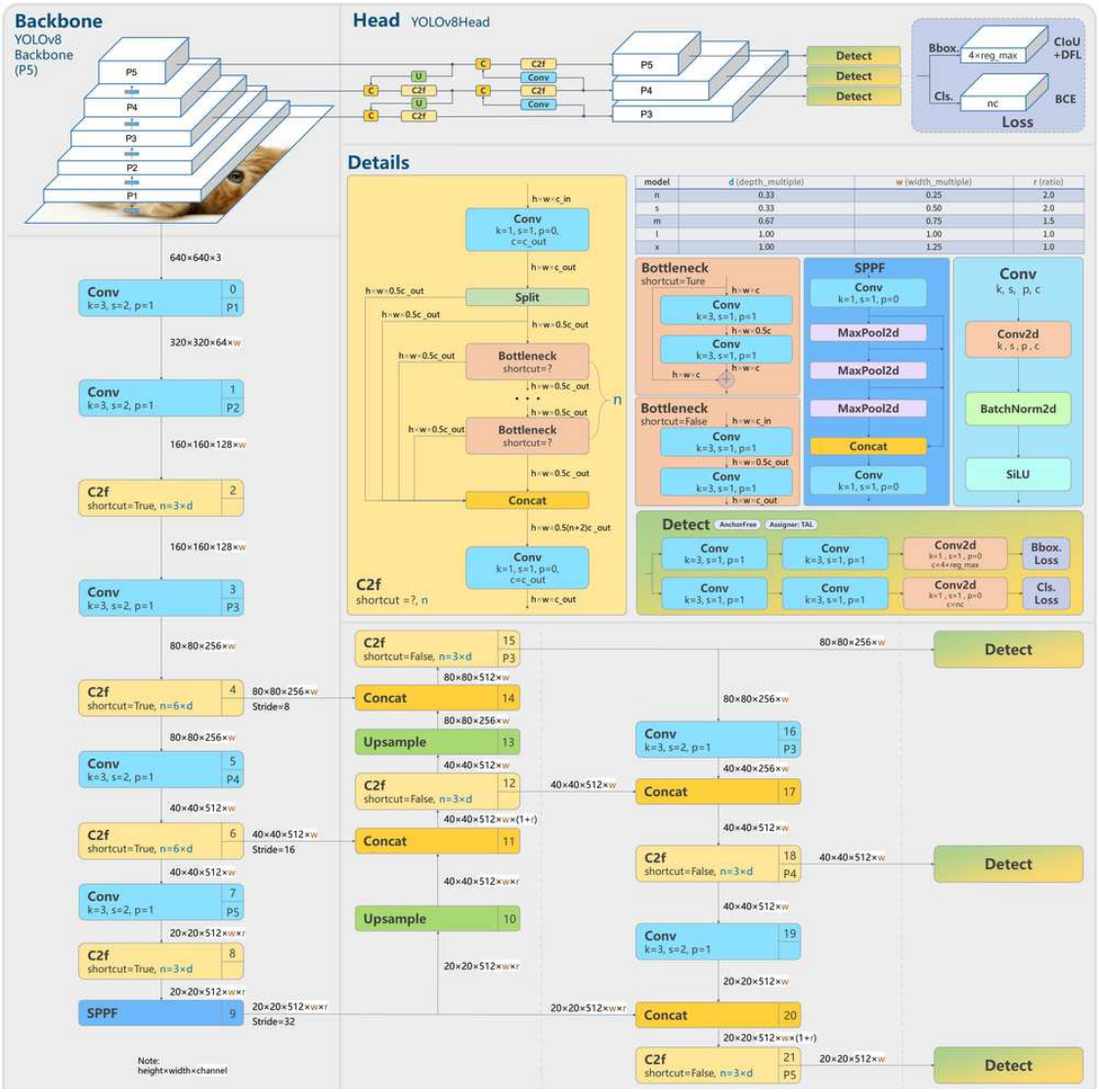


Fig. 4.2 The architecture of the YOLO v8 [73].

Furthermore, as illustrated in Fig. **4.3**, YOLOv8 is an anchor-free model that predicts the center of an object directly, rather than predicting the offset from a predetermined anchor box. This feature reduces the number of box predictions, which speeds up NMS.

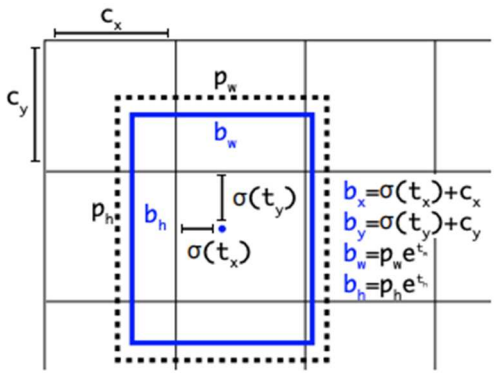


Fig. 4.3 Illustration of the anchor box feature in YOLOv8 [73].

In YOLOv8, the stem's first 6x6 conv is replaced by a 3x3, and the main building block was changed with C2f replacing C3. The module uses new convolutions where features are concatenated directly without forcing the same channel dimensions, which reduces the parameter scount and overall size of the tensors. Fig. **4.4** shows the new C2f module structure:

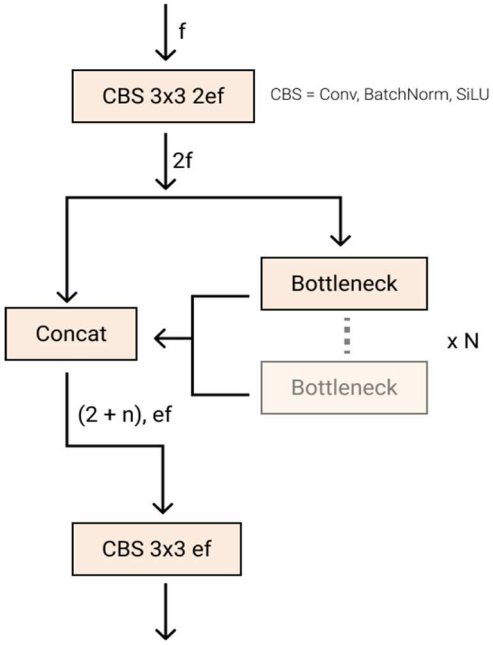


Fig. 4.4 The new YOLOv8 C2f module [73].

YOLOv8 come also with the mosaic augmentation technique, which stitches four images together to force the model to learn objects in new locations, in partial occlusion, and against different surrounding pixels, and is used during online training.

Phase 2: Object Tracking

We have used DeepSort as an object tracker for our model, we get the detected objects from the Yolov8 and then we perform the data association for every project in every single frame and then we give each object an id, which helps us to keep tracking of this object even if the object

will be absent from the scene for a while. Fig. **4.5** shows the object tracking phase with all the steps. After receiving the frames from the detection phase, the Kalman filter is employed to estimate the state of each tracked object. The Kalman filter predicts the future state of the object based on its previous state and motion dynamics by providing us with a probability distribution of the future state. it also provides a mechanism to handle occlusion and missing detections. In our implementation, we use the Mahalanobis distance between the predicted state of the Kalman filter and the detected state as the measurement input to the filter. The Mahalanobis distance is a metric that takes into account the covariance of the measurements and the predicted state. By continuously updating the estimated state of each tracked object using the Mahalanobis distance measurements, the Kalman filter provides a more reliable foundation for the data association and tracking steps that follow.

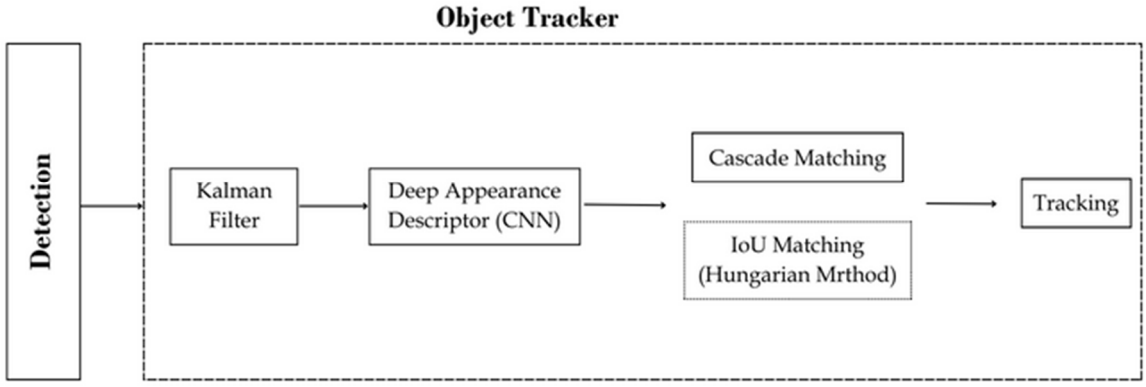


Fig. 4.5 The object tracking workflow.

To associate the detected objects with the existing tracks, the deep appearance descriptor is utilized. The deep appearance descriptor is a feature representation of the object's appearance based on a CNN. The appearance descriptor of each detected object is compared with the appearance descriptors of all existing tracks to find the best match.

To refine the data association results, the cascade matching algorithm is applied. The cascade matching algorithm is a multi-stage data association algorithm that gradually increases the matching criteria and thresholds in each stage. In each stage, the algorithm uses different matching criteria and thresholds to associate the detected objects with the existing tracks. The tracking results are a set of trajectories that represent the motion of the identified objects over time and also a set of bounding boxes with a classification probability.

6.2. Model Training

Dataset

The Udacity Self-Driving Car Dataset is an important resource for developing models for self-driving cars, but the original dataset is missing labels for thousands of pedestrians, bikers, cars, and traffic lights, which can lead to poor model performance and even human fatalities. However, a re-labeled version of the dataset with corrected errors and omissions is available, containing 97,942 labels across 11 classes and 15,000 images. The annotations have been hand-checked for accuracy by Roboflow, and there is also a downsampled version of the images that is suitable for most common machine learning models. While this dataset is valuable for research and development, it's crucial to thoroughly evaluate the performance of models built with it to ensure safety and reliability in real-world scenarios.

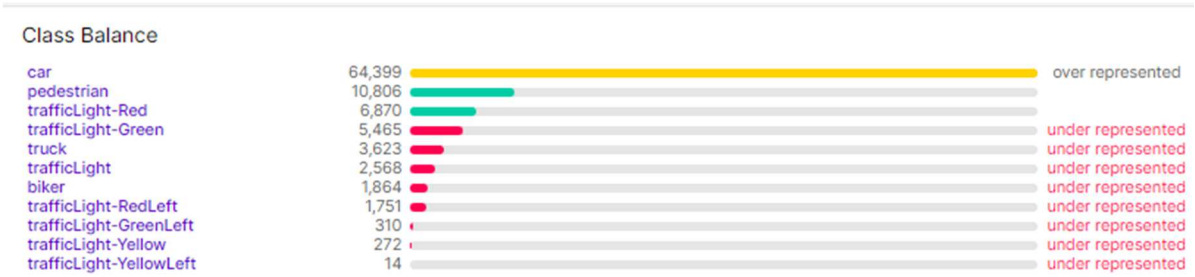


Fig. 4.6 The classes of the Self-Driving Car dataset.

Data pre-processing and balancing

Our dataset underwent pre-processing and balancing to improve the quality and diversity of the data. During pre-processing, we removed duplicate images and added additional images in different positions to increase the variability of the dataset. To address the class imbalance, we performed partial balancing by repeating some images to achieve a logical percentage of the presence of some objects in the dataset. However, we avoided completely balancing the dataset to retain meaningful information about the data, and we also kept the possibility to insert more images with new annotations in the future.

To handle errors that may occur with over or under-represented objects, we will be using weighting techniques. These techniques will help us to keep the balance between the classes, even if there are differences in the number of examples for each class.

Overall, our pre-processing and balancing techniques will be tested and will be checked if they have contributed to a higher quality and more diverse dataset, which can improve the accuracy and generalization performance of our machine learning model.

By using weighting techniques, we can ensure that the model is trained on a balanced dataset, which can prevent it from being biased towards one particular class.

Dataset Preparation

For this project, I split the dataset into three parts: 80% for training, 10% for testing, and 10% for validation. This is a common practice in machine learning to ensure that the model generalizes well to new data. The training set is used to fit the model, the test set is used to evaluate the model's performance on unseen data, and the validation set is used to tune the model's hyperparameters. By using three separate sets, we can ensure that the model is not overfitting to the training data and can perform well on new, unseen data. Overall, this approach can help to improve the accuracy and reliability of the machine learning model.

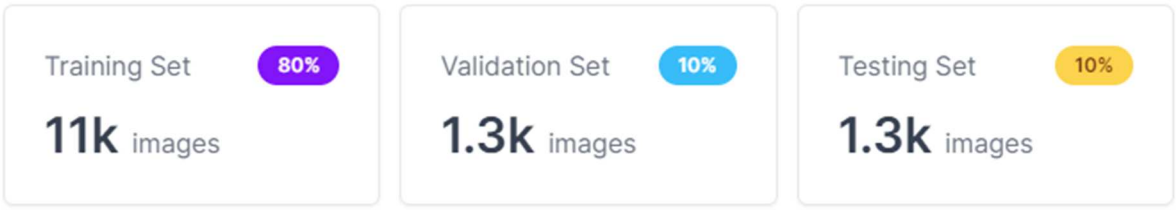


Fig. 4.7 Dataset sets distribution.

Training results

Since we aim to optimize the inference, we are going to present here some training results, this is important to present the behaviors of the model while training, also we will present some graphs related to the evaluation phase, but the accuracy results will be presented in the following chapter.

Fig **4.8** and Fig **4.9** shows the plots of the 3-training metrics

- box_loss: this is the loss function that quantifies the discrepancy between the predicted boxes and the ground-truth bounding boxes. Based on the plot of the box loss during training, I observed that the loss initially started at 1.55 and gradually decreased over the course of training. This indicated that the model was improving its ability to predict accurate bounding box coordinates for the objects of interest in the images.

  Based on the plot of the box loss during training, I observed that the loss initially started at 1.55 and gradually decreased throughout training. This indicated that the model was improving its ability to predict accurate bounding box coordinates for the objects of interest in the images.

- cls_loss: this is associated with the classification task and it is calculated based on the discrepancy between the predicted class labels and the ground truth class labels for the objects in the images. We observed that the loss started at 1.2 and gradually decreased throughout training, reaching a value of 0.7 at the end. This indicates that the model's classification accuracy was improving, as it was better able to correctly predict the class labels of objects in the images. The decreasing trend of cls_loss suggests that the model was learning to make more accurate class predictions, resulting in improved performance over time
- dfl_loss: the distribution focal loss is a variant of the focal loss that takes into account the class distribution in the training data. This loss function is particularly useful when the class distribution is heavily skewed towards one or a few classes. The distribution focal loss aims to give more weight to the minority classes during training, to ensure that the model learns to distinguish between all classes, not just the majority class. The dfl_loss value started at 1.28 at the beginning of training and gradually decreased to 1.18 over the course of training. This indicates that the model is becoming better at distinguishing between all classes, not just the majority class, and is making progress in addressing the issue of class imbalance

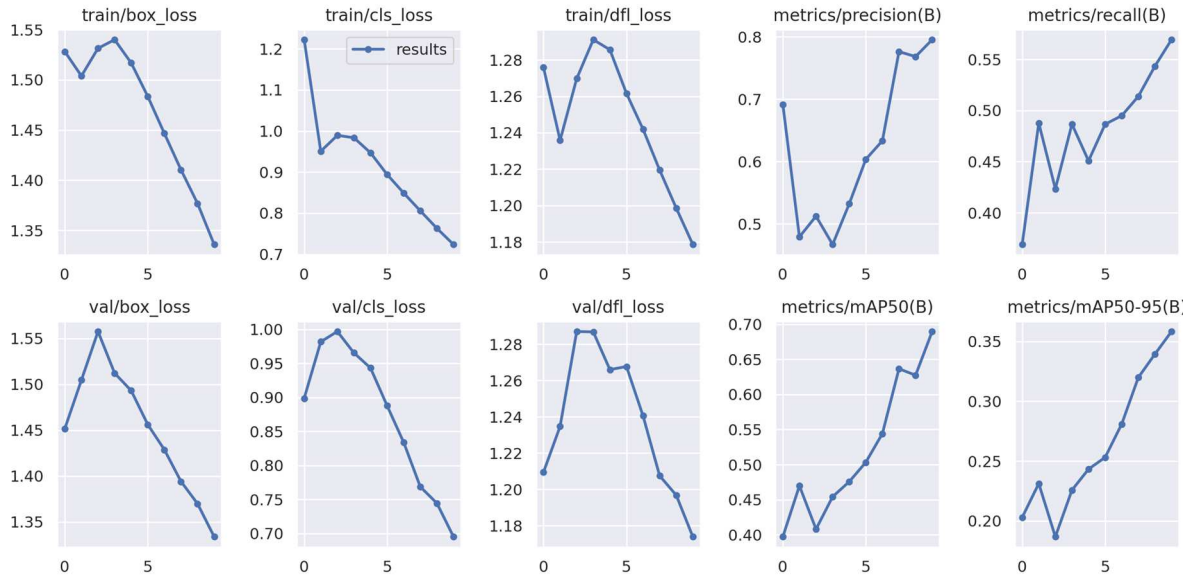


Fig 4.8 The results of the losses from the training and validations for the large model.

We can see that the accuracy results of the nano model are similar to the large model, just we can notice that the large model is performing a little bit better than the nano model.

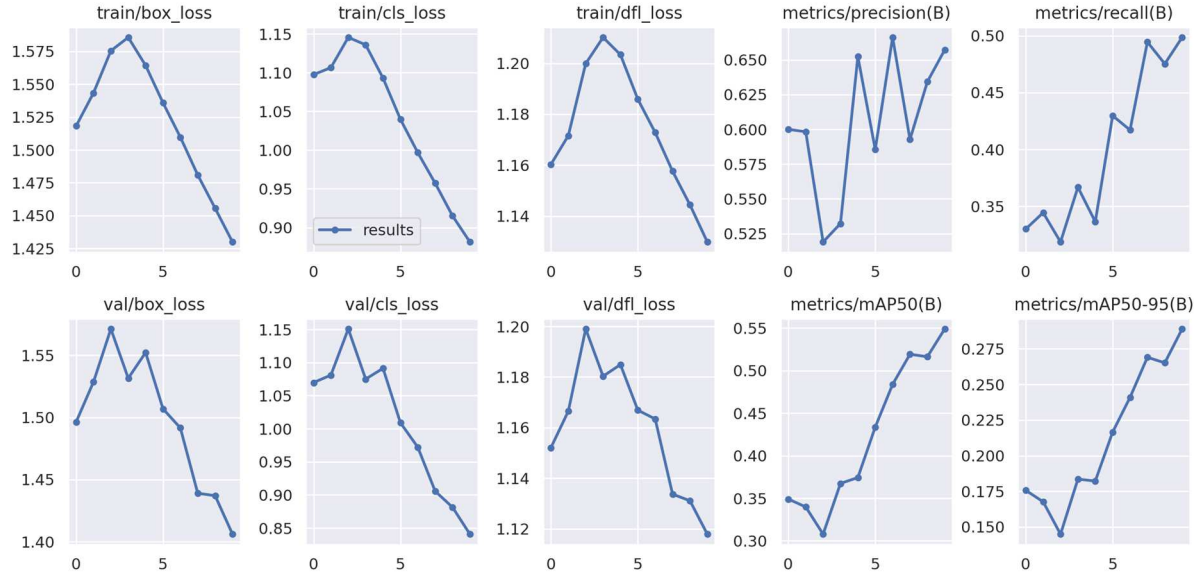


Fig 4.9 The results of the losses from the training and validations for the nano model.

As we can see from the confusion matrix shown in Fig. **4.10**, the models are classifying most of the objects accurately, except the traffic light left and the traffic light yellow. After all, we gave a null weight to the first one and an epsilon one for the second, because we don't have much data about them.

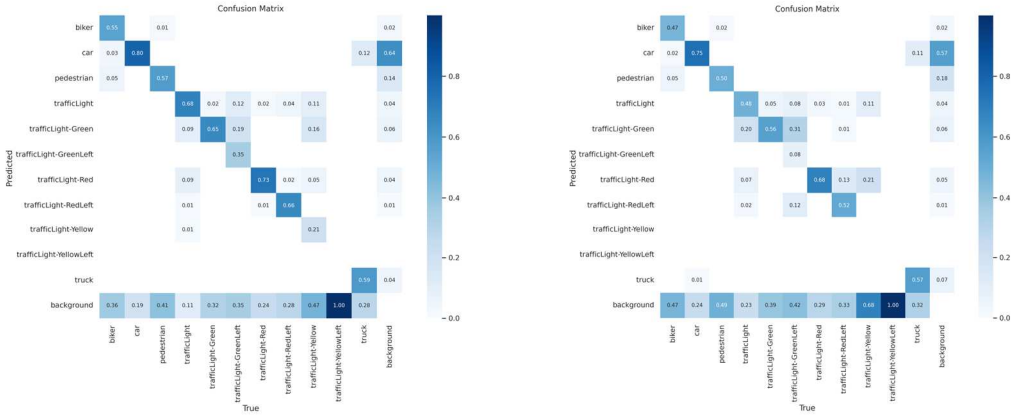


Fig. 4.10 Confusion matrix of large and nano models.

Visual results for our models

The aim of presenting these results is to link them with the training result of object detection and not to compare it with other algorithms.

The following figures Fig. **4.12** show the visual results of our model (nano model). Fig. **4.11** presents the results from the large model.

We can see that the model is performing well in many different situations where challenges are raised. We can see the objects are detected and classified even when they are in correlation

with other objects when the objects are far and not clear and overall, our model is performing well for both versions.

Fig. 4.11 The visual results of the validation for the large model.

Fig. 4.12 The visual results of the validation for the nano model.

The results achieved by our model are highly satisfactory in terms of accuracy. After meticulous testing and validation, our model has demonstrated remarkable performance, meeting and even surpassing our expectations. With such promising outcomes, we are now considering the next step - deploying the model to the edge.

To ensure efficient deployment, we are focused on optimizing the obtained weights from our trained model. This optimization process aims to convert the model's weights into the IR (Intermediate Representation) format, which is a representation that can be used with edge devices such as the Neural Compute Stick (NCS). The IR format is specifically designed for deployment on edge devices, as it is optimized for inference tasks and can deliver faster and more efficient results compared to the original model.

6.3. Model Optimization

After training our model for motion tracking using YOLOv8 and DeepSORT, we obtained a weights file that can be used to make inferences. However, to execute our model on the Intel Movidius Neural Compute Stick (NCS), we need to convert it to a format that the NCS can understand. The NCS is designed to work with deep learning models that have been converted to the Intel Movidius Neural Compute API (NCAPI) format, which is optimized for the NCS hardware.

To convert our obtained weights file to the NCAPI format, we first need to convert it to an intermediate representation (IR) format that is compatible with the NCS. We can use a converter tool such as the OpenVINO Model Optimizer to convert our weights file to the ONNX format, which is a popular choice for the IR format.

Next, using the Model Optimizer tool, we can compile the ONNX model to the NCAPI format, which optimizes the model for the NCS hardware and generates an NCAPI format model that can be used for inference. This process also involves obtaining .xml and .bin files which can be used to run inference on the NCS.

Finally, we can execute our model on the NCS using the compiled NCAPI model and the Neural Compute API (NCAPI) in conjunction with the NCSDK. Although the process of converting a weights file to the NCAPI format requires some technical knowledge, it is possible to achieve high performance and accuracy on the NCS using our trained model. In the next chapter, we will test our model and share the obtained results.

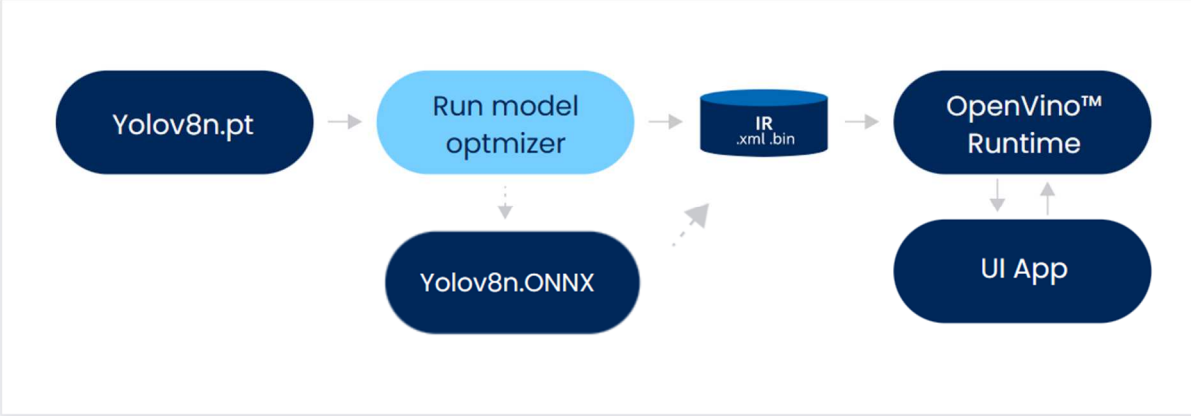


Fig. 4.13 The model optimization step.

CONCLUSION

In this chapter, we have proposed a motion-tracking model using YOLOv8 and DeepSORT. We have presented the training phase and optimization steps, including the

conversion of our model to the Intel Movidius Neural Compute API (NCAPI) format for efficient inference on the Intel Movidius Neural Compute Stick (NCS). Our model combines state-of-the-art object detection and tracking techniques to achieve real-time motion tracking with high accuracy and robustness. After training and optimizing the model, it is now ready for comprehensive testing in real-world scenarios, and the integration of deep learning and edge computing using the NCAPI format and NCS device has the potential to significantly enhance motion tracking systems. In the next chapter, we will report the obtained results, including accuracy, robustness, and inference speed, to further validate the effectiveness of our proposed model and contribute to the field of motion-tracking research.

"Ideas are easy. Implementation is hard"

—Guy Kawasaki

5

Obtained Results

INTRODUCTION

In this section, we provide a detailed description of the hardware and software specifications used in the development and establishment of the system. The hardware specifications included a Raspberry Pi and other devices such as the camera and NCS, as well as the laptop used in the experiments. We also listed the operating systems and GPU used in performing the experiments and provided information on the programming languages and frameworks that were utilized. Additionally, we present the results of our experiments on both traditional and modern algorithms, providing a comparison between the two. These results demonstrate the effectiveness of our proposed system and provide insights into the performance and capabilities of the algorithms used.

ENVIRONMENT DESCRIPTION

7.1. Hardware

7.1.1. *Raspberry pi*

Fig. 5.1 The RPi specifications card.

a. Setup of raspberry pi

In this dissertation, we are going to use many different tools, sensors, and devices to set up the appropriate system for our research:

- RPi v 4.0 B
- SD card with a minimum size of 4 GB to install the OS
- Display monitor (or a VNC tool to connect remotely to the RPi using SSH)
- Mouse and keyboard (if using without VNC)
- Computer and VNC software (if want to connect via SSH)
- PiCam (Necessary to get real-time videos)
- Intel Movidius Neural Compute Stick (to use for training)

After the installation of the Raspberry PI operating system, we can start using the system and installing the necessary packages for our project.

7.1.2. *Laptop*

To perform a part of the experiments, a laptop with the following specifications was used. These specifications are listed in Fig. **5.2**.

Fig. 5.2 The laptop specifications card.

7.1.3. *Google Colab*

For additional computing power, a Colab virtual machine (VM) equipped with a hardware GPU was utilized in the experiments. The specifications of the Hardware used in the Colab VM are listed in Fig. **5.3**.

Fig. 5.3 Google Colab specifications.

7.2. Software

In this part, we will be showcasing the software configuration of a Raspberry Pi device. We will cover the various programming languages and frameworks that we will be using to develop and run our models on the Raspberry Pi and the CPU.

7.2.1. *Operating systems tools*

We set up and configured a Raspberry Pi device running Raspbian OS using a Windows 10 computer. To establish a secure connection to the Raspberry Pi, we utilized Putty, a popular SSH client, and established a remote desktop session using VNC Viewer to access the Raspberry Pi's graphical user interface.

7.2.2. *Programming languages and frameworks*

Here we are going to present the used programming languages, tools, and frameworks to avoid the issue that might face the process of re-doing what we have done.

Table 5.1 List of the most important tools and frameworks used in our experiments.

Tool	Version
Python	3.11.0
OpenCV	4.7.0
OpenVino	2022.3.0
Matplotlib	3.3.0
NumPy	1.21.5
Tensorflow	2.9.1
Conda	3.0.2
Pandas	3.0.3

OBTAINED RESULTS

8.1. Comparison Metrics

Here we are going to present the important metrics that will be used to perform the comparison between the traditional algorithms and their importance:

- **IoU** stands for Intersection over Union which compares the overlap between the predicted bounding box (or segmentation mask) and the ground truth bounding box (or segmentation mask) for a given object.

The mathematical equation for the Intersection over Union (IoU) between two bounding boxes can be represented as IoU = Area of overlap / Area of union where the "Area of overlap" is the area of the region where the two bounding boxes intersect, and the "Area of the union" is the total area covered by both bounding boxes.

- **Mean IoU:** is the mean of the obtained results from the frames of the video.
- **Mean Accuracy:** is calculated based on the distance between the ground truth and the predicted box.

And now we are going to present the metrics used to perform the tests on the YOLOv8n model

- **mAP50**: is a specific version of this metric where the AP is calculated only for those detections where the intersection over union (IoU) between the predicted bounding box and the ground truth bounding box is above 0.5. In other words, a predicted bounding box is considered a true positive only if its IoU with the ground truth bounding box is above 0.5.
- **mAP50-95:** is more comprehensive than the mAP50 metric, as it considers a wider range of IoU thresholds and thus provides a more nuanced evaluation of the detector's performance. However, it is also more computationally expensive to calculate, as it requires evaluating the detector at multiple IoU thresholds.
- **Precision:** is a metric used to evaluate the accuracy of the algorithm in identifying true positive (TP) detections among all detections, i.e., the ratio of true positives to the total number of detections made by the algorithm. Precision is often used in conjunction with recall, which measures the ratio of true positives to the total number of ground truth objects in the image. Together, precision and recall are used to calculate the F1 score, which provides a combined measure of the algorithm's accuracy. A high precision value indicates that the algorithm can correctly identify a large portion of true positive detections while minimizing the number of false positive detections.
- **Recall:** this is a metric used to evaluate the completeness of the algorithm in detecting all the ground truth objects in the image, i.e., the ratio of true positives to the total number of ground truth objects. A recall is often used in conjunction with precision, which measures the ratio of true positives to the total number of detections made by the algorithm. Together, precision and recall are used to calculate the F1 score, which provides a combined measure of the algorithm's accuracy. A high recall value indicates that the algorithm can detect a large portion of the ground truth objects in the image while minimizing the number of false negatives, i.e., objects that were not detected by the algorithm but were present in the image.

The common performance metric between them is FPS.

- **FPS (frames per second)** is an important metric in motion tracking, as it measures the number of video frames that are processed by the motion tracking algorithm per second. A high FPS value indicates that the motion tracking algorithm can process and analyze a large number of frames quickly, which can result in a smooth and responsive tracking performance.

8.2. Traditional Algorithms

In this part, we are going to compare 2 traditional algorithms, one from model-based methods and one from feature-based methods, in which we will evaluate the accuracy based on the Euclidean distance and IoU metrics and we will see how they perform in some particular cases such as the similarity of objects and also the unstable environment (Noisy) and also, we will check the performance using FPS metric.

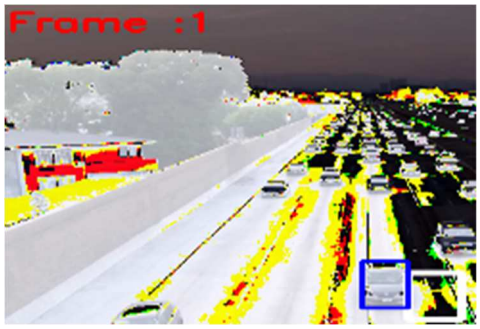
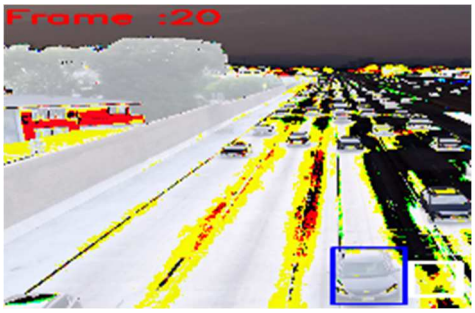
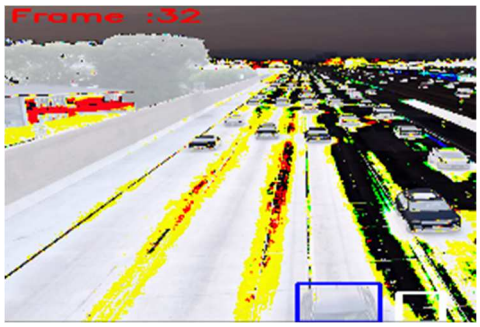
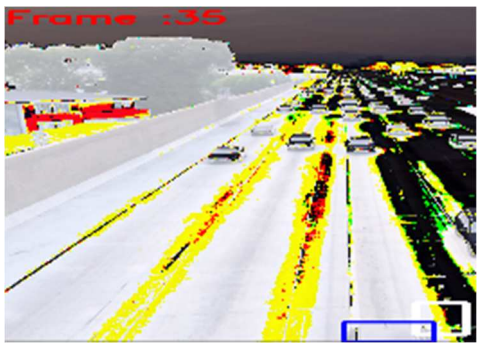
We've implemented the basic version of the Mean Shift Algorithm, and the Lukas Kanade optical flow algorithm to evaluate their performance we have used a video sequence that contains similar objects (cars) moving and we have already the tracked frames defined and we will just compare it with the manual selection of a car withing first 35.

As we can see in Table 5.2 which represents the visual results of the ground truth is the one with the blue rectangle for both algorithms and the predicted box is the white one in MS and the black one in LKOF.

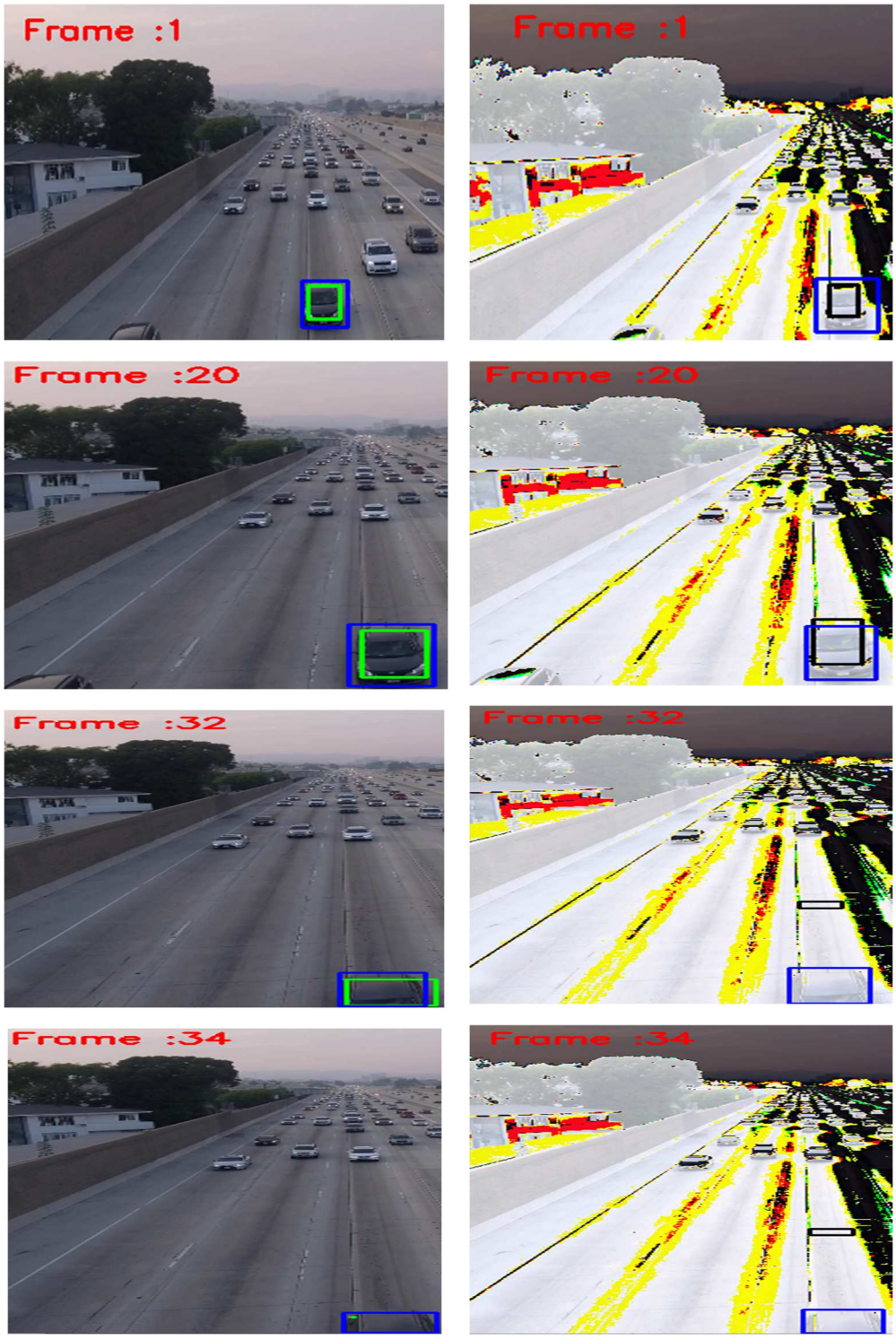
The first row of the table represents the visual results of the mean shift with normal and noisy frames, I selected some frames from 1 to 35 in which the target object is present.

The second row of the table represents the visual results of the Lukas Kanade Optical Flow with normal and noisy frames, I selected some frames from 1 to 34 in which the target object is present.

Table 5.2 Obtained frames with detection.

Detector	Normal frames	Noisy frames
Mean Shift		
		
		
		

Lukas-Kande Optical Flow



Now, we will do the interpretation of the results presented in Table 5.3. We can see that the LKOF algorithm is giving better results, especially when we are facing noise.

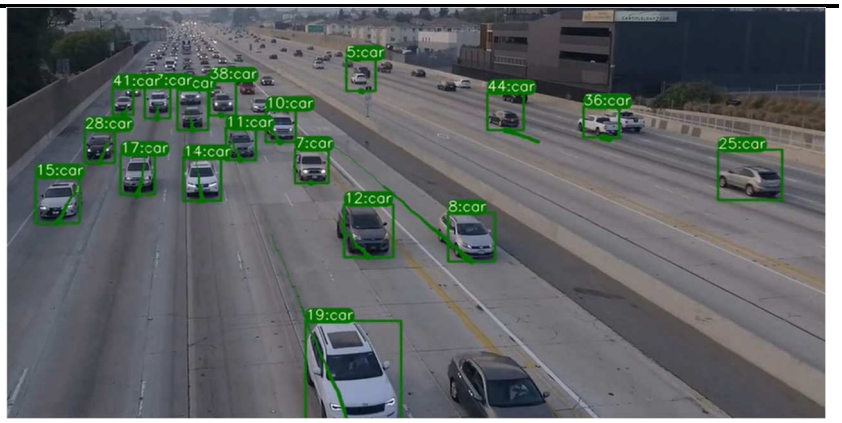
Table 5.3 Obtained results for trackers using Colab.

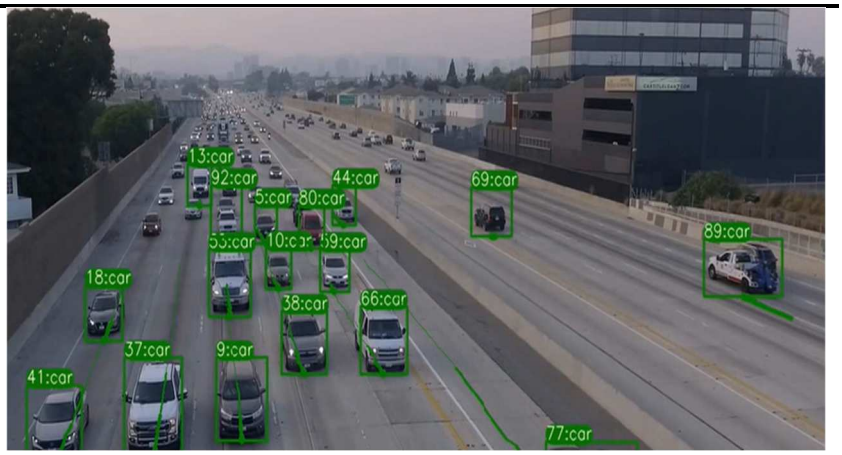
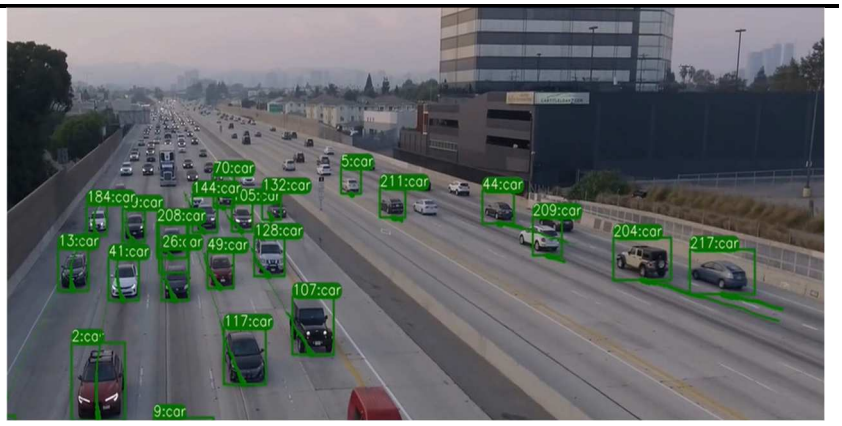
Detector	Mean IoU (normal video)	Mean IoU (noisy video)	Mean Accuracy (Normal)	Mean Accuracy (Noisy)	FPS
Mean Shift	0.30	0.04	0.98	0.91	130
Lukas-Kanade	0.35	0.26	0.99	0.96	150
Kalman-Filter	0.37	0.27	0.99	0.98	157

According to the obtained results, we can see that IoU and accuracy in the normal case for all algorithms are similar but in the noisy case, we can see a big difference in terms of IoU but a small difference in terms of accuracy because here the accuracy is just the Euclidian distance between the canter of the ground truth and the predicted box and even small distances are considered to have a very big impact on the accuracy.

8.3. Deep Learning-Based Solutions

The following shows the obtained tracking results using our tracking model using YOLOv8 nano and DeepSort.

Frame	Results
1	

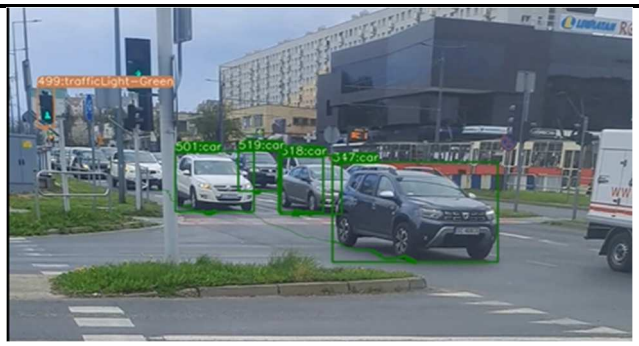
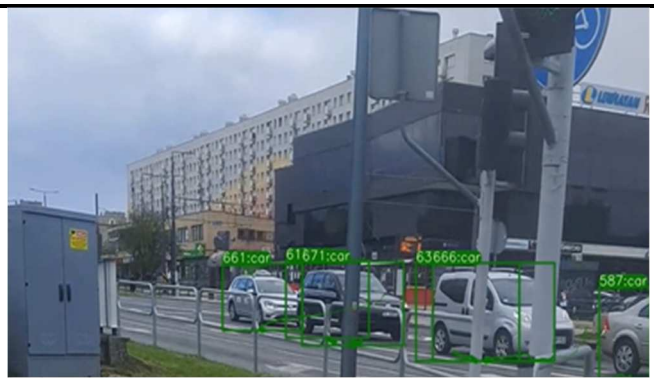
60	
200	

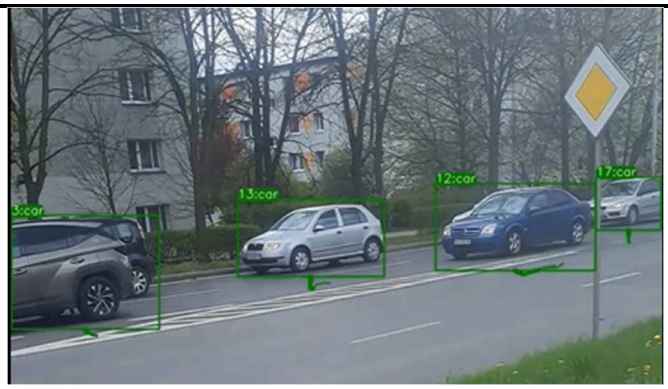
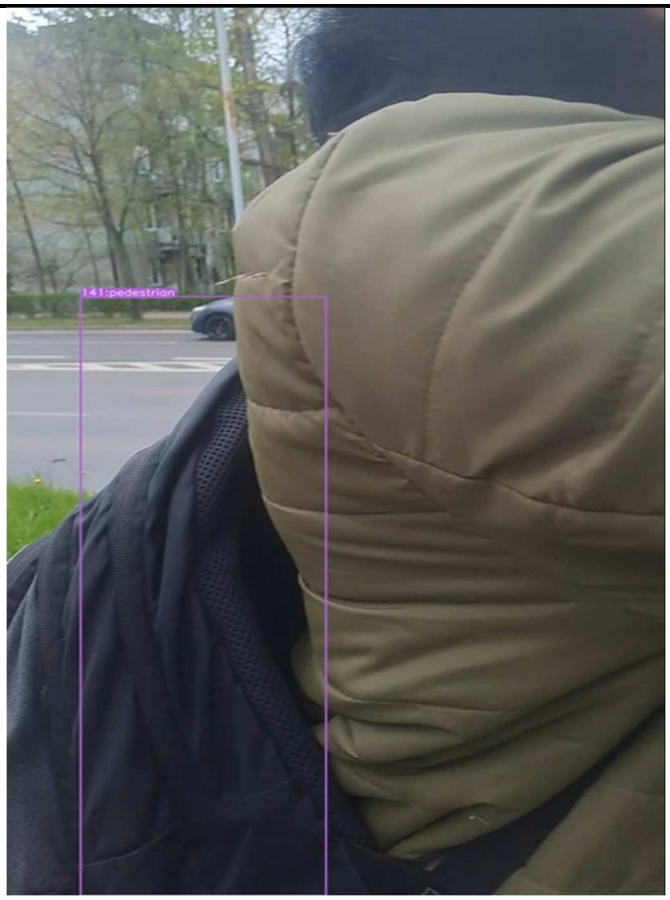
The following shows the results that our model is identifying the object and does not lose it over time.

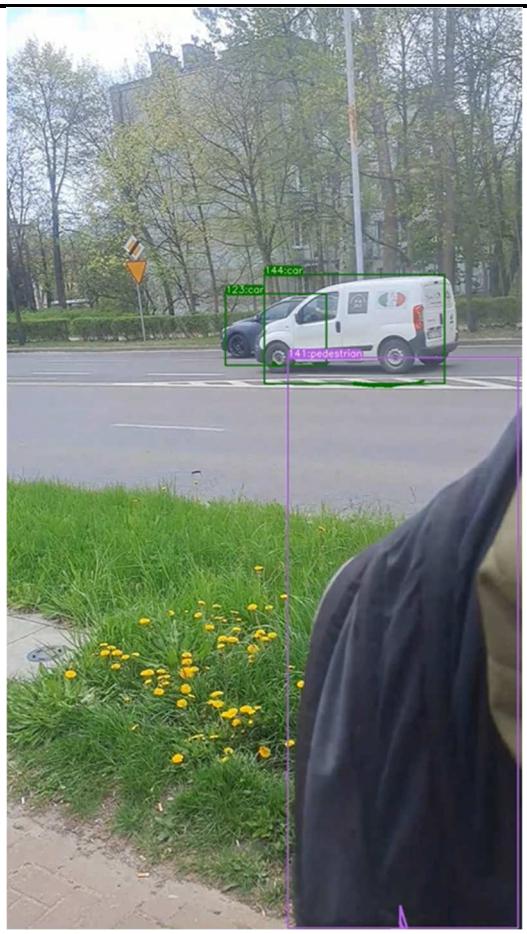
Frame	Results
1	
40	

100	

After using the available online testing videos, now it is the time to move to test our model on a real-life problem. We choose to test it on the so-called Hala Polonia Street near to our university and the following table shows the obtained results:

Frame	Observation	Results
5	In this frame we can see that our model is detecting cars and truck and assigning ids to them, the only challenge here is the ability to detect multiple objects.	
20	One challenge here is the occulsion of cars, as we can see our model , is performing good in distinguishing between cars and able to detect the traffic light which has a very small region within the image.	
68	The challenge here is the ability to detect and track partially hidden cars, as we can see our model is able to detect cars even if they are not fully clear in the image	

120	Here we tried to check the tracking from the other side.	
150	In this frame we can see a real example of the challenge: - Where the object is not fully present in the frame we can see that our model us detecting it.	

157	In this frame we can see a real example of the challenge: - Occulsion	

After completing the evaluation of the visual results, it has been determined that the accuracy of the large and nano models is quite similar. However, to obtain a more comprehensive assessment of the algorithms, it is crucial to analyze their performance based on a set of well-defined metrics. These metrics provide a quantitative analysis that enables a logical and concise comparison between the two models. To this end, we have generated two tables, namely Tables 0.9 and 1.0, which represent the performance of the large and nano models, respectively. These tables are an essential tool for the evaluation process as they allow us to compare the results of the models across different dimensions, including precision, recall, mAP, and FPS.

The metrics reported in these tables provide an accurate reflection of the performance of the algorithms, and we can use them to draw meaningful conclusions regarding their effectiveness. By analyzing these metrics, we can identify areas where the models excel, as well as areas where they may require improvement.

Table 5.4 shows that the large model is having the same results in terms of accuracy in all the environments but its performance is very low on the RPi, we can see that it is improved using the NCS but not enough for real-time applications.

Table 5.4

Obtained results for YOLOv8l with optimization and without**.**

Tracker	mAP50	mAP50-95	Precision	Recall	FPS
Colab GPU	0.69	0.36	0.8	0.57	35
RPi	0.62	0.34	0.8	0.57	<1
RPi + NCS	0.63	0.34	0.8	0.56	4

Table 5.5 shows that the nano model also is having a little bit lesser results in terms of accuracy in all the environments, but its performance is very low on the RPi, but we can see that it is improved using the NCS but still not enough for real-time applications. However, for non-real-time applications is very good and acceptable.

Table 5.5

Obtained results for YOLOv8n with optimization and without.

Tracker	mAP50	mAP50-95	Precision	Recall	FPS
Colab GPU	0.55	0.28	0.67	0.5	42
RPi	0.53	0.28	0.7	0.46	2
RPi + NCS	0.55	0.27	0.71	0.48	12

CONCLUSION

In this chapter, we focused on the technical part that we will use to develop the model and do the object tracking/detection experiments to compare the existing solution with our model, by describing our environment including the used devices and their characteristics, as well as giving a glimpse into some of the software, programming languages, and frameworks used, to give the results that we will get more accurate.

Based on the results that we obtained, we can assert that traditional methods and modern methods of motion tracking can be combined to obtain optimal results (hybridization), as we see that traditional methods have several advantages in terms of single object tracking, but as for MOT, modern methods are better.

"I imagine a world in which AI is going to make us work more productively, live longer, and have cleaner energy."

—Fei-Fei Li, a professor of Computer Science at Stanford University

6

Conclusions And Perspectives

In this study, we set out to design and develop a motion-tracking system using Raspberry Pi. The thesis aimed to create a low-cost and accessible solution for motion tracking that could be used in a variety of applications. Through the course of the study, I was able to achieve my objective and create a fully functioning motion-tracking system that demonstrates the potential of Raspberry Pi as a platform for motion tracking. In this chapter, I will summarize the key findings of the study and discuss their implications for the field of motion tracking.

CONCLUSIONS

The key findings of this study demonstrate the benefits of hybridizing traditional motion-tracking systems with deep learning-based methods. By combining these two approaches, we were able to achieve higher levels of accuracy and reliability in motion tracking compared to using either approach alone. This highlights the potential of integrating different techniques and technologies in motion tracking to achieve better results. However, the comparison between traditional and modern methods of motion tracking can be challenging, as they have different underlying concepts and metrics for evaluation. Despite this, our study was able to demonstrate clear differences in performance between the two approaches, another important finding of this study is the potential of the Neural Compute Stick (NCS) as an optimizer for motion tracking performance. Our experiments showed that using the NCS for certain processing tasks can significantly improve the speed and efficiency of the motion-tracking system. This suggests that the NCS could be a valuable tool for optimizing performance in motion-tracking applications where speed is a critical factor.

POTENTIAL LIMITS

There are a few potential limitations to this project that should be considered. First, the chosen object-tracking algorithms may face challenges when tracking objects that are moving at

high speeds, particularly in the context of real-time object tracking. However, this limitation can be addressed by expanding the model architecture to include additional layers and integrating hardware accelerators to improve tracking performance.

It's important to note that the dataset used in this project is primarily focused on autonomous driving cars and may not be as comprehensive in other domains. While the dataset is quite large and contains a significant amount of data, it's important to recognize that it may not be representative of all possible use cases. Additionally, the dataset may not contain enough data for some objects, particularly if they are rare or not typically encountered in autonomous driving scenarios. Despite these limitations, we believe that the dataset is well-suited for this project and should provide a strong foundation for training and evaluating the object-tracking algorithms.

FUTURE WORK

In the future, I plan to expand upon my previous work on the motion tracking system using YOLOv8n and deepSORT by increasing the size of my dataset and generalizing it to make the system more accurate. By doing so, I hope to improve the accuracy and reliability of my system. In addition, I plan to utilize multiple NCS optimizers and NVIDIA accelerators to further enhance the performance of my system. To improve the usability of the system, I aim to develop a user-friendly interface that allows users to interact with the system and visualize the tracking results in real time. Furthermore, I plan to explore hardware acceleration options such as FPGAs or custom ASICs to accelerate the computation of the system. Since our scope is to make the models executable in an Edge environment, developing efficient and optimized models is crucial. By incorporating these advanced tools and techniques into my workflow, I am confident that I can develop more robust and efficient motion-tracking systems that can deliver real-world impact and value.

Overall, I am excited about the potential of my motion-tracking system and the many different ways in which it can be improved and expanded in the future. By incorporating cutting-edge deep learning techniques and advanced hardware solutions, I am confident that I can develop a truly world-class system that can deliver real-world impact and value.

RECOMMENDATIONS AND PERSPECTIVES

Certainly! Based on my experience developing and working with motion tracking systems, I have a few recommendations and perspectives that I believe could be valuable for readers who are interested in this area of research:

Invest in high-quality data: As I mentioned earlier, having a large and diverse dataset is critical for developing accurate and robust motion-tracking systems. Investing time and effort in

data collection, annotation, and preprocessing can pay off in the form of better model performance and more reliable tracking results.

Don't neglect hardware optimization: While deep learning algorithms are the backbone of motion-tracking systems, the hardware on which they run is equally important. Optimizing your system for different types of hardware can help improve performance and flexibility and can enable you to deploy your system in a wider range of scenarios.

Consider post-processing and analysis techniques: While tracking results is important, they are only the beginning of the story. Developing innovative post-processing and analysis techniques can help unlock new insights and applications for motion-tracking systems and can help you derive even more value from your data.

Collaborate and share your work: Finally, I believe that collaboration and knowledge-sharing are critical for advancing the state of the art in motion tracking and other areas of deep learning research. By sharing your code, data, and insights with others, you can help accelerate progress and foster a more vibrant and productive research community.

References

[1] "What Is Object Detection?," *MATLAB and Simulink*, 2023. https://www.mathworks.com/discovery/object-detection.html (accessed Jan. 06, 2023).

[2] G. Boesch, "Object Detection in 2023: The Definitive Guide," *viso.ai*, 2023. https://viso.ai/deep-learning/object-detection/ (accessed Jan. 12, 2023).

[3] V. Lendave, "Guide To Simple Object Detection Using InceptionResnet_v2," *Analytics India Magazine*, Jul. 13, 2021. https://analyticsindiamag.com/guide-to-simple-object-detection-using-inceptionresnet_v2/ (accessed Jan. 26, 2023).

[4] M. Saeed, "Object Detection in Computer Vision: A Guide," *AI Exchange*, Jan. 18, 2022. https://exchange.scale.com/public/blogs/object-detection-in-computer-vision-a-guide (accessed Jan. 12, 2023).

[5] F. Chen, X. Wang, Y. Zhao, S. Lv, and X. Niu, "Visual object tracking: A survey," *Computer Vision and Image Understanding*, vol. 222, p. 103508, Sep. 2022, doi: 10.1016/J.CVIU.2022.103508.

[6] R. Potter, "What is the Main Purpose of Video Annotation in Machine Learning and AI?," *Becoming Human: Artificial Intelligence Magazine, Medium*, Feb. 15, 2020. https://becominghuman.ai/what-is-the-main-purpose-of-video-annotation-in-machine-learning-and-ai-11805710bd95 (accessed Jan. 04, 2023).

[7] M. Bashar, S. Islam, K. K. Hussain, Md. B. Hasan, A. B. M. A. Rahman, and Md. H. Kabir, "Multiple Object Tracking in Recent Times: A Literature Review," Sep. 2022, Accessed: Apr. 13, 2023. [Online]. Available: https://arxiv.org/abs/2209.04796v1

[8] N. Klingler, "Object Tracking in Computer Vision (Complete Guide)," *viso.ai*, 2023. https://viso.ai/deep-learning/object-tracking/ (accessed Apr. 14, 2023).

[9] X. Li, W. Hu, C. Shen, Z. Zhang, A. Dick, and A. Van Den Hengel, "A Survey of Appearance Models in Visual Object Tracking," *ACM Trans Intell Syst Technol*, vol. 4, no. 4, Mar. 2013, doi: 10.1145/2508037.2508039.

[10] R. Szeliski, *Computer Vision: Algorithms and Applications*. in Texts in Computer Science. Cham: Springer International Publishing, 2022. doi: 10.1007/978-3-030-34372-9.

[11] A. D. Worrall, R. F. Marslin, G. D. Sullivan, and K. D. Baker, "Model-based Tracking," in *British Machine Vision Conference*, P. Mowforth, Ed., London, UK: Springer-Verlag, 1991, pp. 310–318. doi: 10.1007/978-1-4471-1921-0_39.

[12] L. Jiao *et al.*, "A survey of deep learning-based object detection," *IEEE Access*, vol. 7, pp. 128837–128868, 2019, doi: 10.1109/ACCESS.2019.2939201.

[13] "What is Artificial Intelligence (AI)?," *IBM Cloud Education*, 2020. https://www.ibm.com/cloud/learn/what-is-artificial-intelligence (accessed May 07, 2021).

[14] S. Russell and P. Norvig, *Artificial intelligence: A modern approach*, Fourth. in Pearson Series in Artificial Intelligence. Pearson, 2020.

[15] J. Sen *et al.*, *Machine Learning: Algorithms, Models and Applications*. London, UK: IntechOpen, 2021. doi: 10.5772/intechopen.94615.

[16] R. V. Zicari, Ed., *Explorations in Artificial Intelligence and Machine Learning*. CRC Press. Accessed: Jan. 29, 2023. [Online]. Available: https://www.routledge.com/rsc/downloads/AI_FreeBook.pdf

[17] H. Cenan, "What's the deal with AI, anyway?," *ZASTI, Medium*, Jul. 12, 2019. https://medium.com/zasti/whats-the-deal-with-ai-anyway-56a30177f438 (accessed Jan. 02, 2023).

[18] "What is Machine Learning?," *IBM Cloud Education*, 2022. https://www.ibm.com/topics/machine-learning (accessed Apr. 13, 2023).

[19] A. Zhang, Z. C. Lipton, M. Li, and A. J. Smola, *Dive Into Deep Learning*, no. 5. 2022. doi: 10.48550/arxiv.2106.11342.

[20] L. Deng and D. Yu, *Deep Learning: Methods and Applications*, vol. 7, no. 3–4. Now Foundations and Trends, 2014. doi: 10.1561/2000000039.

[21] J. Wu, "Introduction to Convolutional Neural Networks," *National Key Lab for Novel Software Technology, Nanjing University*. Nanjing, China, May 01, 2017. Accessed: Jan. 28, 2023. [Online]. Available: https://cs.nju.edu.cn/wujx/paper/CNN.pdf

[22] M. Milicevic, K. Zubrinic, I. Obradovic, and T. Sjekavica, "Application of transfer learning for fine-grained vessel classification using a limited dataset," *Lecture Notes in Electrical Engineering*, vol. 574, pp. 125–131, Jun. 2019, doi: 10.1007/978-3-030-21507-1_19.

[23] K. Patel, "Convolutional Neural Networks — A Beginner's Guide," *Towards Data Science, Medium*, Sep. 08, 2019. https://towardsdatascience.com/convolution-neural-networks-a-beginners-guide-implementing-a-mnist-hand-written-digit-8aa60330d022 (accessed Jan. 27, 2023).

[24] "Max-pooling / Pooling," *Computer Science Wiki*, Feb. 27, 2018. https://computersciencewiki.org/images/8/8a/MaxpoolSample2.png (accessed Jan. 14, 2023).

[25] A. Challa, "What is a ReLU layer?," *educative*, 2023. https://www.educative.io/answers/what-is-a-relu-layer (accessed Jan. 18, 2023).

[26] V. Praharsha, "ReLU (Rectified Linear Unit) Activation Function," *OpenGenus IQ*, 2023. https://iq.opengenus.org/relu-activation/ (accessed Jan. 27, 2023).

[27] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet classification with deep convolutional neural networks," *Commun ACM*, vol. 60, no. 6, pp. 84–90, May 2017, doi: 10.1145/3065386.

[28] B. Ramsundar and R. B. Zadeh, *TensorFlow for Deep Learning: From Linear Regression to Reinforcement Learning*. O'Reilly Media, Inc., 2018. Accessed: Jan. 31, 2023. [Online]. Available: https://books.google.com/books/about/TensorFlow_for_Deep_Learning.html?id=GZ1ODwAAQBAJ

[29] R. Girshick, J. Donahue, T. Darrell, J. Malik, U. C. Berkeley, and J. Malik, "Rich Feature Hierarchies for Accurate Object Detection and Semantic Segmentation," in *Proceedings of*

the IEEE Computer Society Conference on Computer Vision and Pattern Recognition, IEEE Computer Society, Sep. 2014, pp. 580–587. doi: 10.1109/CVPR.2014.81.

[30] R. Girshick, "Fast R-CNN," in *2015 IEEE International Conference on Computer Vision (ICCV)*, Santiago, Chile: Institute of Electrical and Electronics Engineers Inc., Dec. 2015, pp. 1440–1448. doi: 10.1109/ICCV.2015.169.

[31] "OpenVINO: Open Visual Inference and Neural network Optimization," *Intel Corporation*, 2023. https://software.intel.com/en-us/openvino-toolkit. (accessed Jan. 08, 2023).

[32] "Overview of Intel® Distribution of OpenVINO™ Toolkit," *Intel Corporation*, 2023. https://www.intel.com/content/www/us/en/developer/tools/devcloud/edge/learn/openvino.html (accessed Jan. 28, 2023).

[33] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, Las Vegas, NV, USA: IEEE Computer Society, Jun. 2016, pp. 770–778. doi: 10.1109/CVPR.2016.90.

[34] A. Karpathy and L. Fei-Fei, "Deep Visual-Semantic Alignments for Generating Image Descriptions," *IEEE Trans Pattern Anal Mach Intell*, vol. 39, no. 4, pp. 664–676, Apr. 2017, doi: 10.1109/TPAMI.2016.2598339.

[35] F. Okeke, "Benefits of edge computing," *TechRepublic*, Sep. 06, 2022. https://www.techrepublic.com/article/edge-computing-benefits/ (accessed Jan. 28, 2023).

[36] R. Lowman, "How AI In Edge Computing Drives 5G And The IoT," *Semiconductor Engineering, Inc.*, Feb. 13, 2020. https://semiengineering.com/how-ai-in-edge-computing-drives-5g-and-the-iot/ (accessed Jan. 28, 2023).

[37] N. Heath, "What is the Raspberry Pi 4? Everything you need to know about the tiny, low-cost computer," *ZDNET*, Jul. 02, 2019. https://www.zdnet.com/article/what-is-the-raspberry-pi-4-everything-you-need-to-know-about-the-tiny-low-cost-computer/ (accessed Jan. 28, 2023).

[38] "Raspberry Pi 4," *Edge Impulse Documentation*, Jan. 03, 2023. https://docs.edgeimpulse.com/docs/development-platforms/officially-supported-cpu-gpu-targets/raspberry-pi-4 (accessed Jan. 08, 2023).

[39] K. Wiggers, "Intel's Neural Compute Stick 2 is 8 times faster than its predecessor," *VentureBeat*, Nov. 14, 2018. https://venturebeat.com/business/intels-neural-compute-stick-2-is-8-times-faster-than-its-predecessor/ (accessed Jan. 28, 2023).

[40] "Neural Compute Stick 2," *Mouser Electronics*. Feb. 09, 2021. Accessed: Jan. 31, 2023. [Online]. Available: https://www.mouser.in/new/intel/intel-neural-compute-stick-2/

[41] "Intel® Movidius™ Neural Compute SDK Documentation," *Intel Corporation*, 2019. https://movidius.github.io/ncsdk/ (accessed Jan. 28, 2023).

[42] D. Comaniciu and P. Meer, "Mean shift: a robust approach toward feature space analysis," *IEEE Trans Pattern Anal Mach Intell*, vol. 24, no. 5, pp. 603–619, May 2002, doi: 10.1109/34.1000236.

[43] I. S. Topkaya and H. Erdogan, "Using spatial overlap ratio of independent classifiers for likelihood map fusion in mean-shift tracking," *Signal Image Video Process*, vol. 13, no. 1, pp. 61–67, Feb. 2019, doi: 10.1007/S11760-018-1328-3/METRICS.

[44] D. Comaniciu, V. Ramesh, and P. Meer, “Kernel-based object tracking,” *IEEE Trans Pattern Anal Mach Intell*, vol. 25, no. 5, pp. 564–577, May 2003, doi: 10.1109/TPAMI.2003.1195991.

[45] J. Bouguet, “Pyramidal implementation of the Lucas Kanade feature tracker,” *Intel corporation*, vol. 5, no. 1–10, p. 4, 2001.

[46] A. Radgui, C. Demonceaux, E. M. Mouaddib, D. Aboutajdine, and M. Rziza, “An adapted Lucas-Kanade’s method for optical flow estimation in catadioptric images,” in *The 8th Workshop on Omnidirectional Vision, Camera Networks and Non-classical Cameras-OMNIVIS*, Marseille, France, Oct. 2008. Accessed: Apr. 10, 2023. [Online]. Available: https://hal.inria.fr/inria-00325390/

[47] P. Weinzaepfel, Z. Harchaoui, and C. Schmid A A Inria, “Learning to Track for Spatio-Temporal Action Localization,” in *IEEE International Conference on Computer Vision (ICCV)*, Santiago, Chile: IEEE, Dec. 2015, pp. 3164–3172. doi: 10.1109/ICCV.2015.362.

[48] S. Särkkä, “Bayesian Filtering and Smoothing,” *Bayesian Filtering and Smoothing*, pp. 1–232, Jan. 2013, doi: 10.1017/CBO9781139344203.

[49] D. Tomer, “What I Was Missing While Using The Kalman Filter For Object Tracking,” *Towards Data Science*, Jun. 14, 2022. https://towardsdatascience.com/what-i-was-missing-while-using-the-kalman-filter-for-object-tracking-8e4c29f6b795 (accessed Apr. 10, 2023).

[50] I. Arasaratnam and S. Haykin, “Cubature kalman filters,” *IEEE Trans Automat Contr*, vol. 54, no. 6, pp. 1254–1269, Jun. 2009, doi: 10.1109/TAC.2009.2019800.

[51] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, “You only look once: Unified, real-time object detection,” in *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, Las Vegas, NV, USA: IEEE Computer Society, Jun. 2016, pp. 779–788. doi: 10.1109/CVPR.2016.91.

[52] J. Redmon and A. Farhadi, “YOLOv3: An Incremental Improvement,” *Computer vision and pattern recognition*, vol. 1804, pp. 1–6, Apr. 2018, Accessed: Apr. 10, 2023. [Online]. Available: https://arxiv.org/abs/1804.02767v1

[53] G. Jocher *et al.*, “ultralytics/yolov5: v7.0 - YOLOv5 SOTA Realtime Instance Segmentation,” Nov. 2022, doi: 10.5281/ZENODO.7347926.

[54] “YOLOv8: The State-of-the-Art YOLO Model,” *Ultralytics*, 2023. https://ultralytics.com/yolov8 (accessed Apr. 10, 2023).

[55] W. Liu *et al.*, “SSD: Single shot multibox detector,” in *European Conference on Computer Vision (ECCV)*, Amsterdam, The Netherlands: Springer Verlag, Oct. 2016, pp. 21–37. doi: 10.1007/978-3-319-46448-0_2/FIGURES/5.

[56] R. Khandelwal, “SSD : Single Shot Detector for object detection using MultiBox,” *Towards Data Science, Medium*, Nov. 30, 2019. https://towardsdatascience.com/ssd-single-shot-detector-for-object-detection-using-multibox-1818603644ca (accessed Jan. 31, 2023).

[57] J. Huang *et al.*, “Speed/accuracy trade-offs for modern convolutional object detectors,” in *30th IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Institute of Electrical and Electronics Engineers Inc., Nov. 2017, pp. 3296–3305. doi: 10.1109/CVPR.2017.351.

[58] A. Bewley, Z. Ge, L. Ott, F. Ramos, and B. Upcroft, “Simple online and realtime tracking,” in *International Conference on Image Processing, ICIP*, Phoenix, AZ, USA: IEEE Computer Society, Aug. 2016, pp. 3464–3468. doi: 10.1109/ICIP.2016.7533003.

[59] H. Nam and B. Han, “Learning Multi-domain Convolutional Neural Networks for Visual Tracking,” in *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, Las Vegas, NV, US: IEEE Computer Society, Dec. 2016, pp. 4293–4302. doi: 10.1109/CVPR.2016.465.

[60] Y. Zhang, C. Wang, X. Wang, W. Zeng, and W. Liu, “FairMOT: On the Fairness of Detection and Re-identification in Multiple Object Tracking,” *Int J Comput Vis*, vol. 129, no. 11, pp. 3069–3087, Nov. 2021, doi: 10.1007/S11263-021-01513-4/METRICS.

[61] N. Wojke, A. Bewley, and D. Paulus, “Simple online and realtime tracking with a deep association metric,” in *International Conference on Image Processing, ICIP*, Beijing, China: IEEE Computer Society, Feb. 2018, pp. 3645–3649. doi: 10.1109/ICIP.2017.8296962.

[62] X. Chen, K. Kundu, Y. Zhu, H. Ma, S. Fidler, and R. Urtasun, “3D Object Proposals Using Stereo Imagery for Accurate Object Class Detection,” *IEEE Trans Pattern Anal Mach Intell*, vol. 40, no. 5, pp. 1259–1272, May 2018, doi: 10.1109/TPAMI.2017.2706685.

[63] S. Chopra, R. Hadsell, and Y. LeCun, “Learning a similarity metric discriminatively, with application to face verification,” in *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition, CVPR 2005*, San Diego, CA, USA: IEEE Computer Society, 2005, pp. 539–546. doi: 10.1109/CVPR.2005.202.

[64] L. Bertinetto, J. Valmadre, J. F. Henriques, A. Vedaldi, and P. H. S. Torr, “Fully-convolutional siamese networks for object tracking,” in *European Conference on Computer Vision, ECCV 2016 workshops*, Springer Verlag, Nov. 2016, pp. 850–865. doi: 10.1007/978-3-319-48881-3_56/TABLES/3.

[65] N. I. Mokhtari, “What are Siamese Neural Networks in Deep Learning?,” *Towards Data Science*, Feb. 15, 2022. https://towardsdatascience.com/what-are-siamese-neural-networks-in-deep-learning-bb092f749dcb (accessed Apr. 10, 2023).

[66] G. Koch, R. Zemel, and R. S. workshop, “Siamese neural networks for one-shot image recognition,” in *Proceedings of the 32nd International Conference on Machine Learning, ICML deep learning workshop*, F. Bach and D. Blei, Eds., Lille, France: JMLR.org, Jul. 2015. Accessed: Apr. 10, 2023. [Online]. Available: http://www.cs.toronto.edu/~gkoch/files/msc-thesis.pdf

[67] S. Guo *et al.*, “A Review of Deep Learning-Based Visual Multi-Object Tracking Algorithms for Autonomous Driving,” *Applied Sciences*, vol. 12, no. 21, p. 10741, Oct. 2022, doi: 10.3390/app122110741.

[68] D. Zhao, H. Fu, L. Xiao, T. Wu, and B. Dai, “Multi-Object Tracking with Correlation Filter for Autonomous Vehicle,” *Sensors*, vol. 18, no. 7, p. 2004, Jun. 2018, doi: 10.3390/s18072004.

[69] L. Tan, X. Dong, Y. Ma, and C. Yu, “A Multiple Object Tracking Algorithm Based on YOLO Detection,” in *2018 11th International Congress on Image and Signal Processing, BioMedical Engineering and Informatics (CISP-BMEI)*, IEEE, Oct. 2018, pp. 1–5. doi: 10.1109/CISP-BMEI.2018.8633009.

[70] C. Kim, L. Fuxin, M. Alotaibi, and J. M. Rehg, “Discriminative Appearance Modeling with Multi-track Pooling for Real-time Multi-object Tracking,” in *2021 IEEE/CVF Conference on*

Computer Vision and Pattern Recognition (CVPR), IEEE, Jun. 2021, pp. 9548–9557. doi: 10.1109/CVPR46437.2021.00943.

[71] X. Lin, C.-T. Li, V. Sanchez, and C. Maple, "On the detection-to-track association for online multi-object tracking," *Pattern Recognit Lett*, vol. 146, pp. 200–207, Jun. 2021, doi: 10.1016/j.patrec.2021.03.022.

[72] H. Liang, T. Wu, Q. Zhang, and H. Zhou, "Non-Maximum Suppression Performs Later in Multi-Object Tracking," *Applied Sciences*, vol. 12, no. 7, p. 3334, Mar. 2022, doi: 10.3390/app12073334.

[73] J. Solawetz and Francesco, "What is YOLOv8? The Ultimate Guide.," *Roboflow, Inc.*, Jan. 11, 2023. https://blog.roboflow.com/whats-new-in-yolov8/ (accessed Apr. 11, 2023).