AN OPENISS FRAMEWORK SPECIALIZATION FOR DEEP LEARNING-BASED PERSON RE-IDENTIFICATION

Haotao Lai

A thesis

in

The Department

of

Computer Science and Software Engineering

Presented in Partial Fulfillment of the Requirements

For the Degree of Master of Science (Computer Science) at

Concordia University

Montréal, Québec, Canada

August 2019

Concordia University

School of Graduate Studies

This is to certify that the thesis prepared

By: **Haotao Lai**

Entitled: **An OpenISS Framework Specialization for Deep Learning-based Person Re-identification**

and submitted in partial fulfillment of the requirements for the degree of

Master of Science (Computer Science)

complies with the regulations of this University and meets the accepted standards with respect to originality and quality.

Signed by the final examining committee:

_____ Chair
Dr. Aiman Hanna

_____ Examiner
Dr. Nematollaah Shiri

_____ Examiner
Dr. Jinqiu Yang

_____ Supervisor
Dr. Joey Paquet

_____ Supervisor
Dr. Serguei A. Mokhov

Approved by _____
Chair of Department or Graduate Program Director

_____ 20 _____ _____
Dr. Amir Asif, Dean
Gina Cody School of Engineering and Computer Science

Abstract

An OpenISS Framework Specialization for Deep Learning-based Person Re-identification

Haotao Lai

Person detection and person re-identification are rapidly increasing research areas in computer vision. They are independent but related. In fact, the output of person detection is the input of person re-identification. There are a certain number of solutions for each of these two individual tasks. But currently, there is no existing solution that can combine them to form an integrated working pipeline.

To fill the gap, we propose a highly modular and structural framework solution that provides the functionalities including not only cross-language invocation and pipeline execution mechanism but also viewer, device, tracker, detector, and recognizer abstraction. We instantiate the proposed framework to achieve our goal of tracking the same person across multiple cameras, which essentially is the combination of person detection and person re-identification. Besides the main task of person re-identification, we also support skeleton tracking, as well as camera calibration, image alignment and green screen image which commonly comes with a computer vision framework. We evaluate our proposed solution according to the requirements and usage scenarios and report the major metrics used by the research community for person detection and person re-identification tasks, respectively.

Acknowledgments

During the time I was working on this thesis, I was lucky enough to get help, suggestion and advice from many people. I would like to firstly offer my sincere gratitude to my co-supervisors Dr. Joey Paquet and Dr. Serguei Mokhov for their indispensable supervision and guidance. Also, I need to say thank you to my lab mates, Yiran Shen, Jashanjot Singh and Jyotsana Gupta for their valuable suggestions. Besides that I would like to thank my whole family, especially my parents Yong Lai and Qin Luo for their support and understanding.

During the time I was working on my MSc degree, I was fortunate enough to know many good friends in this beautiful city — Montreal. I would like to express my great gratitude to them for their encouragement and companionship. They are: Yixin Yao, Outong Li, Chen Feng, Xingjian Zhang, Bo Li, Qinwei Luo, Jingye Hou, and Jing Yang.

Lastly, I also need to say thank you to my friends who are not in Montreal but in my hometown Guangzhou. Even though with such long distance and 12 hours jet-lag, when I was depressed or felt stressed, I still could get their greeting and phone call frequently. They are: Zijian Kong, Haien Long, and Xuyi Huang.

Contents

List of Figures

List of Tables

Acronyms

CMC Cumulative Matching Characteristics.

CNN Convolutional Neural Network.

FR Functional Requirement.

IOU Intersection over Union.

ISS Illimitable Space System.

mAP mean average precision.

NFR Non-functional Requirement.

R-CNN Region with CNN.

ReID Re-identification.

SPP-net Spatial Pyramid Pooling Network.

SSD Singe Shot Detector.

YOLO You Only Look Once.

Chapter 1

Introduction

In this chapter, we will first introduce our team's previous work named Illimitable Space System (ISS) then point out the pain spots we encounter when using it. Motivated by these issues, we propose the goal we would like to achieve in this thesis. Furthermore, we define our research problems and extract requirements from usage scenarios. Finally, we discuss our contributions, followed by a brief introduction to each of the following chapters.

1.1 Background

Due to our previous work [34] named ISSv2, we were able to create real-time motion capture, projection mapping and artistic performance on the stage with one camera. Some example images of our performance are shown as Figure 1.

ISS is a real-time interactive configurable artist's toolbox used to create music visualizations, visual effects and interactive documentary based on the inputs from users such as gestures or voice. The goal of ISS is to enhance interaction between actors and graphics so that it is all projected as an integrated piece. It aims at freeing up the artists as much as possible so that they are able to perform freely in their own way without worrying about the performance technology. ISS was originally proposed in [46] and improved in [34], in order to differentiate them, we named the former ISSv1 and the latter ISSv2. Also, there was ISSv3 [48, 60] which was designed

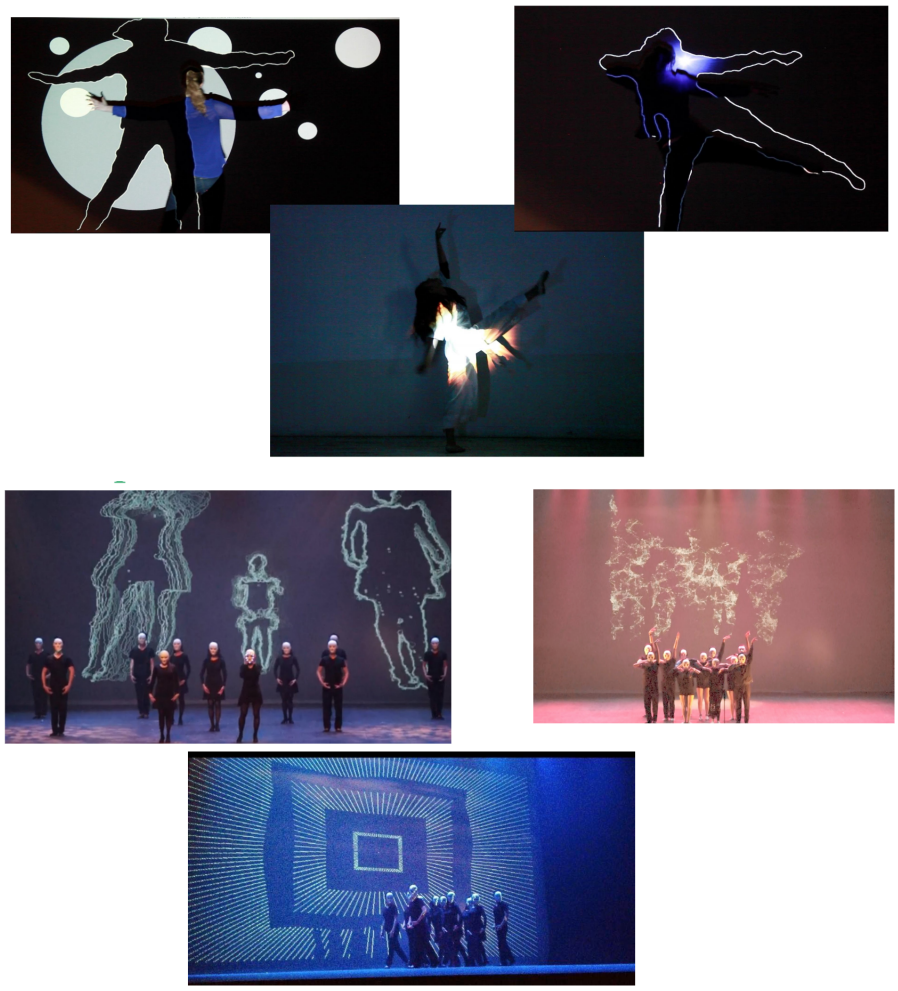


Figure 1: Artistic show produced by ISS

for virtual reality applications, but out of the scope of this thesis. We mainly focus on ISSv2 which can be conceptually represented by Figure 2, which is a significant improvement over ISSv1 and is used for rapid development of real-time, motion-based graphics applications implemented using Processing which is a software sketchbook for visual arts written in Java.

1.2 Limitations of ISSv2

When we have more chances to do more performances in different places, we found that sometimes the stage given to us is too large and cannot be covered by only a single camera [47, 49]. It restricts us to design the performance within a limited

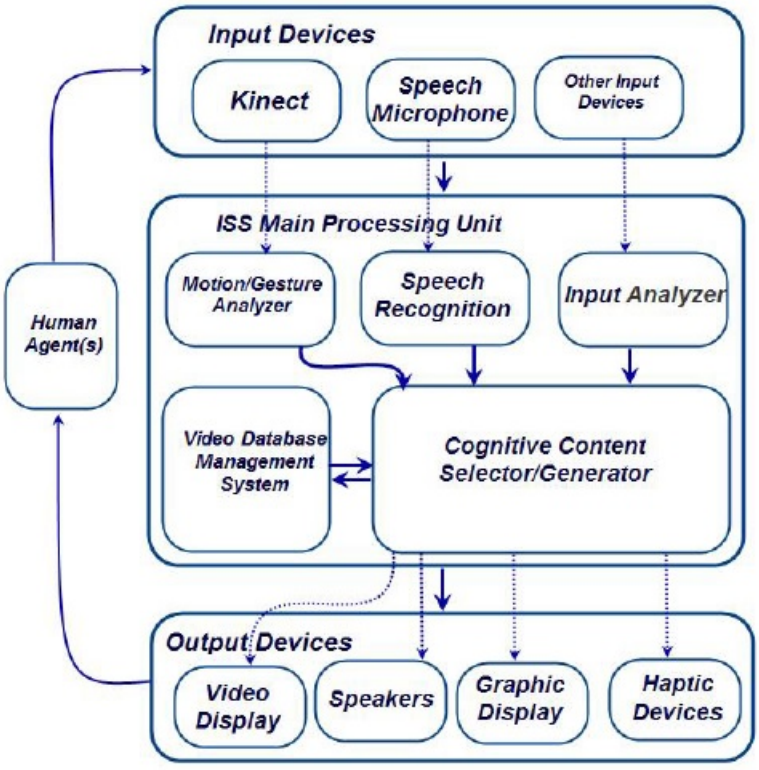


Figure 2: Block diagram of Illimitable Space System (ISS)

space, which is actually conflicting with our project's name, Illimitable Space System. Also, recently the price of consumer-level depth camera has become much cheaper. In the market, there are different kinds of depth cameras being manufactured (like Kinect v1, Kinect v2 and RealSence) and these cameras have become more and more powerful (higher speed, frame rate, resolution, larger bandwidth and etc). But ISSv2 is hardware-dependent, as it can work only with Kinect v1 which is kind of out of date now.

Under such a situation, we started to think that if one camera is not enough, we can have more than one and each of them takes care of a certain area of the large stage so that we break the restriction and can design much better performance without space limitation. By using more than one camera, a problem comes to our mind naturally: How can we identify the same person across different cameras since we need to track them and apply specific visual effects on certain actors? From this

point, developing a system that can track people across multiple cameras becomes essential. This idea can not only benefit us but also the security-and-protection industry or even the police department. Since with such a system, one can identify a specific target (like suspect) across multiple cameras if he/she is captured by any of the cameras.

In order to catch up with the device evolution, we would like to move from older devices to the latest models. But we don't want to discard our previous compatibility while evolving to new technologies. So how to enable our previous work to be compatible with various kinds of cameras is also a challenge in our works.

1.3 Research Problem

From the situation described above, our research tasks can be intuitively divided into two parts: (1) person re-identification and (2) camera abstraction. Person re-identification (ReID) recently is a hot research topic in the computer vision community. It requires the system to identify the same person across different cameras, which can be further broken down into three components:

- person detection
- person tracking
- person retrieval

If we have a fast enough system, we don't explicitly need person tracking. Instead, we can perform person detection on each coming frame from the camera which is functionally equivalent to person tracking. Under this consideration, we omit person tracking in this thesis. For device encapsulation, we focus on the total of three kinds of cameras: Kinect v1, Kinect v2 and RealSense D435, and will ensure that when a new device needs to be appended, the process will be simple, easy to implement, and will have no side-effect with the existing devices.

In the following subsections, we are going to formulate our research problem scientifically. For person detection, we enlarge our target set not only for the person

but also for all kinds of objects and the same applies to person retrieval. So we end up with object detection and object retrieval.

1.3.1 Object Detection

In object detection, we are given an image I and a list of classes C which the objects appear in I belong to. The task is to detect instances of the object within I belong to a specific class in C. For each instance i, we need to output first $c \in C$ which represents which class this instance belongs to and second a bounding box B to indicate the location of that instance in image I.

1.3.2 Object Retrieval

In object retrieval, we are given a query image q and a set of gallery images G. The task is to find the most likely image $g \in G$ for which both q and g represent the same instance $instance(q) = instance(g)$.

1.3.3 Device Abstraction

The second part we mentioned above is that we try to conceptually eliminate the differences among various kinds of cameras. It can be translated as we would like to access data via a set of common APIs without considering what kind of hardware we are using. Assume we have a list of device $D = d_1, d_2, ..., d_n$ and a list of API $F = f_1, f_2, ..., f_n$, we can trigger the same effect while calling the same API which can be mathematically expressed as: $\exists f_n \in F, \forall (d_i, d_j) \in D \Longrightarrow f_n(d_i) = f_n(d_j)$.

1.4 Motivation and Goal

The original design of ISSv2 targets to create music visualization, visual effects, and interactive documentary easily for artistic people who don't have extensive knowledge in computer science and programming. So the architecture and the design needs to be relatively simple. The main focus should be given to visual effect design and how

to display them to the audiences. What's more, currently, ISSv2 can only use Kinect v1 as the input device. Efforts have been put to enable Kinect v2 but due to the low-level dependencies (e.g. hardware driver) conflict, not all the features can be replicated and compatible.

Based on the limitation we found and some new demands, we conducted a comprehensive survey and found that there is no existing solution targeting our problem directly. So we would like to abstract a back-end system for ISS while keeping the front-end remaining unchanged. The back-end system here means the hardware, scheduling algorithm, pipeline construction, and other common APIs. Front-end basically means the artistic part, like visual effect design, music visualization and so on.

It is worthwhile to mention that as this work is being developed, there are another two other research works going on in parallel under the same umbrella. Jashanjot Singh is working on a system that can do gesture tracking and recognition while Yiran Shen is working on a system that can do facial landmarks detection and facial expressions recognition. We would also like these two works to be accessed via the same set of APIs which means all these three works should somehow be operated within the same operational software framework.

Here is a summary of our goal: **we would like to design and implement a system that provides a way to abstract different kinds of depth cameras and the functionality of person re-identification. It should also support integrating with other modules and enjoy good extensibility and usability.**

1.5 Scenario and Requirement

With the goal we defined above, in this section we will give a few concrete use-case scenarios we expect to achieve in this thesis. These scenarios, in their own way, highlight one or more problems that have not been solved by any existing solution yet. We not only aim at solving these problems individually, but also to provide a general solution to all these problems under the same software solution. We analyze these

scenarios one by one, then extract both functional and non-functional requirements (FR and NFR), which becomes the concrete implementation goal of our solution.

1.5.1 Device Switch and New Device Addition

Imagine a scenario where we would like to develop a new version of ISS may be named ISSv4. This time, we need to support device D_1 and D_2 where D_1 was supported by its previous version and D_2 is a newly added device. The difference between D_1 and D_2 is that D_1 was designed for the indoor environment while D_2 can perform better in the outdoor environment. So depending on where the performance will be given, we need to be able to switch between D_1 and D_2. This kind of switch should just literally unplug one device from the system and plug in the other one. Only a few or even no modifications should be made in the code to obtain the same effect from the application point of view. Also, if later a new device D_3 comes to the market, the system should be easily extended to be able to make use of D_3.

This usage scenario can be abstractly summed-up as the following, which becomes two of our requirements:

- **FR1**: The solution shall provide an abstraction layer for the hardware that enables the physical device transparency property to the users.
- **FR2**: The solution shall ensure the extensibility of the abstraction layer required in **FR1** which means when the new devices come only a few or no modification need to be made and will not affect the existing system.

1.5.2 Back-end Abstraction

Let's continue using the scenario setting we described above, this time we need to map the performance onto another backdrop rather than the one where the actual performance is taking place, which means that we need to extract only the actors out with all the other background removed. Keep in mind that there are two different devices supported D_1 and D_2, according to our settings. The most straightforward

way to do is that for each device, we create a filtering algorithm employing some methods provided by the hardware driver to perform background removal. But the limitation is also obvious when a new device is added: you have to re-implement the same algorithm again and again for each new device. If another demand is required, you need to implement all of them when a new device is being supported. Another elegant solution is that we could extract the data needed to perform background subtraction into a common data structure then apply a general algorithm based on it. Next time, when a new device comes, what we need to do will be just the transformation from the device-specific data structure to our common one. If some other demands like background removal are required, we can always follow the same pattern to solve them. We call all these common data structures and common algorithms as the back-end of the system.

This usage scenario can be abstractly summed-up as the following, which becomes one of our requirements:

- **FR3**: The solution shall be able to serve as a back-end for the existing ISS system providing a set of commonly used data structures and functionalities for reusability.

1.5.3 Person Re-identification (ReID)

Imagine a scenario where there is a show given by two actors, and we would like to project visual effect vfx_1 on actor a_1 while vfx_2 on actor a_2. Unfortunately, the stage is too large and cannot be covered by a single camera. We have to employ two cameras, each of them covers half the space of the stage. According to the performance director, we need to make sure that no matter where these two actors are, the visual effects have to be mapped properly.

This usage scenario can be abstractly summed-up as the following, which becomes one of our requirements:

- **FR4**: The solution shall be able to detect all the appearance of human bodies and provide their location by bounding boxes.

- **FR5**: The solution shall be able to recognize a cropped image with an identity across multiple cameras if the identity has been defined in advance.

1.5.4 Skeleton Tracking

In ISSv2, we have some visual effects designed specifically for the human skeleton. But because of the underlying dependencies issue, we can only obtain the skeleton data from Kinect v1 camera. In order to lift the restriction from the specific device-oriented dependencies, we need to make the skeleton extraction process device-independent. This means that we should be able to extract skeleton data from a variety of devices and convert them into the common skeleton data format which is simply a set of points in a picture. A possible scenario can be described as the following: a performance director would like to make use of some skeleton-based visual effect and want them to be used with different devices.

This usage scenario can be abstractly summed-up as the following, which becomes one of our requirements:

- **FR6**: The solution shall provide the functionality to enable users to perform skeleton tracking among various kinds of cameras.

1.5.5 Interaction with Other Modules

As mentioned before, there are gesture and facial recognition modules being developed concurrently in our research. Theses modules should be able to use the infrastructure (e.g. device abstraction) we proposed in this thesis. Also the module we develop here should be able to communicate with these two other modules from other system developers. Imagine a scenario where the visual effect needs to change according to the actor's gestures. When the actor push their hand, a zoom-out effect should be applied while the actor pull their hand, a zoom-in effect occurs.

This usage scenario can be abstractly summed-up as the following, which becomes one of our requirements:

- **FR7**: The solution shall provide fundamental infrastructure for other modules to use and vice versa. It shall enable an abstract communication infrastructure common to all developed modules.

1.5.6 Non-functional Requirements

By analyzing the scenarios described above, we found that in order to achieve them our solution must meet the following non-functional requirements:

Real-time Response: *The system should be able to process the task at a speed of at least 10 frames per second (FPS).*

As our solution aims to address covering a large area on the stage during a live performance issue, it is extremely important for our solution to keep the whole person re-identification process in real-time since it is a live show and the audiences are watching it on the scene. If we cannot make it in real-time, our solution is actually useless, which means it is a hard constraint for our solution. According to [1], the human visual system can process 10 to 12 images per second (FPS) and perceive them individually, while higher rates are perceived as motion. So we set 10 FPS to be our baseline.

Accuracy: *The system should be able to detect the appearance of a person and re-identify them across multiple cameras with an accuracy comparable to the state-of-the-art solutions.*

While keeping the real-time requirement, we also need to make sure our solution provides a considerable accuracy for both the detection and retrieval processes. In our case, since we are in the context of real-time artistic performance, it is hard to state an acceptable base line for it. But we should try our best to achieve a higher accuracy making sure our solution is comparable to existing state-of-the-art solutions in their respective communities.

Extensibility: *The ability of a software system to acquire and integrate new components.*

As the devices will become more and more multitudinous and the algorithms for person re-identification will be more and more advanced, it becomes essential to

design a solution with great extensibility that later on will still be usable to add or integrate more devices and algorithms into our existing solution smoothly and easily.

Usability: *The ability of a software system to effectively provide the expected functionalities to the user, further more, it should be also easy to use.*

For any kind of software system, in order to attract the users and/or programmers, as the software solution designer we should try our best to provide simple, meaningful and understandable APIs. That means the name of our APIs, the required parameters for each functionality and the final result from the return value should be concise, compact and logically make sense. For the experienced users in the same area, they should be able to move from other similar solutions to ours without much effort.

1.6 Contribution

Our contribution is five-fold:

- We offer an overall architecture that allows users to perform real-time skeleton tracking, person detection, and person re-identification.
- We offer a way in general that can allow the user to access different depth cameras within the same set of APIs with good extensibility.
- We offer a way to allow cross cameras tracking with a pluggable detector and recognizer.
- We offer a pipeline execution mechanism to allow the user to assemble various components to form their application without knowing the underlying details.
- We offer a way for the researcher to enable person re-identification experiment on top of TensorFlow and Keras.

To achieve the above, we implemented the following features:

- Design and implement a modular framework solution which consists of the core and specialized frameworks that enable real-time skeleton tracking, person detection, and person re-identification.

- Design a device module that encapsulates the depth cameras from different brands with good extensibility. Instantiate this module to support three kinds of physical devices.
- Design and implement a pipeline module that provides a linear execution mechanism over a series of filter that each encapsulate a specific transformation step.
- Design a detector specialized framework and instantiate it for person detection task with a deep learning-based algorithm named YOLO.
- Design a recognizer specialized framework and instantiate it for person re-identification using the deep learning-based identification and triplet models.
- Use the general framework solution to build several commonly used applications like camera calibration, green screen image, and image alignment.
- Implement an abstraction layer for deep learning-based person re-identification dataset to allow training and validation among multiple dataset easily.

1.7 Thesis Outline

In this chapter, we introduced our research background, pointed out the existing limitations, and stated the research problems giving them clear definitions and restricted our scope. In the following chapters, the thesis will be structured in the way listed below:

- In Chapter 2, we will review existing literature related to our research problem and also the available software which may be useful for our implementation. In the summary section of this chapter, based on our needs, we will select one and switch to it as our target detector and recognizer in our implementation described in the following chapters.

- In Chapter 3 and Chapter 4, we will propose our framework solution in detail in a top-down manner. Precisely, in Chapter 3 we describe our design of the solution framework and in Chapter 4 we describe how it has been instantiated to satisfy the needs.

- In Chapter 5, we will describe the applications built on top of our proposed solution, which becomes a proof of concept that our solution can actually fulfill our listed requirements.

- In Chapter 6, we will first demonstrate both the functional and non-functional requirements are really fulfilled by our framework solution. Then we report our results showing the benchmarks for the main components in our solution with commonly acknowledged metrics.

- In Chapter 7, we sum up our work with advantages and limitations and point out some potential research directions in the future.

Chapter 2

Related Work

As discussed in the previous chapter, we split our work into two parts: (1) person re-identification (2) device abstraction. In this chapter, we are going to introduce the background for each part. For the person re-identification part, we will review the literature related to object detection and person re-identification, which the former is a prerequisite of the latter. For the device abstraction part, we are going to examine currently available software which may become useful for our objectives and be adopted as dependencies in our solution.

2.1 Object Detection

Object detection is one of the fundamental tasks in computer vision research. It is a natural extension of the classification problem that requires the detection of the presence of objects and to accurately locate them within the given image. This subject has been explored by many researchers for a long time and a lot of detailed surveys have been published on this topic [9, 30]. We can observe from Figure 3, since 2012 the deep learning-based approaches have dominated the object detection domain. In this section, we are going to review the two classes of literature in object detection based on deep learning methods (detailed explanation will be given as subsection for each of the items listed below).

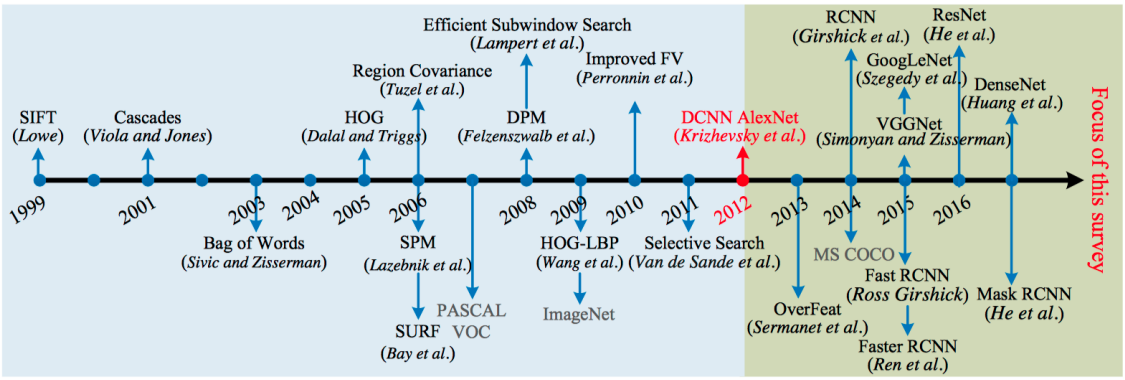


Figure 3: Timeline of various methods proposed for object detection [30]. The methods in blue area are the hand-crafted detector and those in green area are the deep learning-based approaches.

- Two stages approaches
  - R-CNN
  - SPP-net
  - Fast R-CNN
  - Faster R-CNN
  - Mask R-CNN
- One stage approaches
  - YOLO v1, v2 and v3
  - SSD

The world "stage" here means an independent process or a separate branch within the deep neural network structure. As their names imply, one-stage approaches finish the whole object detection process in one shot while the two-stages approach has an independent process to propose the areas that may contain an object then perform detection on those areas. The presence of the area proposition stage takes more time but since it is elaborated and can cover more potential areas than the one-stage approach, it can achieve better results in terms of accuracy. To determine which kind of detector to use depends on the realistic requirement of the application. In this

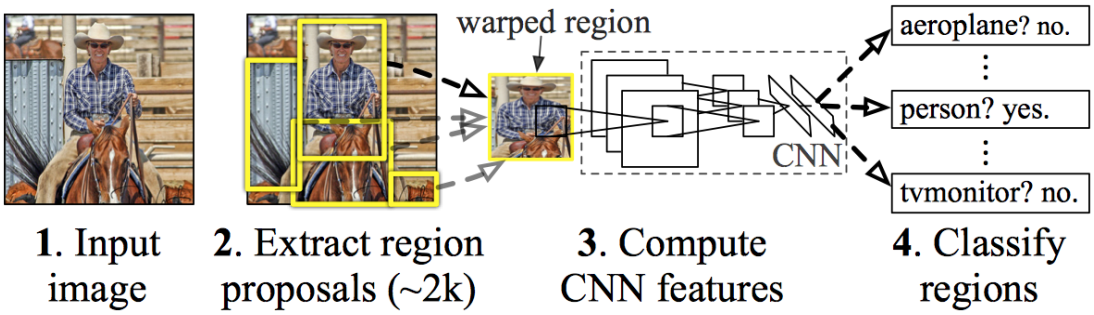


Figure 4: R-CNN system overview [17].

thesis, since we have real-time response as a non-functional requirement, one stage detector will definitely be a better choice.

2.1.1 Two Stages Detector

Most of the two stages detectors follow the same methodology. Firstly, propose candidate regions that may contain object(s) then using a convolutional neural network (CNN) to extract feature descriptors of each region. Finally, feed the descriptor into a classifier to figure out what kind of object they are if it exists, and classify them according to a set of pre-defined categories.

2.1.1.1 R-CNN

R-CNN was the seminal work of employing CNN on object detection task, the name R-CNN stands for region with CNN features. At the time it was proposed, it boosted the accuracy from 35.1% to 53.7% on PASCAL VOC dataset [17]. It designed a cascade pipeline containing four modules shown as Figure 4. There are two important points this work brought to the table: (1) it proposed a selective search [1] method to find the possible candidates replacing the old fashion sliding window method, which improves the computation time significantly. (2) it adopted CNN as a feature extractor which yields more robust features rather than the hand-crafted one.

[1]Selective Search is a region proposal algorithm used in object detection. It is designed to be fast with a very high recall. It is based on computing hierarchical grouping of similar regions based on color, texture, size and shape compatibility.

2.1.1.2 SPP-net

Even though R-CNN brought a large margin of improvement into object detection research, it still has the potential to be better, as observed by Kaiming He and his team. They proposed spatial pyramid pooling network (SPP-net) [20] one year after the publication of R-CNN. It improved both runtime efficiency and accuracy compared to R-CNN. They identified two main issues in the R-CNN solution:

- The generated candidate regions are easy to have overlap among each other which can lead to repeated computation of the same feature maps.
- When ensuring the input image to a fixed size by cropping or warping, it may leads to information missing which can affect the training significantly.

SPP-net solved the first problem by reversing the order of selective search and convolution operation. It also employed a special layer named "spatial pyramid pooling" at the end of the convolutional layer to eliminate the second problem, that is how its name comes from. By using SPP-net, the final feature maps will only be computed once and the features are pooled in arbitrary regions to generate fixed-length descriptor for training, as shown in Figure 5.

2.1.1.3 Fast R-CNN

Fast R-CNN [16], as its name suggests, is a performance-enhanced version of R-CNN. This update mainly focuses on speeding up both training and testing procedures. It targeted its improvements at not only R-CNN but also SPP-net for their common drawbacks:

- The training pipeline is a multi-stage process.
- Features are written to disk during training.
- The inference phase is too slow.

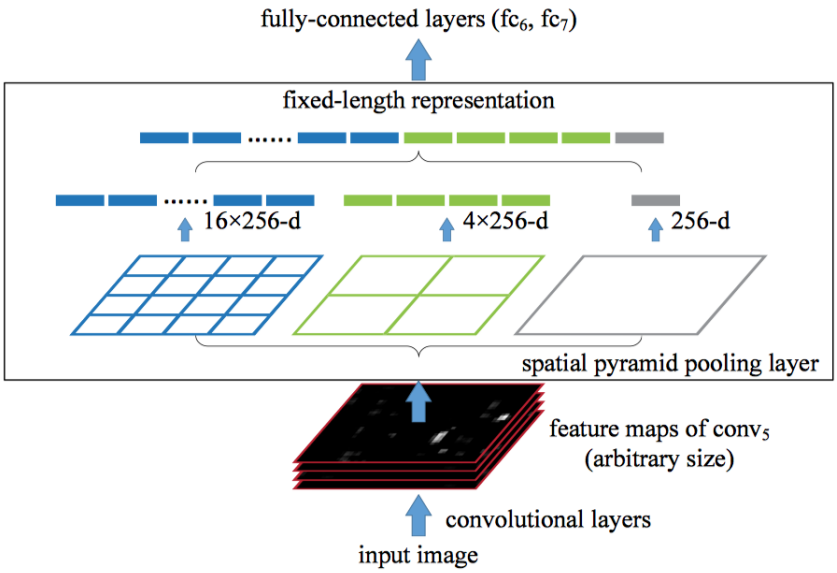


Figure 5: Illustration of how spatial pyramid layer works [20].

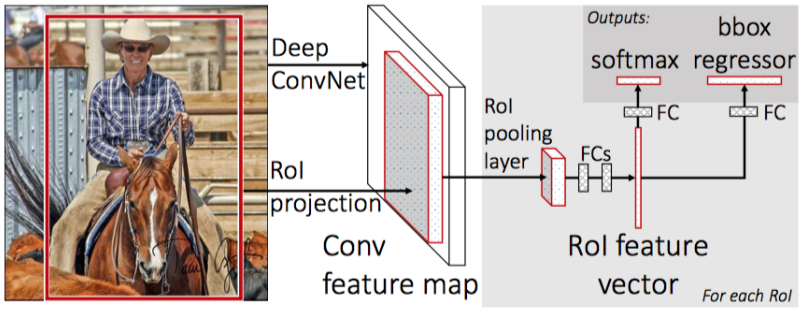


Figure 6: Parallel workflow of Fast R-CNN [16].

The solution for these issues is quite straightforward. Firstly, the author changed the cascade pipeline to become a parallel one, shown as Figure 6. Secondly, it modified the loss to be a multi-task loss which reduces the training complexity and makes all the layers updatable (the proposed fine-tuning algorithm cannot update the convolutional layer that precedes the spp layer). Thirdly, the features cached on the disk were no longer needed.

2.1.1.4 Faster R-CNN

Faster R-CNN [43] was a milestone of the usage of deep convolutional neural network in the research of object detection. It was created by the combination of the teams which proposed R-CNN and SPP-net. Before it came up, the candidate regions were calculated via a method called selective search. But in this case, they introduced a region proposal network (RPN) which was embedded as a branch into the model that can learn how to produce reliable candidate regions during the training time, the overall structure of Faster R-CNN as shown in Figure 7. By using such a solution, the model can achieve object bounding box prediction and object classification simultaneously (through one forward pass). There are several advantages of Faster R-CNN compared to previous works:

- Deep learning features are more reliable than the selective search one.
- The whole network can be trained end-to-end [2].
- The whole pipeline can be done on GPU (selective search need to be done on CPU before) which can speed up the training time.

There is one more thing that needs to be pointed out, this work introduced the concept of "anchor" which has been widely used in the latter object detection research including the one stage detector we are going to review in the next subsection. At each sliding-window position, the network would simultaneously predict k region proposals relative to k reference bounding boxes. These reference boxes are called anchor, an example of which is shown in Figure 8.

2.1.1.5 Mask R-CNN

The most recent work of the R-CNN-based research, called Mask R-CNN [19], was proposed by Kaiming He again. It was an object detection network also which could be used in the object segmentation domain. The idea was that (1) it added another

[2]The learning that optimizes the network weights by considering the inputs and outputs directly is called end-to-end learning.

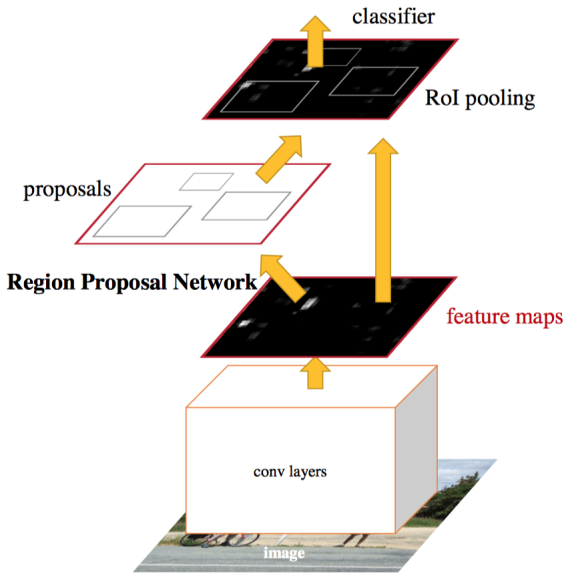


Figure 7: Structure of Faster R-CNN [43] network.

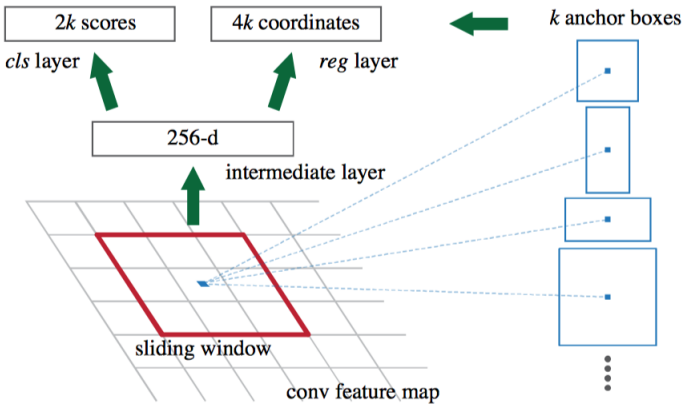


Figure 8: Illustration of the anchor mechanism [43].

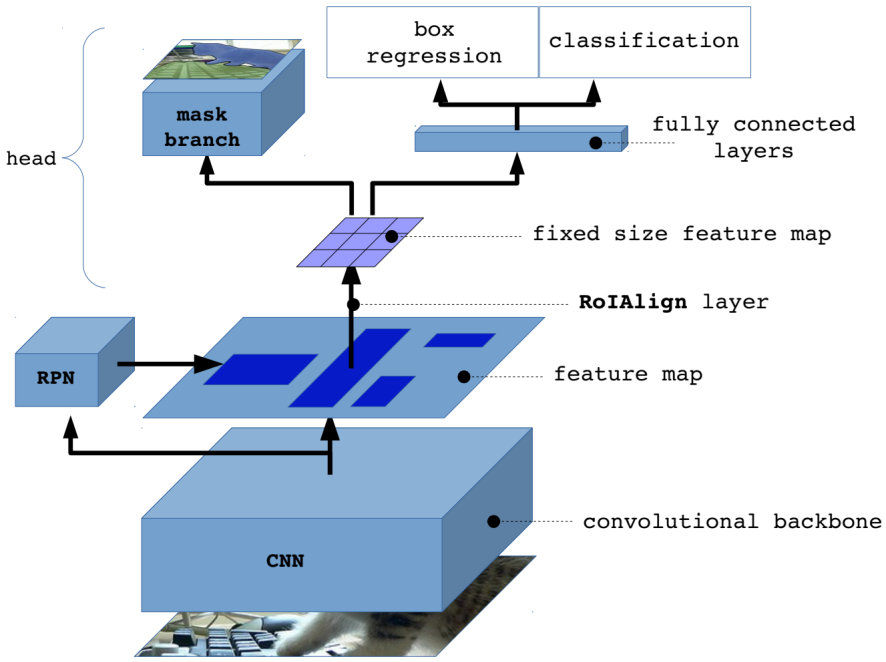


Figure 9: Architecture of Mask R-CNN [14].

mask branch into the network to create a mask for each detected object instance and (2) it used a more advanced backbone network for feature extraction (e.g. ResNet and FPN). Its architecture is shown in Figure 9. Again, the loss used to guide the training is the multi-task loss from Faster R-CNN with the mask loss added, expressed as: $L = L_{cls} + L_{bbox} + L_{mask}$. Another creative idea, ROI alignment, is notable to mention which can improve accuracy for bounding boxes regression.

2.1.2 One Stage Detector

One stage detector refers to those who directly predict classification score and bounding box offsets from the input with a single feed-forward CNN network. There is no separate region proposition process that exists at all. All the computation is done within just a single network which can be optimized end-to-end directly on detection performance.

2.1.2.1 YOLO

YOLO stands for "you only look once", which indicates it is an one-stage object detector. Until now, there is a total of three versions of the YOLO algorithm. They are YOLO v1, YOLO v2, YOLO v3 which were published in 2015, 2016, 2018 respectively. Compared with the R-CNN series, YOLO doesn't have the stage of candidate region calculation. It uses a single network to directly compute the object classification score and regress the bounding boxes if the object exists.

YOLO v1 [40], this work is the fundamental building block. The latter YOLO algorithms just borrowed or added advanced techniques or tricks to improve the performance. The workflow of the algorithm can be summarized as the following steps and visualized as Figure 10:

- Take the input image (with size 448×448), cut it into $S \times S$ grids, each of them is responsible for detecting those objects whose center located within this grid.
- Each grid will predict B (an integer) bounding boxes and the confidence score of each box. For each predictive bounding box, the result should be a five-dimensional vector (x, y, w, h, c) representing the center location of the box (x, y), the width and height of the box (w, h), and the confidence score c.
- For each grid (no matter how many bounding boxes are required), it should also output a probability for all required classes (if the dataset contains 10 classes of object, it should output a probability for each class which means 10 probabilities in total).
- Sort all the results according to the score and use a threshold to filter out the detected bounding boxes whose probability is lower than the threshold.

An example process for an input image can be shown as Figure 11. From that figure we can find that the classification score computation and bounding box prediction happened in parallel. At the end, another common technique, non-maximum suppression [37] was applied to eliminate the highly overlapped boxes

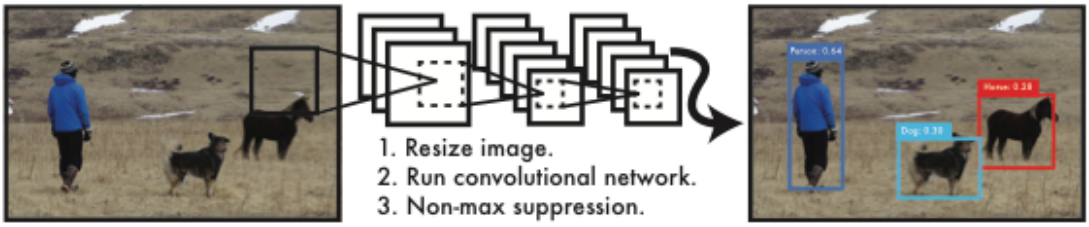


Figure 10: Workflow of YOLO v1 [40].

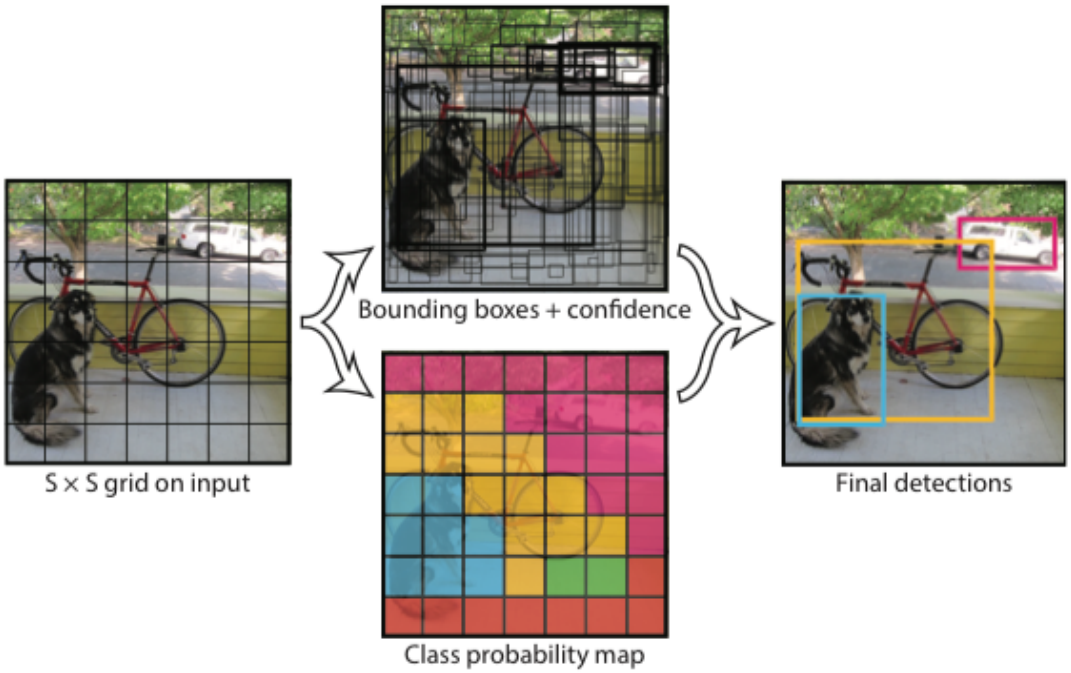


Figure 11: Process example YOLO v1 [40].

but point to the same object. From the implementation point of view, the YOLO series was written in pure C. The author provided his own deep learning framework named Darknet [39] which includes backbone network, optimizer, parameters update methods, etc. The backbone network's architecture is shown as Figure 12, which is a 24 convolutional layers network with 2 dense layers appended. The whole model was pre-trained on ImageNet using image with resolution 224×224 then doubled the size for detection.

YOLO v2 [41] is an update of the previous version to improve the performance while keeping its speed advantage (67 FPS vs. 5 FPS for Faster R-CNN). A

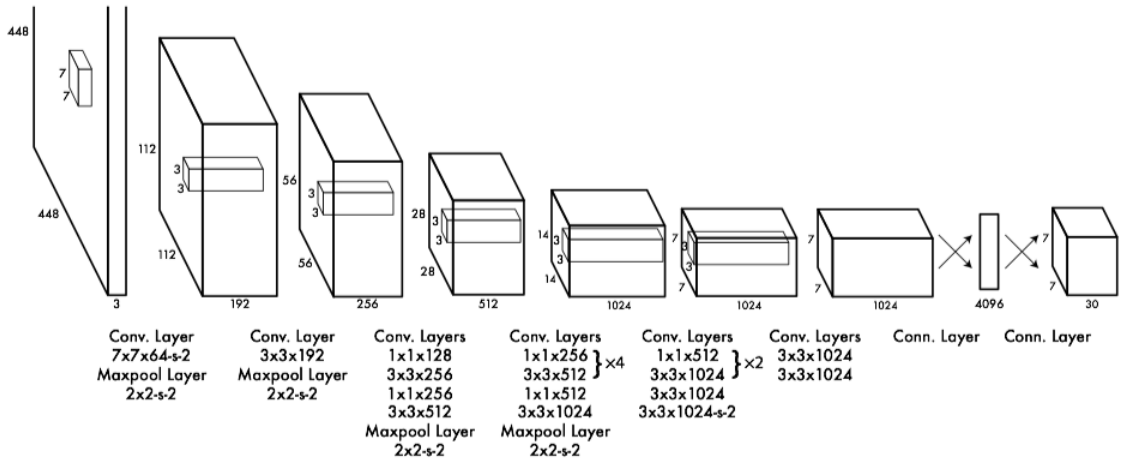


Figure 12: YOLO v1 network architecture [40].

comparison result with other detector can be found in Figure 13. The development efforts have been concentrated on:

- Added batch normalization layer to gain 2% improvement in the context of mAP .
- Pre-train on ImageNet using images with size 448×448 (double compared to YOLO v1) which increases 4% mAP.
- Introduced "anchor mechanism" from Faster R-CNN which bring 7% recall improvement.
- During training, every 10 epochs change the scale of images to obtain more powerful generalization ability.
- 13×13 resolution feature map is enough for ordinary objects but in order to overcome the disadvantage from v1 that perform poorly on small objects, a passthrough layer has been added that bring the feature from an earlier layer at 26×26 resolution.
- Proposed a new backbone network called Darknet-19 which contains 19 convolutional layers and 5 maxpooling layers.

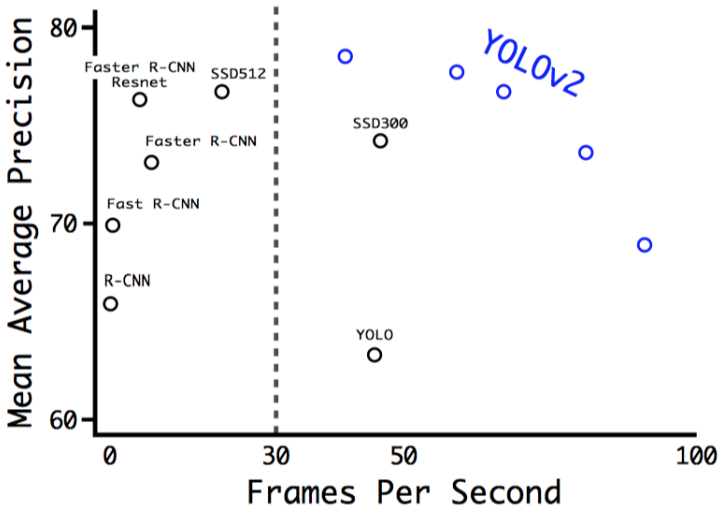


Figure 13: YOLO v2 accuracy and speed on VOC 2007 dataset [41].

YOLO v3 [42], the author tried a number of tricks and found some of them can increased the speed and accuracy. A comparison of the inference time between YOLO v3 and most of considerable methods can be shown as Figure 14. According to their paper, some significant changes have been applied in this version:

- It used a new backbone classification network design based on the works of Residual Network [21] [22] named Darknet-53 which can improve the top1 accuracy about 3.1% while maintaining the speed of 78 FPS.
- It discarded the softmax function for classification instead using a simple logistic classifier. During the training, binary cross-entropy loss was employed which can solve the problem of overlapping labels (i.e. man and person) in a complex dataset.
- It adopted a similar concept of feature pyramid networks (FPN) to enhance the scale-invariant ability of the model.

2.1.2.2 SSD

Five months after YOLO v1 was published, another famous one-stage approach, named single shot detector (SSD) was created [31]. It achieved much higher

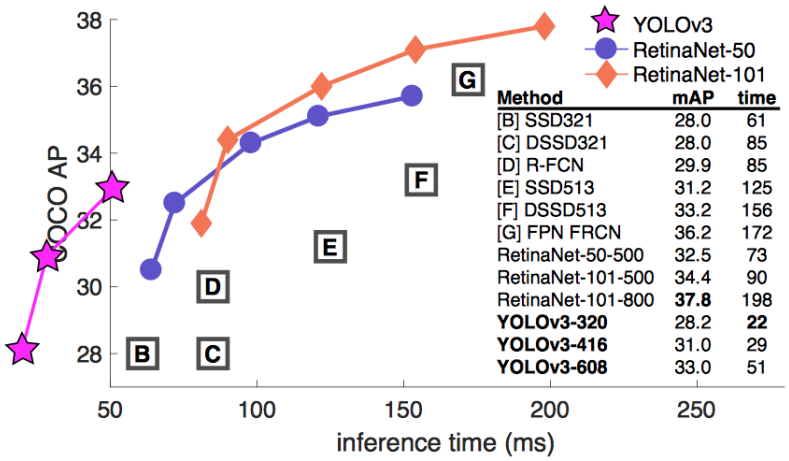


Figure 14: Inference time of YOLO v3 compared with other methods [28].

performance on mAP metric than YOLO v1, precisely, 72.1% mAP on VOC2007 dataset at 58 FPS with 300×300 input size while YOLO v1 only have 63.4 % at 45 PFS with 448×448 input size. The architecture difference between YOLO v1 and SSD can be shown by Figure 15. This performance increase can be explained by the following improvements:

- Use of the "anchor mechanism" to find a set of pre-defined bounding boxes.
- For each anchor, predict the class label and the offset of the anchor which perform better than regress the absolute location of the bounding box.
- For each input, combine feature maps with different scales in order to achieve scale invariant.

2.2 Person Re-Identification

Re-identification means we need to determine whether we encounter the same target on a previous occasion or not.

According to [55], let $\delta = \{\delta_1, ..., \delta_M\}$ represents M descriptors within a gallery set, given a probe descriptor U, the identity of this probe person can be formulated

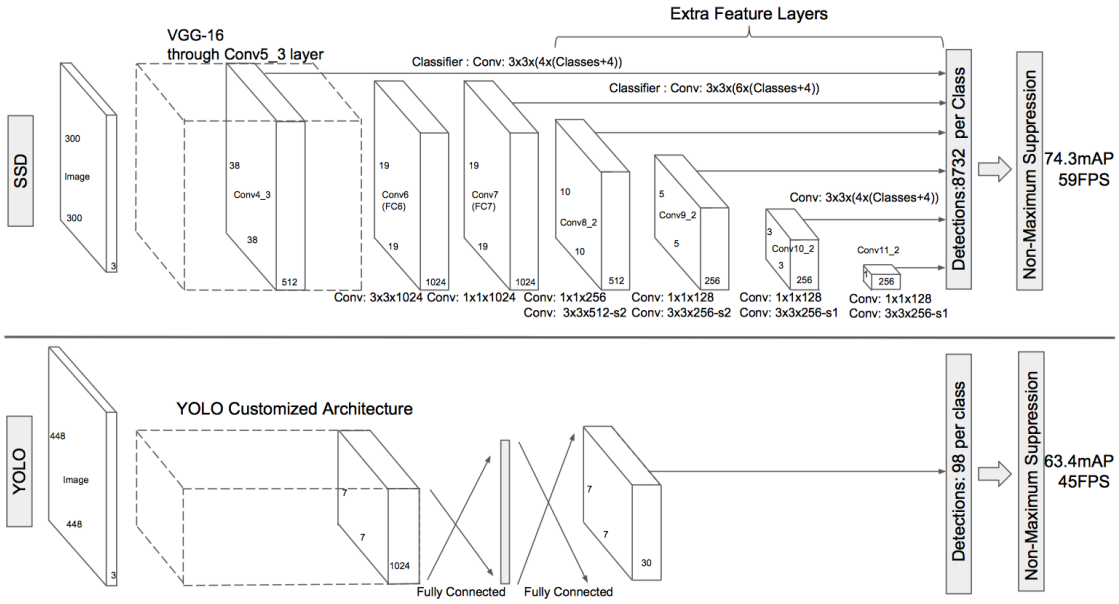


Figure 15: Network architecture comparison between SSD and YOLO v1 [31].

as:

$$I = \arg_{\delta_i} \min(dis(\delta_i, U)),\ \delta_i \in \delta \tag{1}$$

where I represent the identity of U and dis means a proper distance function which will return the distance between U and all $\delta_i \in \delta$. From Equation 1, we should notice that there are two key points of the person ReID task: (1) Feature description and (2) Distance function. How to obtain suitable feature descriptors is always an interesting research problem in the computer vision area. The traditional way to do it through a hand-crafted descriptor which is designed by the domain experts, for example, BRIEF, SIFT, SURF, and ORB descriptors. All these are hand-crafted descriptors and work well in their own domain. But for different tasks, different descriptors have to be created. Obviously, it requires a lot of work and is not very convenient.

Since 2012, AlexNet [25] got a huge success in the ImageNet classification competition with the use of deep learning. Using deep neural network as feature extractor has become popular and its performance defeats most of the descriptors created manually. In the person ReID community, more and more researchers move

	Based On	2010	2011	2012	2013	2014	2015	2016	2017	2018
CVPR	Feature designing	1	0	0	1	2	5	3	3	0
	Distance metric	0	1	1	1	0	1	4	4	2
	Deep learning	0	0	0	0	1	2	5	7	25
ICCV	Feature designing				2		4		3	
	Distance metric				0		2		2	
	Deep learning				0		0		4	
ECCV	Feature designing					2		2		0
	Distance metric					1		3		1
	Deep learning					0		2		18

Figure 16: The number of ReID papers depending on different approaches included by the three top conferences in recent years [55].

their attention to the deep learning-based features (statistic shown in Figure 16) and a lot of creative methods have been developed based on (deep) neural networks.

In this section, we are going to review the existing deep learning-based approaches for the person ReID task, most of them can be sorted into the following categories [55]:

- Identification model
- Verification model
- Distance metric-based model
- Parts-based model
- Others

We will go deeper into each of these different models in the following subsections, but mainly concentrate on the identification model and the distance metric-based model. Because our implementation which will be introduced in the next chapter is a combination of these two models.

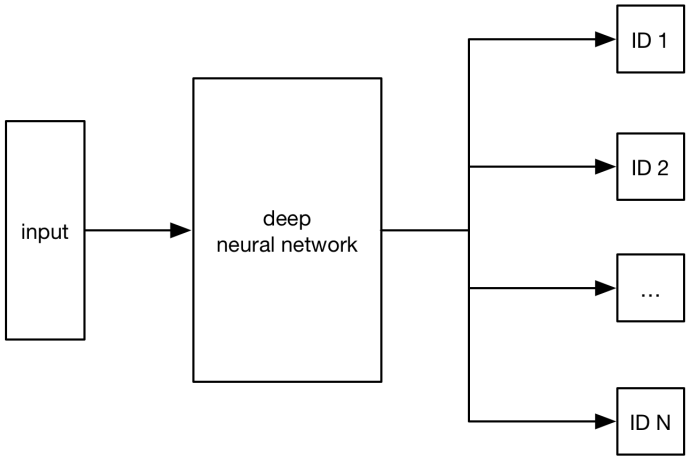


Figure 17: Architecture of identification model.

2.2.1 Identification Model

Identification model regards the ReID task as a classification task, distinct identities will be seen as different classes, the basic architecture of the model is shown as Figure 17. The input to the network is the output from some kind of person detector. Then the deep neural network (e.g. CNN) serves as a feature extractor. By making use of these extracted features the input image is tagged with an identity.

Due to the lack of data, the main issue of the classification model in deep learning is always overfitting which means that the model performs well on the training set but poor on the validation set. Especially on the person ReID task, we want more training samples for each individual but most of the dataset only have few sample per instance (e.g. VIPeR only contains two images per identity). A lot of work have been done to solve this problem, they can roughly be categorized into (1) add other constraints to revise overfitting and (2) apply data augmentation techniques or create a larger dataset.

In [57], the authors proposed a fusion feature network (FFN) which can take both the CNN and the hand-crafted features into consideration at the same time, its architecture can be shown as Figure 18. FFN takes the identity image as input then branches into two paths. For the CNN feature, five layers convolutional network and

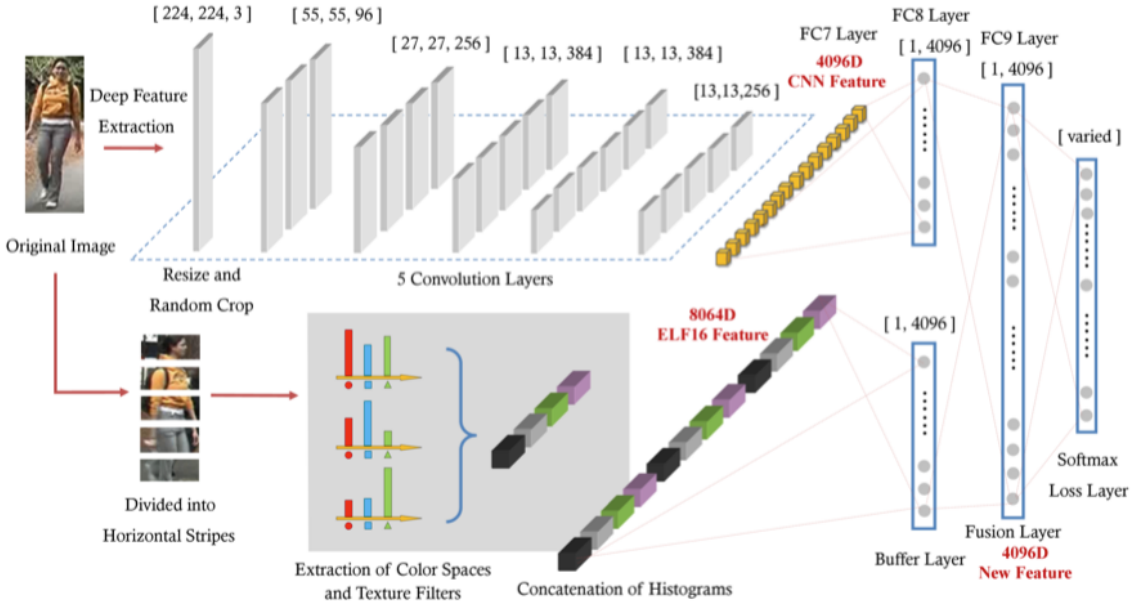


Figure 18: Architecture of fusion feature network [57].

two fully connected layers were employed. For hand-crafted feature extraction, the original image was divided into horizontal stripes then color spaces and texture filters were applied in order to extract histograms which finally would be concatenated to form a feature vector. In the end, theses two kinds of features would be linked together through another dense layer. The whole network was trained under the guidance of a cross-entropy loss function and the classification result was obtained from a softmax activation function [3] and during back-propagation parameters update would also be constrained by the hand-crafted feature [4].

The idea behind classification model is to use several hyperplanes to separate different identities in the feature spaces. But since it is a high dimensional space, in some cases, the intra-class variance may larger than inter-class variance which is not a good property for a classification model. In order to enhance the learned discriminative feature, a hybrid network architecture that combined the Fisher vectors which include color histograms and SIFT and a deep neural network has been

[3]softmax is a function that takes as input a vector of K real numbers, and normalizes it into a probability distribution consisting of K probabilities. [2]

[4]hand-crafted feature is the feature designed beforehand by human experts to describe a set of characteristics of a specific object [36].

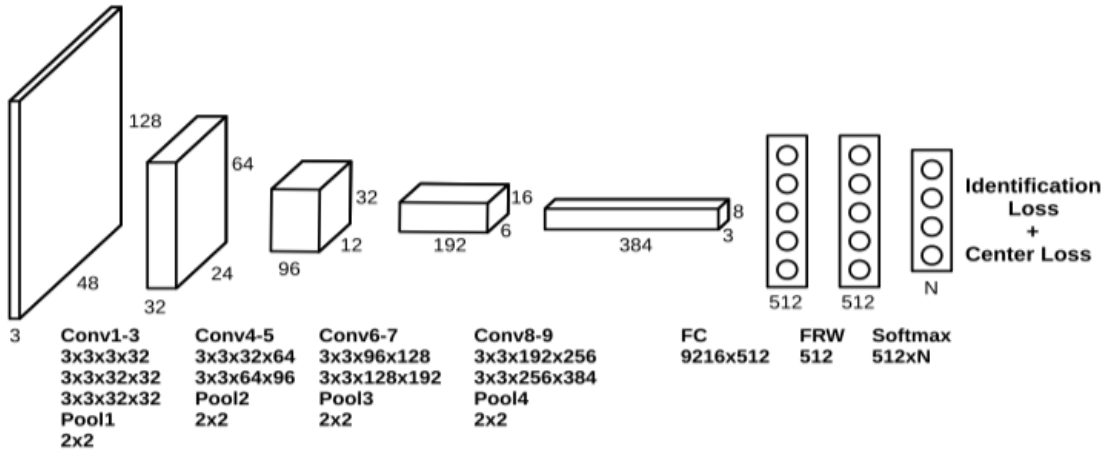


Figure 19: A proposed CNN architecture with ID loss and center loss. [24].

proposed in [56].

It is worth to mention that as deep learning in person ReID area becomes more and more popular, there are several large datasets like [26], [64], [44] and [67] that have been released as well as their corresponding evaluation protocols. Since deep learning methods really depends on data, these datasets do help the community a lot. With such large datasets, we are able to train the network directly based on the plain classification model. [58] trained the plain classification network on multiple datasets with domain-guided dropout [5] strategy aiming at obtaining a cross-domain model.

Jointly learning is another hot topic in the deep learning community, it means the model is trained under the guidance of two or even more loss (objective) functions. In specific person ReID research, [24] proposed a jointly learning method which adopted identification loss and center loss [54] at the same time, its architecture is shown as Figure 19. It claims that it is more efficient than the pairwise or triplet model and the performance is also better by using their feature reweighting layer (FRW). The core idea is that during the training time, the model tries to enlarge the inter-class variation and reduce the intra-class variation supervised by the center loss while the identification loss makes full use of the image label compared with the verification model which just used the weak label information.

[5]Dropout is an implementer-friendly but useful technique to prevent overfitting proposed by [50].

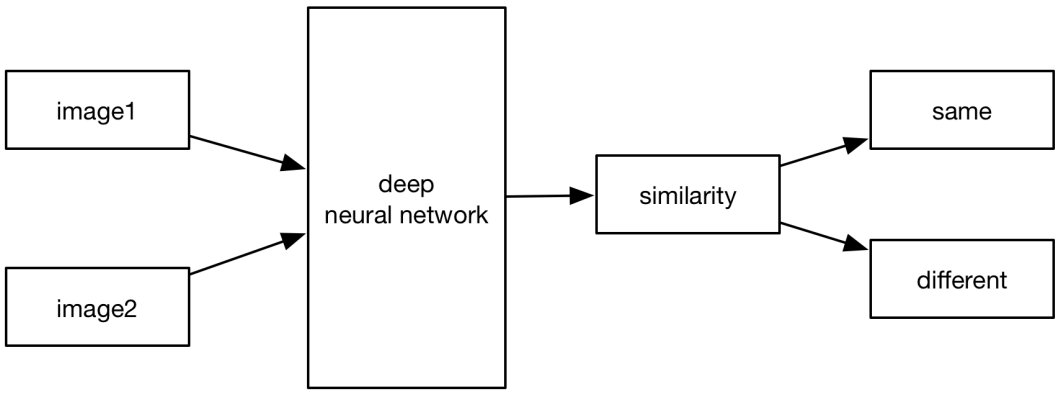


Figure 20: Architecture of verification model.

2.2.2 Verification Model

Verification Model can be seen as a classification problem as well but it is a binary version. It takes a pair of images as input and output a similarity value indicating whether the paired images is the same person or not. Its architecture can be shown as Figure 20 and simply formulated as:

$$f(x_1, x_2) = \begin{cases} 1, & y_1 = y_2 \\ 0, & y_1 \neq y_2 \end{cases}$$

The first verification model to address ReID task named filter pairing neural network (FPNN) [27] was proposed in 2014. Max-out pooling layers and patch-matching were employed to jointly handle geometric transforms and photometric, misalignment, background clutter and occlusions which are common issues in ReID domain. At the time where datasets were still extremely limited, "Siamese" deep network for metric learning which was firstly proposed by [59] was used to target this problem. Like the common verification model, it takes an image pair as input then pass them through three shared parameters but independent convolutional networks to perform operations on three non-overlapping parts of the images. The extracted feature descriptor will be flattened by a dense layer then used to compute the cosine distance which would be converted into a similarity value. If the value is greater than a hyper-parameter threshold then the input pair will be considered as the same

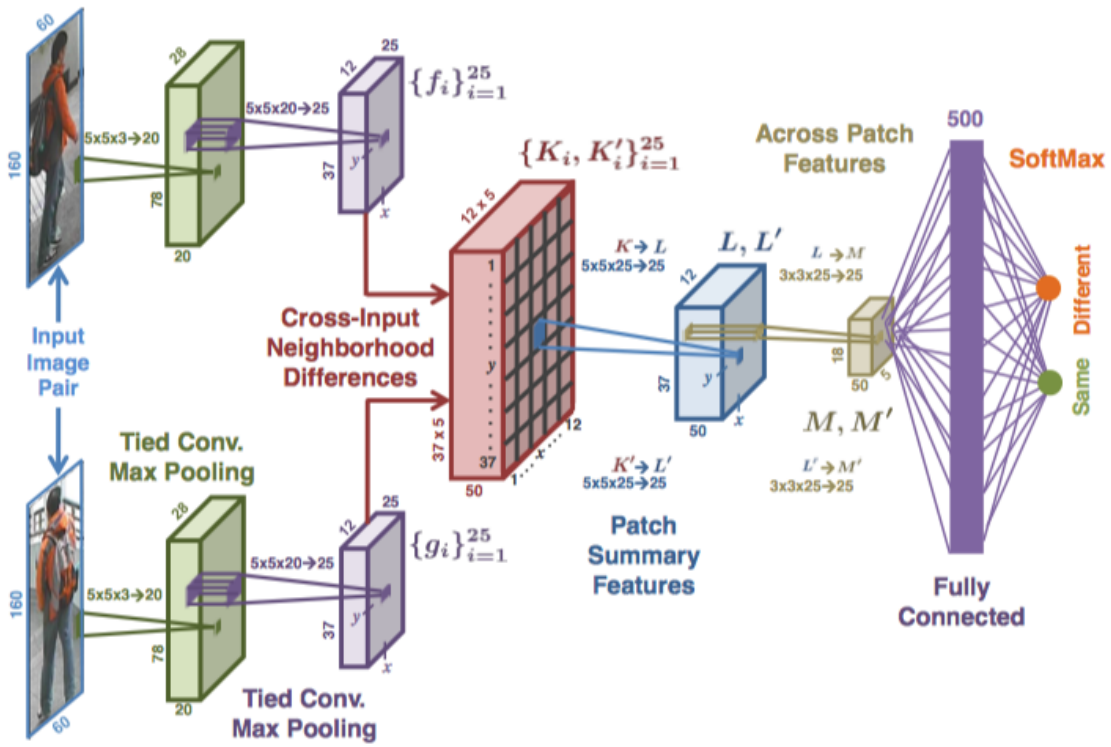


Figure 21: Architecture proposed by [10].

identity, otherwise different. Based on the previous two works, [10] comes up with an improved architecture shown as Figure 21 that include a layer to calculate cross-input neighborhood differences which is based on mid-level features to capture local relationships of the input paired images.

But the verification models introduced above all have a common problem which is that the neural network employed by them are relatively shallow. By this limitation, they did not benefit from digging the features which are discriminative enough to distinct different identities. Besides, since we have to construct the image pair for training, there will be overhead added compared to the identification model. One more thing is that the verification models only use the weak dataset label, which means that for a specific identity instance pair, it doesn't care about their actual identity but they are the same or not. Unlike the identification model, it will tell directly which identity it is for each given input.

Because of these limitations, only using the verification model may not achieve

high accuracy in ReID task. Still, it is worth to mention that the combination of identification and verification model can reach a promising result, [65] proposed such a model and comprehensively compared the advantages and disadvantages of these two models. They replaced contrastive loss by cross-entropy loss which is different from its sibling network on face recognition, then applied dropout regularization to prevent overfitting.

2.2.3 Distance Metric-based Model

The distance metric-based model aims at making the distance between the same identity as small as possible while keeping the distance between distinct identities as large as possible. One of the most popular used approach is the triplet architecture. A triplet unit of images can be defined as: $I_i = \{I_i^1, I_i^2, I_i^3\}$, where I_i^1 called anchor, (I_i^1, I_i^2) is positive pair and (I_i^1, I_i^3) is negative pair. For each triplet unit, the model will try to satisfy the following:

$$\left\|F(I_i^1) - F(I_i^2)\right\|^2 < \left\|F(I_i^1) - F(I_i^3)\right\|^2 \tag{2}$$

where $\|\|^2$ means the L2 norm and $F(I)$ denotes the features learned by the model. Based on Equation 2 we can define the loss as following:

$$L(I) = \sum_{i=1}^{n} \max\{\left\|F(I_i^1) - F(I_i^2)\right\|^2 - \left\|F(I_i^1) - F(I_i^3)\right\|^2, C\} \tag{3}$$

where C is a non-negative constant, guided by Equation 3, the network will be forced to maximize the distance between the anchor-positive and anchor-negative pair under L2 norm, the described procedure can be illustrated by Figure 22.

In [23], comprehensive research had been conducted to the triplet model, covering the strategy of sampling, different representations of the loss functions, comparison between pre-trained and plain model, etc. Some of their methods are worth to mention:

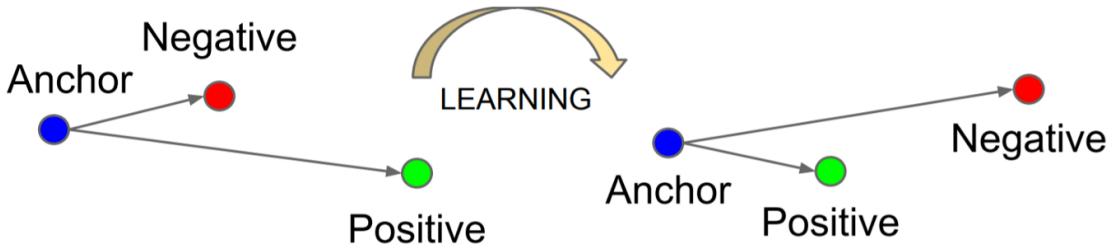


Figure 22: Objective of triplet loss [45].

- They proposed a novel way to sample the triplet unit, in which randomly sampled p identities within the whole training set and for each selected identity randomly pick k instances per mini-batch.

- They came up with several representations of loss functions, the most popular two of them are batch-hard loss L_{BH} and batch-all loss L_{BA}. As their name stated, batch-hard loss means to find the hardest triplet units per batch and use them to contribute to the loss while batch-all loss means to use all the triplet units, no matter what they are, contributing to the loss. They can be formulated as Equation 4.

$$L_{BH} = \sum_{i=1}^{P} \sum_{a=1}^{K} [m + \max_{p=1...K} D(f_\theta(x_a^i), f_\theta(x_p^i)) - \min_{\substack{j=1...P \\ n=1...K \\ j \neq i}} D(f_\theta(x_a^i), f_\theta(x_n^j))]_+ \quad (4)$$

where D is a distance function, f_θ is the feature descriptor, m is a margin constant, and $[]_+$ means the result within bracket will be a non-negative number.

$$L_{BA} = \sum_{i=1}^{P} \sum_{a=1}^{K} \sum_{\substack{p=1 \\ p \neq q}}^{K} \sum_{\substack{j=1 \\ j \neq i}}^{P} \sum_{n=1}^{K} [m + d_{j,a,n}^{i,a,p}]_+ \quad (5)$$

$$d_{j,a,n}^{i,a,p} = D(f_\theta(x_a^i), f_\theta(x_p^i)) - D(f_\theta(x_a^i), f_\theta(x_n^j))$$

Effectively, by using L_{BH} we can have PK units contributing to the loss while

using L_{BA} we have $PK(PK-K)(K-1)$ units contribute to the loss. It really depends on the realistic scenario to determine which loss we should choose. At this point, it is significant to note that these two variations of the loss still respect to the standard triplet loss function Equation 3.

- During the training, they found that using the non-squared Euclidean distance is more stable than the squared one which will make the optimization more prone to collapsing and reduce the interoperability of the margin constant (cannot be absolute distance any more).

2.2.4 Parts-based Model

A hand-crafted part-based model has been proposed for matching two persons based on their appearance [18]. It partitioned a person into horizontal stripes to extract color and texture features. After this work, several other researchers [12, 15] employed more sophisticated strategies to divide a person into parts, but still based on hand-designed approaches. When deep learning comes to the picture, with the help of the research on human pose estimation and landmark detection, the person ReID part-based model achieves several impressive results [51, 53, 63].

Attention mechanism is another milestone in the parts-based model, the current state of the art in person ReID task is produced by this model. In [29], the author first adopted attention network to address ReID task, they proposed a LSTM-based model using a recurrent approach that can output part attention feature dynamically for localizing discriminative regions of the pedestrian image. One year after, [32] came up with a multi-directional attention mechanism for capturing multiple attention information, the proposed network was named HP-Net. For the part-based model, there is always a common issue which is the alignment problem. Once you divide the image into parts, how to align these chunks and calculate the similarity properly will be problematic. To address the misalignment issue, [62] introduced a CNN-based attention model that makes use of the distance between a paired image to learn the part-feature for matching. In ECCV2018, [52] proposed a parted-based

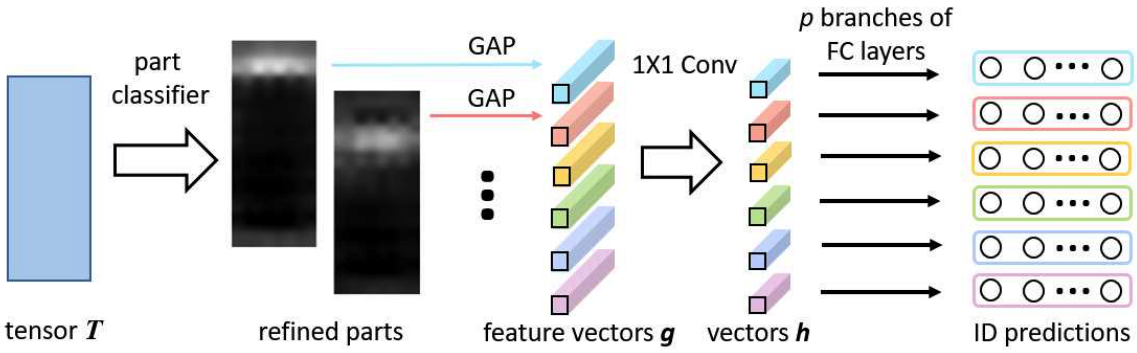


Figure 23: PCB in combination of refined part pooling [52].

convolutional baseline model (PCB) which employed a simple uniform partition strategy and assembles part-informed feature into a descriptor and a refined part pooling (RPP) method which reinforces the within-part consistency introducing a large margin of improvement without requiring any part labeling information, the architecture is shown as Figure 23. However, the part-based model still has its own limitations:

- Adding part-based branches reduced the efficiency of the model.
- Most attention-based model only considers region-level attention and discard the pixel-level saliency.
- Most of the models do not consider the spatial context information between different part-based features.

2.2.5 Others

Besides the four major models described above, there are still some other deep learning-based researches on ReID community. Even with the datasets like CUHK03, Market1501, and DukeMTMC-reID, the average numbers of image per person is still quite limited. The first work tries to enlarge the dataset using a generative adversarial network (GAN) and was introduced by [66]. They employed GAN to generate unlabeled samples and adopted a CNN to extract feature for representation learning. Then label smoothing regularization is used for outliers method.

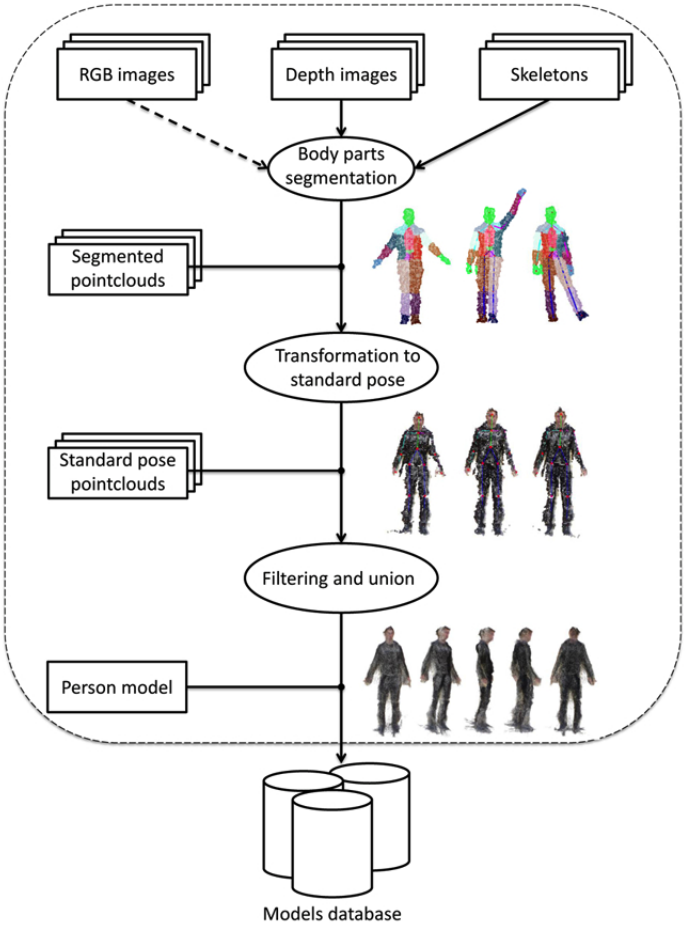


Figure 24: The working pipeline for ReID with RGBD data [35].

A camera style adaption model to adjust the CNN training was proposed in [69]. More precisely, CycleGAN is used to transfer the style of images captured by one camera to another. Given an image from camera No.1, the model can produce the image which looks like captured by camera No.2.

Unlike most of the works which are based on RGB image, it is noteworthy to mention a work which employs RGB-D data as input. A novel method for person ReID using RGB-D data was proposed in [35], their working pipeline can be shown as Figure 24. It took a RGB-D image as input, then performed a segmentation to obtain different parts of the human body. By using this information, they did 3D reconstruction and pose transformation resulting with a standard human 3D model. Then using the attributes computed from the 3D model to perform re-identification.

2.3 Available Software

To develop a general solution for (1) deep learning-based person re-identification and (2) device abstraction, from the implementation point of view, we have to survey the currently available software that may be useful for our solution.

2.3.1 Freenect and Freenect2

The most impressive depth camera to people nowadays may still be the Microsoft Kinect, even if it has been dropped by its own company now. It was the first consumptive depth camera in the market released in November 2010. Kinect was designed for Microsoft Xbox 360, a video game console. In order to let the game designer to fully make use of it, a corresponding closed source library (known as Microsoft Kinect Developer Toolkit) was also released to enable the programmability of the device.

Since the Microsoft Kinect Developer Toolkit is closed source and can only be used on Windows machines. A group of people from the community made an open-source driver for Kinect named Freenect to enable it works on Linux, MacOS and Windows. By using this Freenect driver, people can obtain the raw depth data from the device directly. A lot of researches based on depth data has been based on this solution. When Microsoft released Kinect v2 (the second version of Kinect), the community also came up with the driver Freenect2 with the same functionality.

2.3.2 OpenNI2

OpenNI or Open Natural Interaction library was originally created by PrimeSense which is a depth-sensing solution provider company. Kinect was using their technology to obtain the depth data from the sensors. After the acquisition of PrimeSense by Apple, they shut down the official website but the community (like Occipital and other former partners) is still keeping a forked version of OpenNI2 active as an open-source library for their product.

By using the OpenNI2 framework, we can perform the following operation using

the same unique APIs, if the hardware driver respects the OpenNI2 standard. Fortunately, both Freenect and Freenect2 have the option to build an OpenNI2-supported version of them:

- Voice and voice command recognition.
- Hand gestures.
- Body motion tracking.

2.3.3 OpenCV

OpenCV is a library of programming functions mainly for computer vision. It was originally created by Intel (for image processing) in 2000 and now lead by Itseez. The library is cross-platform and open source under the BSD license, widely supports most of the existing operating systems. The library was originally written in C, but since 2009 it primarily changed to C++. Nowadays, it also adds CUDA-based and OpenCL-based GPU calculation, machine learning and deep learning (TensorFlow, Torch/PyTorch and Caffe) support.

2.3.4 NiTE2

NiTE2 is a piece of middleware of OpenNI2 library. It has to work with OpenNI2 underneath and provides more powerful functionality than OpenNI2 does. It uses the same design philosophy of OpenNI2 which is only supposed to provide the infrastructure and leave all the other functionalities to the middleware. Unfortunately, NiTE2 is a closed source library provided by PrimeSense, it was written in C++ and only comes with the header files and the binary library. By using NiTE2, we can get the following data:

- Skeleton data of the tracked full human body.
- Gesture data of the tracked hand.

2.3.5 RealSense SDK

RealSense SDK is a cross-platform library from Intel for their own depth cameras. It allows the developer to access depth and color streaming and provide the basic camera parameters for calibration. Since the SDK is provided by the manufacturer directly which means they know everything about the hardware, it is more powerful than Freenect and Freenect2 as to Kinect. The SDK is written by C++ and hosted on GitHub. The community is still quite active and they still try to add more functionalities to the SDK (like support OpenNI standard, working with OpenCV library, provide more wrapper for other programming languages rather than C++).

2.3.6 CPython

Our research team proposed the OpenISS framework, which is designed to be written in C++ since most of its dependencies list above are in C++. But nowadays, most of the deep learning programs are written in Python, and for experiment and prototyping purposes, Python has a more efficient development environment. In order to invoke the deep learning model from OpenISS framework, we need something to connect C++ and Python. CPython is our desired bridge, it is the reference implementation of the Python programming language written in C and Python. It has a foreign function interface with support to several other programming languages and C is one of them. By making use of CPython we can exchange a class, a function, a variable or other data structure with the languages on two sides of the bridge.

2.3.7 TensorFlow

Since deep learning got a lot of attention recently, there is a variety of deep learning frameworks being developed. Figure 25 shows the popularity of most of the existing platforms. Among these, one of the most famous is Google's TensorFlow [8], an open-source library written in C++. It is a symbolic math library that provides various kinds of tensor operations for dataflow and differentiable programming. It uses a computational graph with respect to the chain rule to perform back-propagation

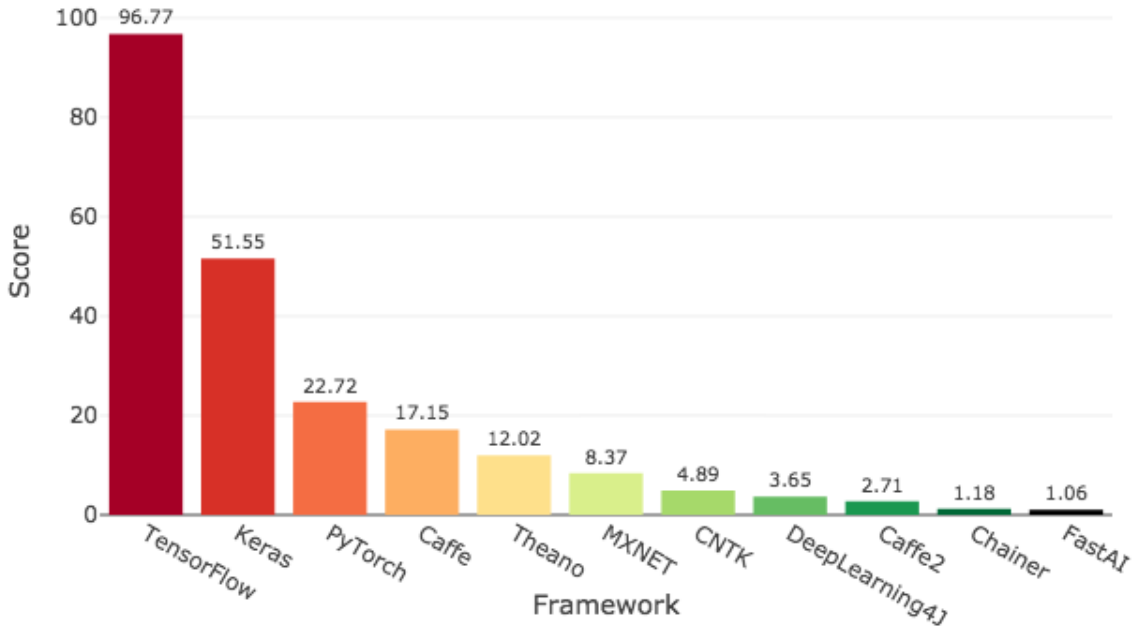


Figure 25: Power score of the most deep learning frameworks in 2018 [6].

which is the core concept for updating the parameters (or we can say training). Also, TensorFlow can encapsulate the hardware differences and support training on multiple GPUs if they are available without any code modification. It is becoming more and more popular in the industry and production environment.

While TensorFlow is popular in the industry, another framework named Pytorch gets more attention among academic users and researchers. In the ReID research community, most of the code is based on Pytorch. There is even no baseline model implemented in TensorFlow and Keras for the ReID task. In contrast, since YOLO is widely used in the industry, there are already numerous existing implementations of YOLO in TensorFlow. In order to keep our implementation consistent in one framework and reduce the overhead for translating data from one to another, we chose TensorFlow as our deep learning platform. By employing it, our research actually fills a gap, as no existing solution for ReID task is currently based on TensorFlow.

2.3.8 Keras

Keras [11] is a high-level open-source library written in Python, firstly developed by Francois Chollet, which can take TensorFlow, Theano and some other frameworks as its back-end. It doesn't provide the mathematics operation implementation as they were left to the backend but a human-friendly APIs which can allow you to prototype your conceptual neural network (or other machine learning architecture) easily and experiment with different deep learning frameworks. TensorFlow adopted Keras into its core and announced it as the official high-level APIs in 2017 and more support has been added to Keras since TensorFlow 2.0 which was released in 2019.

2.4 Summary

In this chapter, we gave an extensive review to the most common deep learning-based methods in both object detection and person re-identification domains. For the object detection problem, we started from the seminal work R-CNN and stated the key contributions of each existing approach and compared them with similar methods if comparable. Due to our real-time limitation from the requirement, we are restricted to use one-stage method. Precisely, we select YOLO v3 as our detector, because its inference time is by a large margin better than all the others and the community already has a lot of existing resources for it. For person re-identification problem, we introduced a total of five different models and explained their network architectures and loss functions. Jointly considering the trade-off between implementation complexity and the model's performance, we decided to use a combination of the identification model and the triplet model. In the last part of this chapter, we listed all the available software that may be employed as dependencies in our work. In the next chapter, we will start to present our solution.

Chapter 3

Framework Design

In this chapter, we are going to describe the design of our proposed solution which can fulfill the requirements stated in Chapter 1. We will first explain why we choose the framework design solution. Then we outline the architecture of our solution. Finally, we detail each component for both the core and specialized frameworks.

3.1 Why Framework Solution?

In the context of software engineering and computer programming, a software framework is an abstraction in which software providing generic functionality can be selectively adapted by additional user-written code, thus providing application-specific software [2]. According to [38], a software framework consists of two kinds of components:

- Frozen spots, within a framework, define the overall architecture of a software system, that is to say its basic components and the relationships between them. These remain unchanged in any instantiation of the specialized framework.
- Hot spots, within a framework, represent those parts where the programmers using the framework write their own code to add specific functionalities based on their own need.

There are three key distinguishing features that make a framework different from normal software libraries:

Inversion of control: In a framework, the program's flow of control, unlike in libraries or applications, is not dictated by the caller but the framework. In our case, we want to address the person ReID problem (specified by **FR4** and **FR5**). It can be divided into two subproblems: person detection and person retrieval, which the output from the former will become the input of the latter. From the user's point of view, they are totally not interested in the intermediate steps but only want the final result. So the user doesn't have the knowledge and they even don't want to know how the data flow between these two subproblems and how the the device can obtain the raw data at the beginning as well. In this case, the flow of control of the program should actually be done by the framework since for a specific task, the workflow should be deterministic and the user just tells what they want to do but not how they do it. Under such consideration, having the program's flow predictable and controllable is extremely important for us.

Non-modifiable framework code: The framework's core code (i.e. frozen spots), in general, is not supposed to be modified, but should accept user-implemented extensions (i.e. hot spots). In other words, users can extend the framework, but cannot change its code. The reason why the frozen spots cannot be changed is that the framework is responsible for controlling the flow of the program, without knowing what kind of application the framework will be used for. The control flow is defined by these frozen spots, if they are always changed then there is no way to achieve inversion of control.

Extensibility: A user can extend the framework, usually by selectively overriding or adding specialized code to provide specific functionality. As mentioned in Section 1.5.6, extensibility is one of our non-functional requirements. Since we want to support various kinds of cameras (specified by **FR1** and **FR2**) and as introduced in Chapter 2 the algorithms for both person detection and person retrieval are diverse. It is possible that later on we may want to perform comparisons among these algorithms. With such extensible design available, we can save integration efforts, making our

solution more valuable.

From the discussion above, we see that the key features of a software framework can perfectly fit to the demand of our solution. Furthermore, if we think of our main person re-identification scenario in an abstract way, what our solution needs actually is to enable the definition of a pipeline which works in the way described below:

1. Information is gathered by devices which can be diverse.

2. Information is transformed/extracted using filters (person detector and person recognizer), which can also be diverse but must be made abstract so that they can easily interoperate.

3. Every device and filtering algorithm comes with their own data model.

Interoperability comes through the definition of an abstract data structure that is exchanged between the abstractly defined filters. This way, various devices can be used in conjunction with various filtering algorithms to create an application. This is, in fact, a classic example of a problem solvable using a framework approach. So we decide to plan our solution in a framework manner. In our design, we have the core framework as the frozen spots which defines the infrastructure such as device, cross-language calling mechanism and viewer. Then for each specific purpose, we have a specialized framework which, under the core framework umbrella, provides specific functionalities for various application developers like person ReID, skeleton tracking, gesture tracking and facial expression recognition.

3.2 Core Framework Design

In a very high-level description, we design our framework (named OpenISS) as consisting of a total of eight components as shown in Figure 26. Each box in green represents a frozen spot of the core framework and each box in yellow means a set of frozen spots which can combine with the core to form a specialized framework.

For the five core frozen spots, their functionalities are designed as follows:

- Device module: It provides an abstraction of various devices which can be used by any application that needs cameras as input.

- Cross-language module: It provides the ability that from C++ we can invoke algorithms or models implemented in Python which can help us to make sure of most of the existing resources available from the community.

- Pipeline module: It serves as an executor of the framework providing the flow of control for a variety of tasks.

- Common data structures module: It provides our own framework data structures which were adapted from other low-level libraries or software enabling us to perform our own algorithm independently, or to provide interoperability between otherwise incompatible algorithms.

- Viewer module: It provides visualization abstraction of the framework which can be used by any application that needs to display a result.

For the three specialized frameworks, they are designed as:

- Tracker specialized frameworks: It provides the abstraction of a tracker, in the context of motion capture, computer vision or image processing, which takes a frame from a sequence of images and a set of given pixels as input. Then for all the remaining frames in the sequence, it keeps tracking the location of a same set of pixels that have the same meaning.

- Detector specialized frameworks: It provides the abstraction of a detector, in the context of computer vision, which takes an image and an object class list (e.g. cat, dog, person, car) as input then output a bounding box for each detected object instance within the image with respect to a given object list.

- Recognizer specialized frameworks: It provides the abstraction of a recognizer, in the context of computer vision, which takes an image (can be a pedestrian or a face image) or video (a video with facial expression or gesture) as input

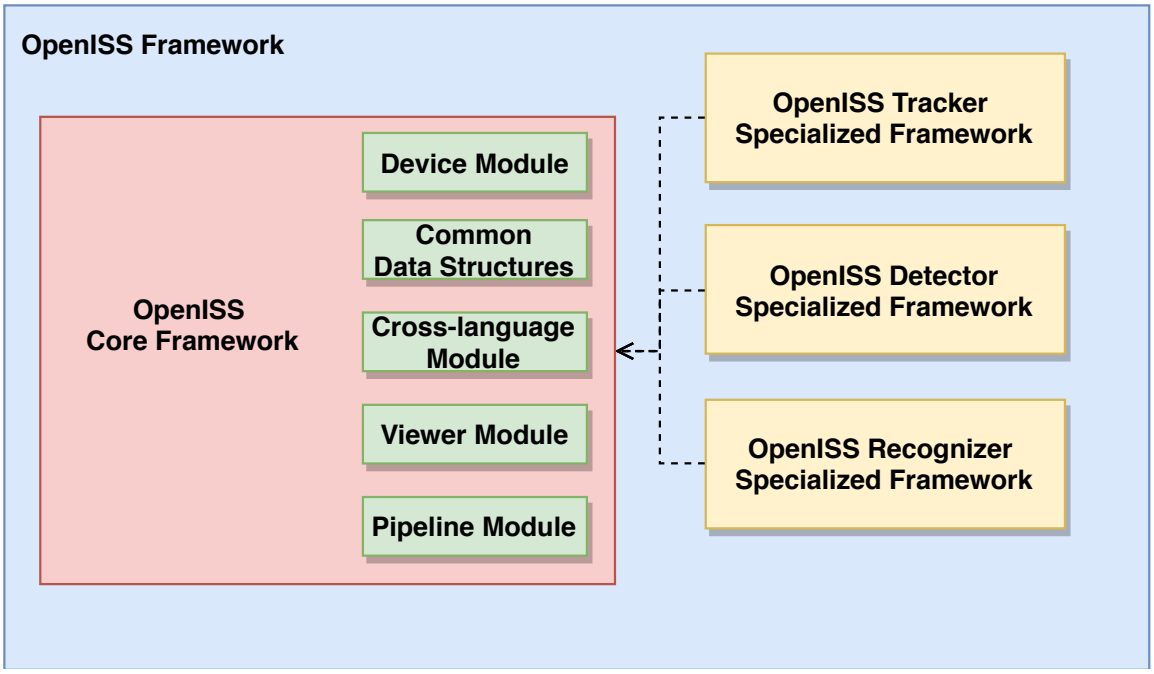


Figure 26: Core components of OpenISS framework, each box in green means the core framework's frozen spot and each box in yellow represents a set of frozen spot of each specialized framework.

then output whether this image or video have been seen before (within the pre-defined database or by any other means).

In the following paragraphs, we are going to explain the design of each module within the core framework.

3.2.1 Device Module

The device module is one of the most essential modules in the whole OpenISS framework because it is the lowest layer from our framework's point of view. It is designed directly on top of the hardware drivers from various device manufacturers. The layer architecture is shown as Figure 37 which will be explained later in Section 4.1.2. Our design goal for this module is that we would like to block the physical differences of cameras accessing them via a set of common APIs. Also, the design should allow us to add support to more kinds of cameras easily without changing the frozen spots themselves.

With those requirements in mind, we found that one possible solution is to make use of the polymorphism feature and dynamic dispatch mechanism, accessing the subclass's method via a reference of its superclass. So what we need to do will be just to come up with an abstraction that can be applied to most of the common devices. After overall consideration, our design for the device module is depicted as Figure 27. Since it is a core module, we are going to explain some of these important abstract methods defined in the `OIDevice` class:

- `rawDevice`: Since our device model needs to depend on the hardware driver, sometimes we may want to access the original device object created by the driver, this method is designed for it.
- `init`: This method contains the logic used to initialize the device.
- `open`: This method is used for opening the device, it should be called after `init` method.
- `close`: This method is used for closing the device logically, most of the time we will release the resources which are not needed anymore.
- `enable`: This method is a shortcut for the following three specific enable methods.
- `enableColor`: This method tells the device to enable the color data stream.
- `enableDepth`: This method tells the device to enable the depth data stream.
- `enableRegistered`: This method tells the device to enable the registered data (infrared image data, a.k.a. IR data) stream.
- `getIntrinsic` and `getExtrinsic`: These two methods are designed to obtain the intrinsic or extrinsic matrix respectively of the device if they are provided by the hardware driver.

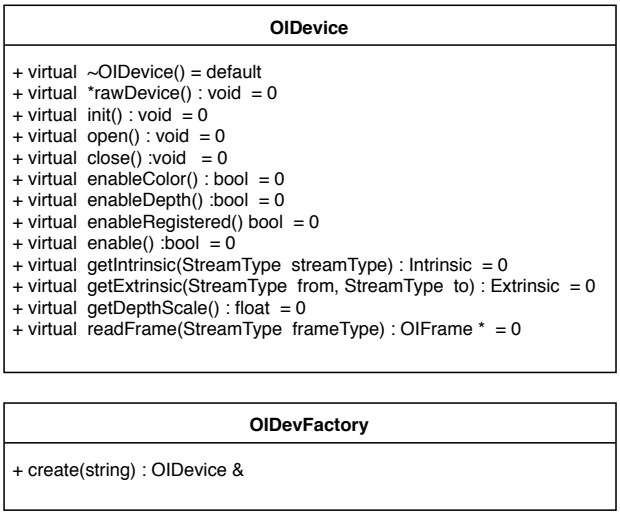


Figure 27: Design of device module within OpenISS core framework.

- `getDepthScale`: This method is used to get a depth scale value. It can be used to multiply the data value from the depth image to get the distance in meter or centimeter unit.

- `readFrame(StreamType type)`: This method is the most important one, it is used to get a frame of data respect to the stream type (color, depth or IR) specified by the parameter.

3.2.2 Cross-Language Module

As mentioned in Section 2.1 and Section 2.2, the research community of both object detection and person retrieval have been dominated by deep learning approaches since 2012. Nowadays most of the available deep learning frameworks provide Python APIs for convenient purpose. Thus, most of the existing deep learning models were written in Python. But unfortunately, our OpenISS framework itself is written in C/C++ (the reason will be explained later in Section 4.1). Obviously, there is a gap between our framework and many of the existing solutions. To fill this gap, we designed a cross-language module shown as Figure 28 which enables the user to separate deep learning oriented program development implemented in Python from the normal framework related programming task which is in C++. It provides an encapsulated internal

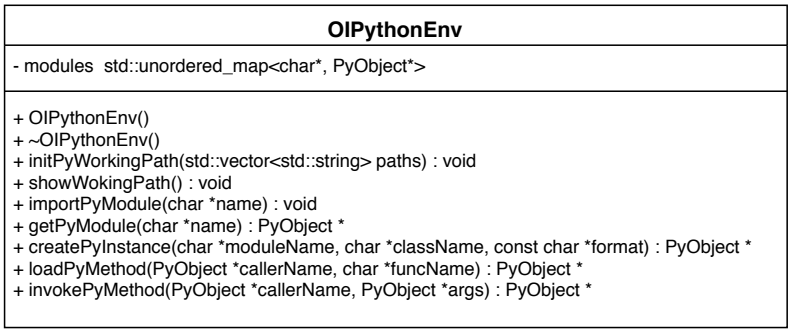


Figure 28: Design of cross-language module within OpenISS core framework.

API to invoke Python model from C/C++ by employing CPython introduced in Section 2.3.6. It can not only help us to decouple the framework's functionalities but also make good use of the existing community resources.

Even it is named the cross-language module, currently, we are not aiming to provide a generic mechanism to support communication between C++ (our framework's language) to any other language other than Python. The reason is that there are many kinds of programming languages. It is a notoriously difficult problem to provide an abstraction among all of them and there is no existing solution in the community. Also, we doubt that if it is really worthwhile to put time and efforts to create a generic cross-language mechanism that applies to all languages.

The `OIPyhonEnv` is not an abstract class. The reason why we put it under the core framework is that it actually serves as an infrastructure even though the classes which will invoke the Python code don't need to inherit from it but they have to use it as a dependency. The `OIPythonEnv` class is used to encapsulate a Python script, four core functions:

- `initPyWorkingPath`, it is used to add a given path to the Python interpreter as a working path (Python will search the requested module under all the working paths).
- `createPyInstance`, it is used to create an object of a given class which is visible within the encapsulated Python script.
- `loadPyMethod`, it is used to load (without executing) the handler of a specified

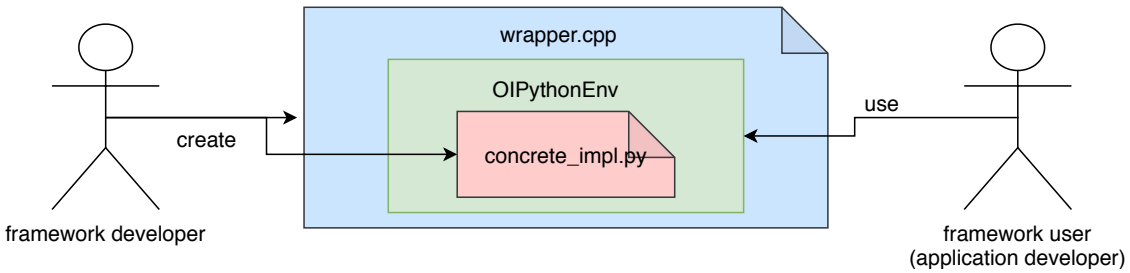


Figure 29: Usage scenario of the cross-language module of OpenISS core framework.

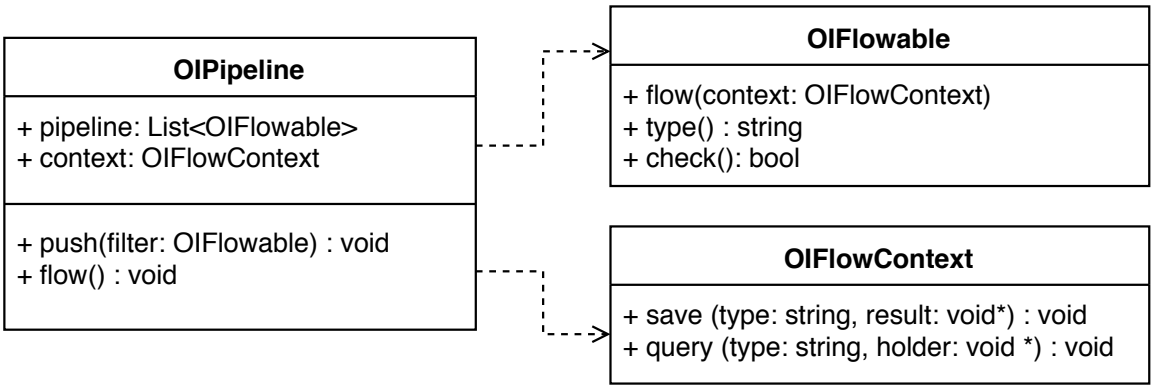


Figure 30: The design of the pipeline module in the core framework.

method which is visible from that file.

- `invokePythonMethod`, it is used to invoke a loaded function using its handler.

The design and usage philosophy of this module can be illustrated by Figure 29. Each instance of `OIPythonEnv` can be used to represent a Python file (the box in green). A framework developer trained a deep learning model in Python (the red box) and would like to expose some of the functionalities of it to the framework users. Then they can write a C/C++ wrapper (the blue box) for it which contains a member variable of type `OIPythonEnv` instantiated by the names of classes and functions would like to expose. The framework users then can use the Python code via our framework without knowing anything about Python under the hood.

3.2.3 Pipeline Module

The pipeline module, as its name implies, provides a pipeline execution mechanism for the framework users. It allows the user to chain multiple filters together that the

former's output will become the latter's input. When all these filters finish execution, we can achieve some specific goal, for example, our final goal, tracking person across multiple cameras (detail will be explained in Chapter 5).

The design of the module can be described by the UML diagram shown as Figure 30. The `OIPipeline` class contains a list of filters and a reference with type `OIFlowContext` which is an abstract class used to define the save and query behaviors of the temporary result generated by the intermediate filters. The `push` method is used to add a concrete filter into the pipeline while the `flow` method is the switch to trigger the pipeline execution process. The component which would like to serve as a filter must agree with the `OIFlowable` contract. There are three functions defined within this interface:

- `flow`: It takes a reference of `OIFlowContext` as parameter, basically what this function does is calling the real logic method. For example, the `flow` function will call the `readFrame` function within `OIDevice`, the `detect` function within `OIDetector` and the `predict` function within `OIRecognizer`.
- `type`: It just returns the type of the filter itself, we may need it to differentiate some operations for various filters.
- `check`: It usually gets called before the `flow` method, we perform necessary checking step here to ensure all the needed data is available before `flow` gets executed.

The control flow of the pipeline module can be expressed by Algorithm 1 and the workflow can be visualized by Figure 31. With the `OIPipeline` class definition in mind, there are two fields: `pipeline` which is a list of filters and `context` which is actually a temporary data holder. What we eventually do is that we loop over all the filters within the list and invoke their `flow` method which is implemented in the concrete classes and store the result in the concrete implementation of the `OIFlowContext` class. Inside the abstract `OIFlowContext` class, we define two methods: `query(name:string, data:void*)`

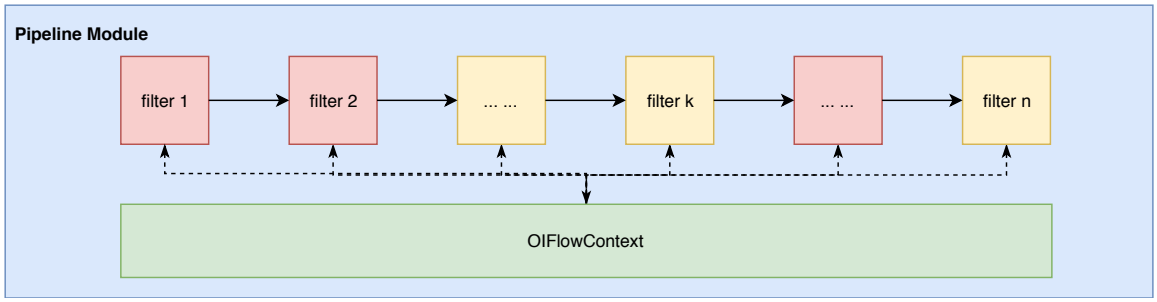


Figure 31: Design of the pipeline module in the core framework.

and `save(name:string, data:void*)`. The method `query` is used to lookup the needed input which output by the preceding of current executing filter while the `save` function is used to save the temporary result generated by the current filter. If there is more than one input or output from the current state, we just need to call these two functions multiple times so we don't limit ourselves to the number of input and output of a filter.

Algorithm 1: The `flow` function within `OIPipeline` class

```
1 foreach filter in pipeline do
2     canFlow = filter.check(context)
3     if canFlow then
4         result = filter.flow(context)
5         context.save(filter.type(), result)
```

As you may notice the `OIPipeline` class only has a `push` method to allow the user to add filters but it doesn't provide any removal interface for the existing filters which means each `OIPipeline` instance is immutable. In other words, once a pipeline has been defined, you cannot change its internal structure. We design in such a way under the consideration that each pipeline instance is used for a specific task. If you have more than one task then you will need to reassemble a new pipeline instance and create another concrete `OIFlowContext` object but you can reuse the same filter object if you want. Also, all the filters residing in the pipeline are chained linearly. But when you executing them, a conditional operation can be achieved by the `check` method since it is the predecessor of the `flow` method.

3.2.4 Common Data Structures Module

The common data structures, in OpenISS core framework, means these data structures that may be used by other modules within the core or across multiple specialized frameworks. `OIFrame` is one of the most significant data structures of our framework which is designed to represent the data captured by an input device at a certain point in time. It will flow between the core and specialized framework or even between several different specialized frameworks. The design of `OIFrame` can be illustrated as Figure 32. `OIFrame` is the highest level of abstraction that provides the fundamental information of a frame. It has two subclasses named `OIAbstractDataFrame` and `ICvImplFrame`, the former is still an abstract class representing a frame contains data and the latter is a concrete class describing a frame provided to the viewer.

3.2.5 Viewer Module

The responsibility of the viewer module is straightforward. As its name implies, it is used for displaying the data for visualization purposes. Our design for the viewer module is simple, as shown in Figure 33, `OIViewer` is an abstract class which contains a variable and a method. The variable `name` is a label used to differentiate from various displaying windows since the user may want to show more than one image, for example, show both color and depth image at the same time. The method `show` defines the behaviour for drawing the window. Currently, our framework only has only one concrete implementation which is `OIOpenCVViewer` based on the OpenCV library.

3.3 Specialized Framework Design

The specialized framework is designed for solving a class of specific problems, it provides a set of unified APIs to the framework users just like other modules within the core framework but also defines the problem-specific data structures and common

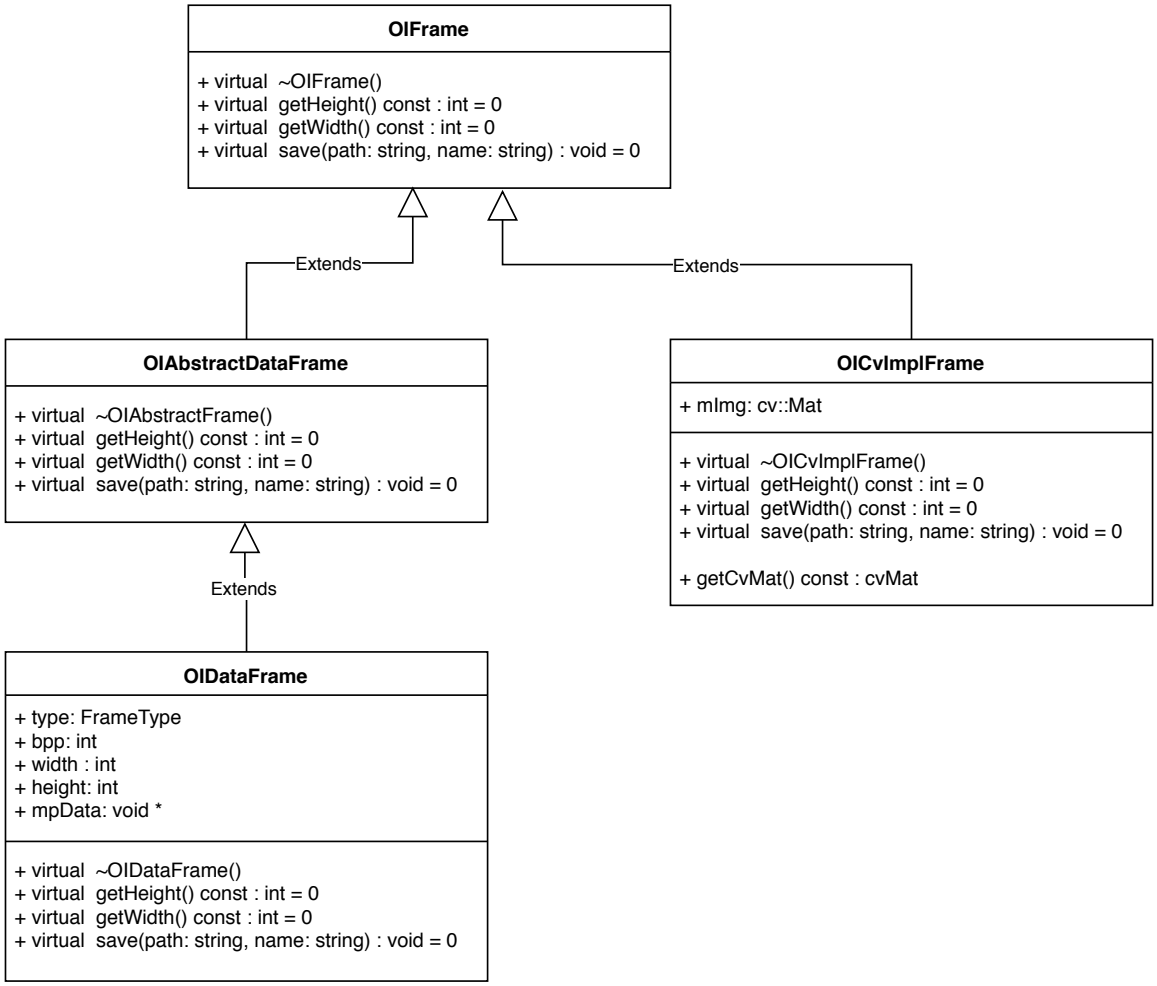


Figure 32: Design of `OIFrame` inside common data structure within the core of OpenISS framework.

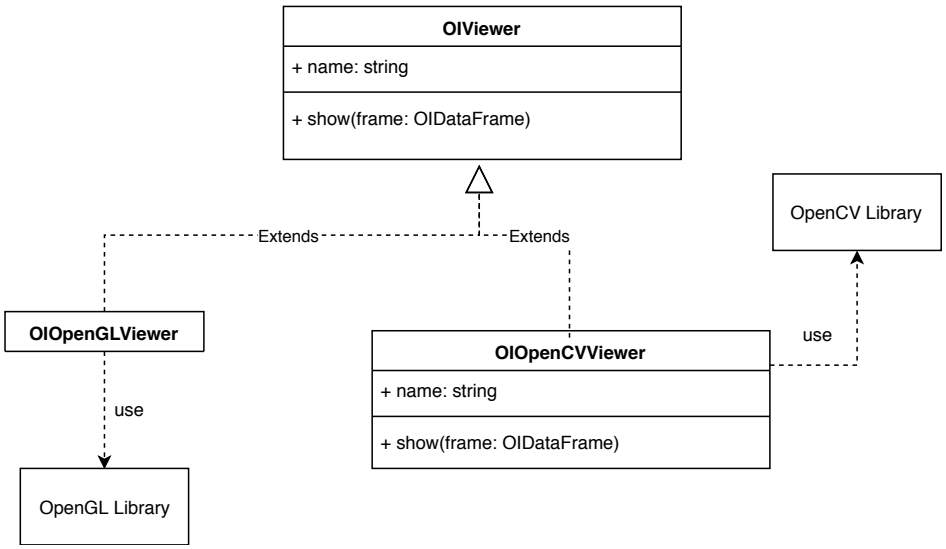


Figure 33: Design of viewer module within the core of OpenISS framework.

methods for the specifica application problems it is designed for. Recap from the previous chapter, a specialized framework is formed by the core and its problem specific module. The relationship between the core the specialized framework is that the specialized framework will take the core as its dependency (a.k.a. infrastructure) then defines its own functionality and exposed APIs.

When we were working on the specialized framework, we believe that a large problem can be divided into several subproblems. Applying the divide and conquer strategy, we design a specialized framework to address each of the subproblem and all these results put together can solve the origin complex problem. If we want to link them then the linking operation between the core and specialized framework or between two specialized frameworks is done by the pipeline module which has already been explained in Section 3.2.3.

In this thesis, we proposed three specialized frameworks, shown as the rectangle in yellow in Figure 26. A combination of two of them (detector and recognizer) aiming re-identify the same person across multiple cameras in order to solve the limitation we mentioned in Section 1.2. And the other one (tracker) is used for skeleton tracking. In this section, we will describe the architecture of each specialized framework.

3.3.1 Tracker Specialized Framework Design

In Section 1.5, we explained the need for skeleton tracking. Also, concurrently with this thesis being written, there is another work happening which aims at providing the functionality of gesture tracking. So it is necessary for us to abstract the common methods of skeleton tracker, gesture tracker and a variety of other possible trackers. Recall the problem a tracker attempt to address is that for a given sequence of frames and a target, it is expected to locate the target for each of the remaining frames.

In order to achieve that, we design the tracker specialized framework as shown in Figure 34. `OITrackerFactory` is responsible for instantiating the concrete tracker object and tracker frame object. `OITracker` is the abstract class of the tracker, it defines three basic methods, `startTracking` for starting tracking, `stopTracking` for stopping tracking and `readFrame` for reading data from the input source and

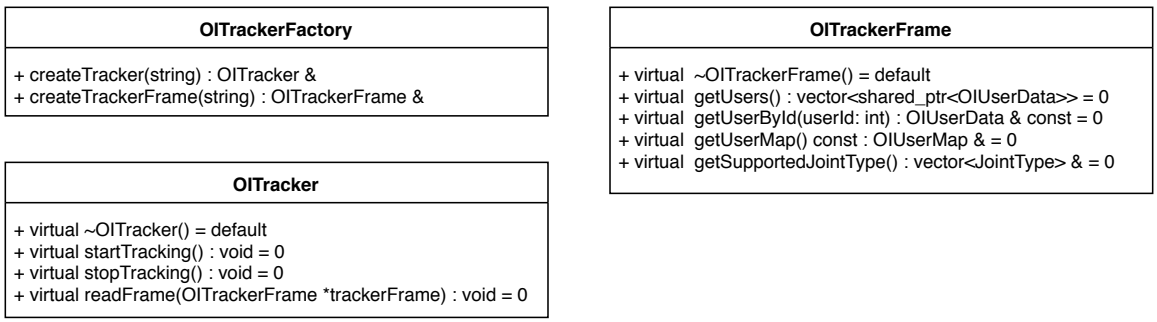


Figure 34: Design of the OpenISS tracker specialized framework.

apply tracking algorithm on it, which is where the magic should be implemented in a subclass. The parameter, `OITrackerFrame`, acts as a data container, the result of the tracking will be placed within this class and it will be updated for each frame.

3.3.2 Detector Specialized Framework Design

As mentioned in Section 1.3, the ReID problem can be divided into two parts and one of them is object detection, so it is necessary to design a specialized framework for it. Since we are designing a framework, in order to maintain its abstractness and extensibility, we need to extract the common parts of different kinds of detector then provide an abstraction of them. Recall the definition of object detection we gave in Section 1.3, given an image I and a list of objects C, the output will be a list of bounding box B which contains the instances of objects listed in C.

With such consideration, we design the specialized framework as shown in Figure 35. The `OIDetector` is the abstract class which will be exposed to the user. Depending on the user's specific demand, they can use either our pre-defined hot spot or create their own hot spot to perform different kinds of detection algorithm. The frozen spot itself `OIDetector` has a member variable `classList` contains the name of the supported classes of a detector and a method named `detect` which take an instance (hot spot) of the OpenISS common data structure `OIFrame` as input and output a list of bounding boxes with the type `OIBBox`. Since the detector may be used to detect any kind of objects, the shape of the bounding box may be different. `OIBBox` is the abstract class of the result which currently has two pre-defined hot spots

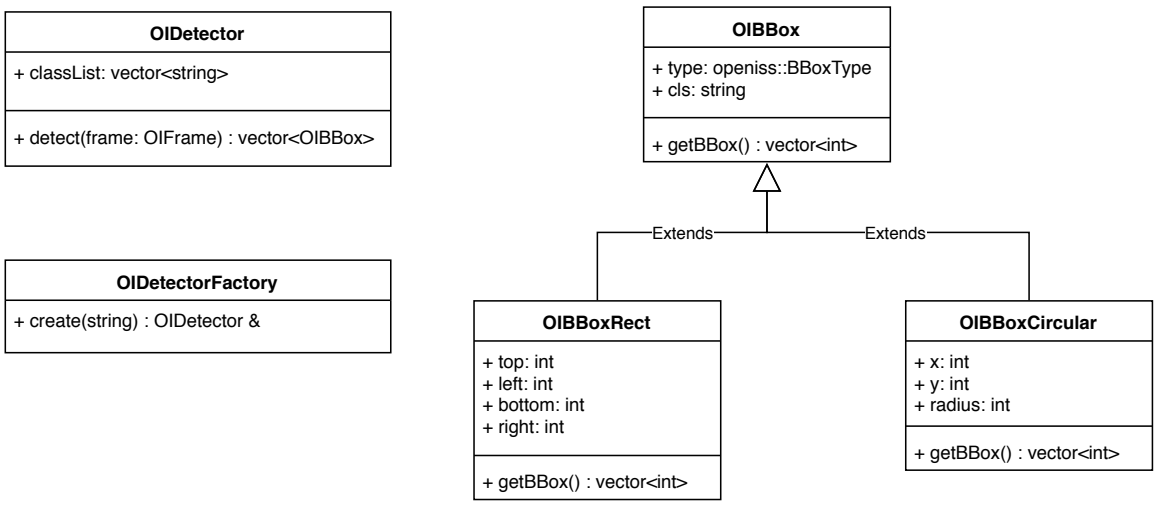


Figure 35: Design of the OpenISS detector specialized framework.

`OIBBoxRect` and `OIBBoxCircular` representing the rectangular and circular shape of bounding box respectively. If any other shape is needed, the user can create their own subclass inherited from the abstract class. Finally, we adapt the factory pattern just like the device module to encapsulate the creation process of different concrete detectors so that the user just needs to specify the name of the detector and pass it to a factory's function named `create`. It will return a reference with type `OIDetector` wrapping the desired concrete detector.

3.3.3 Recognizer Specialized Framework Design

As mentioned in Section 1.3, the ReID task can be divided into two parts and we already explained object detection in the previous section. The other part is object retrieval. In other words, you are given a set of gallery images with their identities in advance. Then for a never seen query image, the recognizer is expected to tell its identity among the gallery images.

Following the same pattern as the detector specialized framework, we design the recognizer specialized framework shown as Figure 36. Because we need to compare an item with all the items within a database, the first step, of course, is that we need to create a database. According to our definition, a database is a hashmap where the key is the identity string and the value is a descriptor represented by the

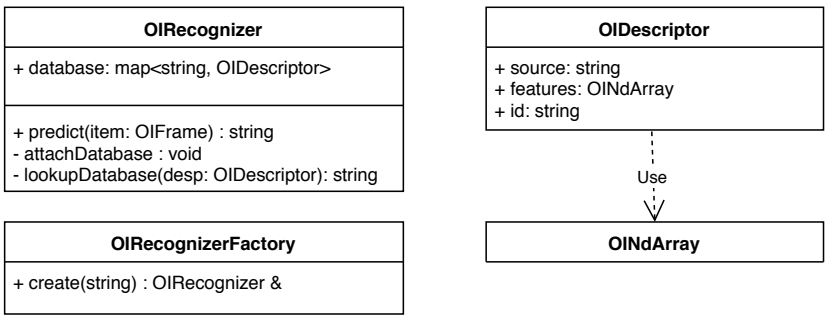


Figure 36: Design of the OpenISS recognizer specialized framework.

class `OIDescriptor` computed from the source (e.g. an image or a video). Inside the `OIDescriptor` class, the variable `srcPath` points to the location of the source represented by this descriptor. The variable `features` means the feature vector of the content of the source and the variable `id` means the identity label. Feature is one of the most significant parts for recognition, the core idea of recognition is trying to find a way that can measure the distance between two features accurately and effectively. Since these features are high dimensional vectors, it is hard to imagine and tell what kinds of metrics can perform better. Feature is represented by the class `OINdArray` within the recognizer specialized framework which encapsulates an N-dimensional array. Since `OIRecognizerFactory` works exactly the same as its sibling within the detector framework, we will omit the explanation for it here. Finally, we reach the frozen spot of the recognizer representing the class named `OIRecognizer`. It has a variable name `database` which holds all the pre-defined identities and three functions. The method `attachDatabase` is easy to understand, as its name implies, it is used to attach the database to the recognizer. The method `lookupDatabase` defines how to look up the possible result with a given descriptor in hand in the database. The method `predict` is one that the user needs to invoke, it takes one parameter typed `OIFrame` which is the image that contains the targeted item.

3.4 Summary

In this chapter, we introduced the design of our framework. We began with establishing a rationale to explain why we choose the framework design solution. This

was followed by a description of the structure of the framework. We logically split the whole solution into two parts, one serving as infrastructure called core framework and the other one is responsible for specific tasks called specialized frameworks, in our design, there can be multiple independent specialized frameworks. Then we explained the design of each module (a.k.a. the frozen spots) within both the core and specialized frameworks. In the next chapter, we are going to describe how we create a framework instance to fulfill the requirements and scenarios proposed in Section 1.5.

Chapter 4

Framework Instantiation

In this chapter, we are going to describe in detail how we create an instance of the framework we proposed in Chapter 3. We start with the implementation decisions by explaining why we choose the selected techniques followed by the project structure used during the implementation time. Then for each frozen spot introduced in Section 3.2 and Section 3.3, we describe our instantiation process and the essential implementation detail.

4.1 Implementation Decision

In Section 3.1, we stated why we chose a framework design approach. For a programming problem, if we just want to solve it and only it specifically, what we need may be just a piece of regular software. But if we want to provide a generic solution that may apply to different problems, it requires extra work and may become more complex. The framework approach aims at providing a general solution, so for sure its complexity is higher and its development will become more difficult than the usual approach. Under such a situation, it is important to make decisions that can enable us to work in an efficient and productive manner. For implementing a framework, what kind of programming language we are going to use, how to compile the whole framework with its dependencies and build executable software and how to manage the whole project efficiently and clearly are all significant points that we have to address, which we are discussing in this chapter.

4.1.1 Programming Language and Compilation Tool

In Section 2.3, a list of currently available software has been surveyed. From that list we found most of them were written in C++, based on this fact, to make good used of the existing resources, we determine to use C++ to develop our framework as well. We can benefit from using C++ in the following aspects:

- Utilizes the existing software maximally.
- Obtains faster speed than most of the other programming languages since it is closer to the hardware.
- Relatively easier to communicate with other languages since a many programming languages were themselves written in C/C++.

With these advantages, C++ also has it own limitations. For example, there is no easy way to manage dependencies when the project becomes complicated. To address this problem, CMake was selected, which is an open-source cross-platform building tool. By using it, we can build our framework in Linux, MacOS or any other *nix-based platform with the same command. Also, with the support of CMake, the end-users can enable the building process partially which means they need only to compile and build the framework based on their specific needs.

4.1.2 Framework Layers and Project Structure

In the context of software engineering, a system can be partitioned using the concept of software layers. Software layers are where each "layer" of a system deals with a certain function of a system which, usually, gets more and more detailed as you burrow down into the layer stack. In our implementation, to separate the responsibility of different modules and specialized frameworks, we proposed a three-layer software model shown as Figure 37 and assign components into corresponding layers according to their functionalities, the description of each layer can be found in Table 1. Within such a logically stated separation, in our implemented code base, we also need to

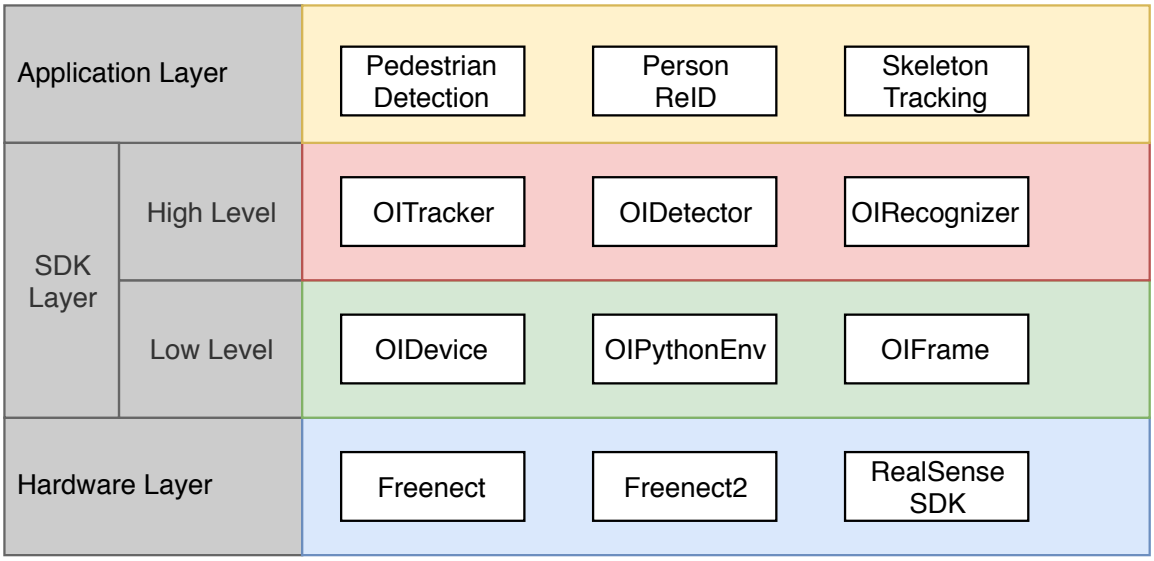


Figure 37: Three-layer model for OpenISS framework

Layer's Name	Description
Application Layer	Provides sample usages of the framework for the end-users (application developers)
SDK Layer (high level)	Provides encapsulated functionalities to end-users, hide the complexity of the implementation
SDK Layer (low level)	Provides atomic APIs to the end-users which enable them to create custom functionalities based on their own demands
Hardware Layer	Provides encapsulation of the hardware, which usually the application developers will not interested in

Table 1: Description of the responsibility of different layers

Directory Name	Description
`src`	Contains all the source code of the framework itself
`samples`	Contains all the source code of the application level samples
`modules`	Contains custom cmake modules for building the framework
`python`	Contains all the python modules using by the framework
`script`	Contains all necessary scripts (e.g. download datasets, configure environment)

Table 2: Directory structure and their functionalities.

structure our code clear and maintain the modularity of our design. For this purpose, we employ a project directory structure shown as Table 2.

4.2 Framework Instantiation Overview

As discussed at the beginning of Section 3.3, the relationship between the core and specialized framework is that each specialized framework takes the needed modules in the core as dependencies. In this section, we would like to give an overview regarding how the instances we create will interact with the core and their corresponding specialized framework.

For the device instances, of course, it will have interaction with the device module within the core. Precisely, the device instances will have to inherit the `OIDevice` class. Also, in order to allow the framework to control the flow of execution, all the devices instance will have to implement the `OIFlowable` interface. For the tracker instance, two instances classes will have to inherit the `OITracker` and the `OITrackerFrame` respectively. For detector instance and recognizer instances, since our implementation are based on deep learning approaches and the programs are written in Python. First we have to inherit from the `OIDetector` and `OIRecognizer` respectively, then take the `OIPythonEnv` as dependency. Finally, like the device module, they also have to implement the `OIFlowable` interface to transfer the control power to the core framework.

In the following section, we are going to explain in detail the instantiation process of the essential device module within the core as well as each of the specialized frameworks.

4.3 Core and Specialized Framework Instantiation

In this section, we describe how we create hot spots for those frozen spots introduced in Section 3.2 and Section 3.3, the augmented architecture diagram shown as Figure 38.

For the device module in the core framework, we describe how we instantiate it to support three different kinds of depth cameras. For the detector specialized framework, we describe what kind of modification we made and how we train the

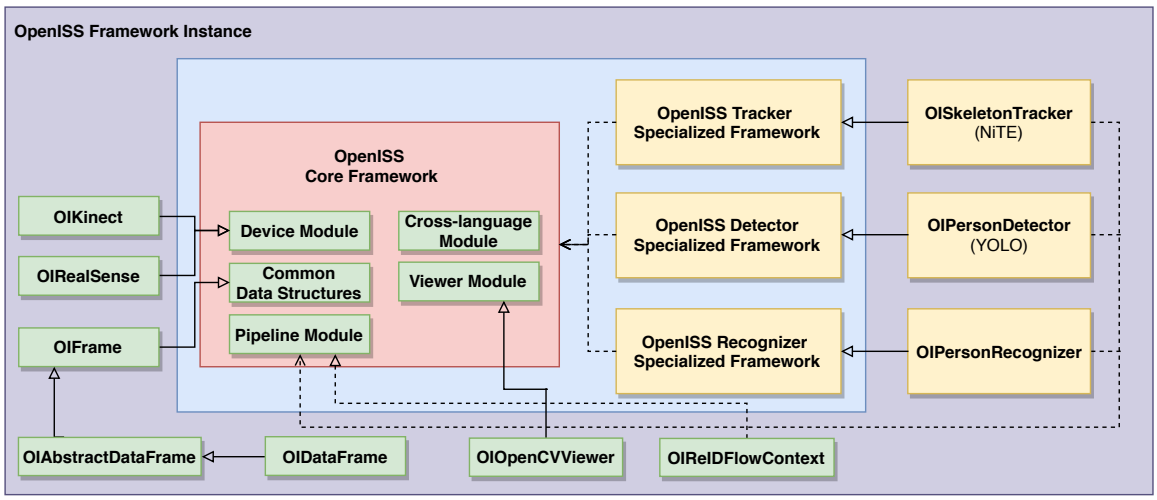


Figure 38: Implemented OpenISS framework instance.

YOLO model introduced in Section 2.1.2.1 and integrate it into our solution as a detector framework instance. For the recognizer specialized framework, we describe how we train a person ReID network which is a combination of the identification model and triplet model explain in Section 2.2.1 and Section 2.2.3. For the tracker specialized framework, we describe how we instantiate it with the existing NiTE2 middleware introduced in Section 2.3.4 implementation.

4.3.1 Device Module Instantiation

In Section 3.2.1, we described the design of the device module within the core framework. To enable our framework to work with real cameras, we have to instantiate the framework by creating a hot spot for the device's frozen spot. Currently, we plan to support three kinds of devices: Kinect v1, Kinect v2 and RealSense D435. These devices come from different manufacturers which are supported by their own hardware drivers. To combine them within a same set of APIs, the idea can be illustrated by Figure 39. For each kind of device, we create a concrete class, in our case, will be the class `OIKinect` and `OIRealSense` to wrap the concrete implementation of the defined functions within the `OIDevice` abstract class explained in Section 3.2.1. Then because of the factory design pattern, what the framework users get from the factory is a reference with type `OIDevice`, so they can obtain the ability accessing

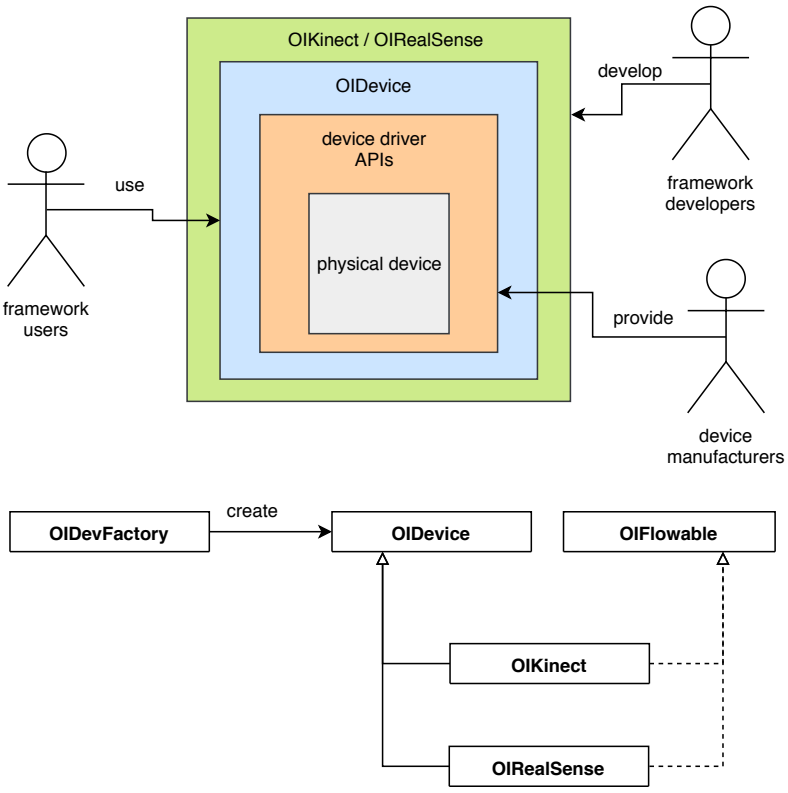


Figure 39: Instantiation of the device module of the core framework.

the same function definition but with different implementation from the dynamic dispatch mechanism.

4.3.2 Detector Specialized Framework Instantiation

In Section 3.3.2, we proposed the design of frozen spot for a detector specialized framework in general. To achieve person detection to fulfill our requirement, based on the detector frozen spot, we create a deep learning-based person detector hot spot. The overall idea shown as Figure 40.

We develop a concrete class `OIPedestrianDetector` which extends the abstract class `OIDetector`, serving as a wrapper of the concrete Python implementation of the YOLO detector. The communication required here between C++ and Python is enabled by the cross-language module introduced in Section 3.2.2 from the core framework. From the users aspect, they don't need to worry about how the framework works with various other detectors, but only need to know the usage of them. For the

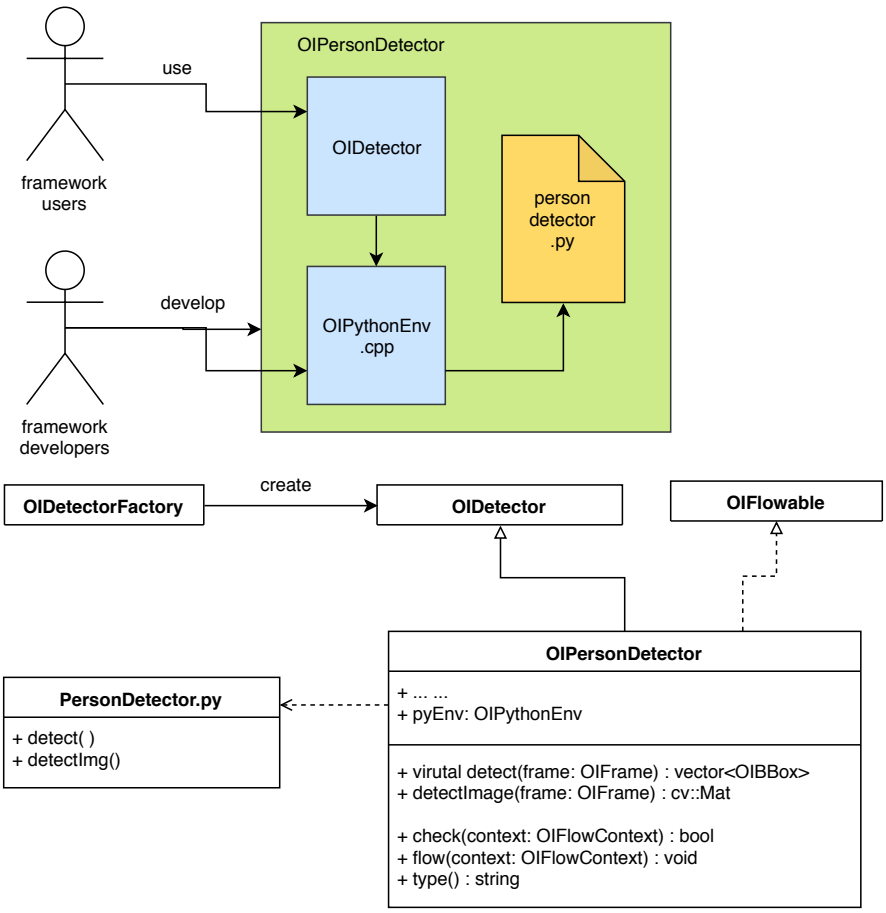


Figure 40: Instantiation of the detector specialized framework.

specialized framework developers, with such design, they don't need to know how the cross-language works as well, but passing the requested methods or classes name to the `OIPythonEnv`, the cross-language module will handle it for you. It simplifies the required impementation effectively by improving the effectiveness of the development process of both the applications and framework itself.

We previously described the big picture of the detector specialized framework. In the following paragraph, we will explain in detail how we implement the YOLO model and reduce its scope from object detection to person detection. In this thesis, we take [7] as a reference and training facility, re-implement our version of the model since we want to adapt it to our framework but use the trainer they provided to re-train the model. The reason why we need to re-train is that the existing YOLO implementation is used for object detection rather than person detection. The difference between

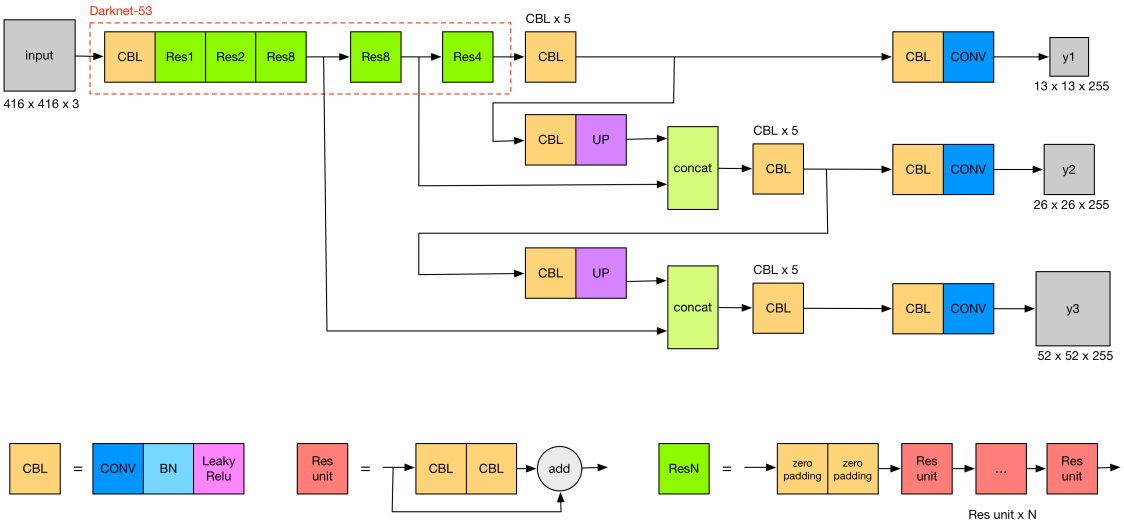


Figure 41: Implemented YOLO v3 architecture

them is that if you want to make your model be able to detect more classes of objects the more training time you need to spend as well as memory space. Since we want to integrate two deep learning-based approaches (detection and re-identification) in real-time, in such a context, both time and space complexity are essential to us.

In our implementation, we follow exactly the same network architecture proposed in [42] illustrated by Figure 41. Since there are already numerous pre-trained YOLO model in existence, we are not going to train it from scratch, but perform some fine-tuning processes on a pre-trained model to get the one which can fit our need. The full tuning process we employed can be described by the following steps:

1. Download the well-trained YOLO model from its official repository.

2. Convert the model's weight into Keras format from its original Darknet format.

3. Download the VOC2012 dataset and loop over all images in the training set annotating the one with person(s) and extracting their corresponding bounding boxes information.

4. Load the converted pre-trained weights then follow the methods proposed in the original YOLO v3 paper [42] to fine-tune the model to obtain a person detector rather than an object detector.

Since we reduce the complexity of an object detector to a person detector, we believe that could help to improve the training speed. The following discussion explains our rationale: According to [42], there is a total of 9 anchors which are obtained by applying K-means cluster algorithm during the training phase. These 9 anchors can be divided into 3 groups, each of them is given to final feature map with the size $[13, 13, 255]$, $[26, 26, 255]$ and $[52, 52, 255]$ respectively. So we can calculate that we are going to have: $13 \times 13 \times 3 + 26 \times 26 \times 3 + 52 \times 52 \times 3 = 10647$ bounding boxes for each input image. For each bounding box, we need to compute a probability for each of the pre-defined classes then multiply the confidence score for this box to obtain the final score for one class illustrated by Figure 42. Take the VOC2012 dataset as an example, there is a total of 20 classes of object. So for each bounding box, there will be 20 times multiplication operations. Since we have 105647 boxes, then it will be $20 \times 10647 = 212940$ operations per input image. **But if we only care about the person we can reduce the 20 classes to 2 classes, then the total calculation will be** $2 \times 10647 = 21294$ **which is one-tenth of the original one.** This can also speed up the non-maximum suppression (NMS) process which could help to eliminate overlapped boxes. In most of the deep learning problems where the training data is normally large, these modifications will significantly help to reduce the training time.

Once we have the score for each interested class, we will (1) use a threshold to filter out the one with a lower score and (2) apply the NMS algorithm to eliminate the boxes with high overlapping ratio. Assume we still use the VOC2012 dataset with 20 classes, since we have 10647 boxes, then the class score can form a 20×10647 (row $\times$ col) matrix. The procedure can be described in the following way and visualized by Figure 43.

1. Examine all the class scores, select a *threshold* then set all the score lower than *threshold* to be zero.

2. Take the class score for the same class from all predictive boxes, sort them in a decreasing order.

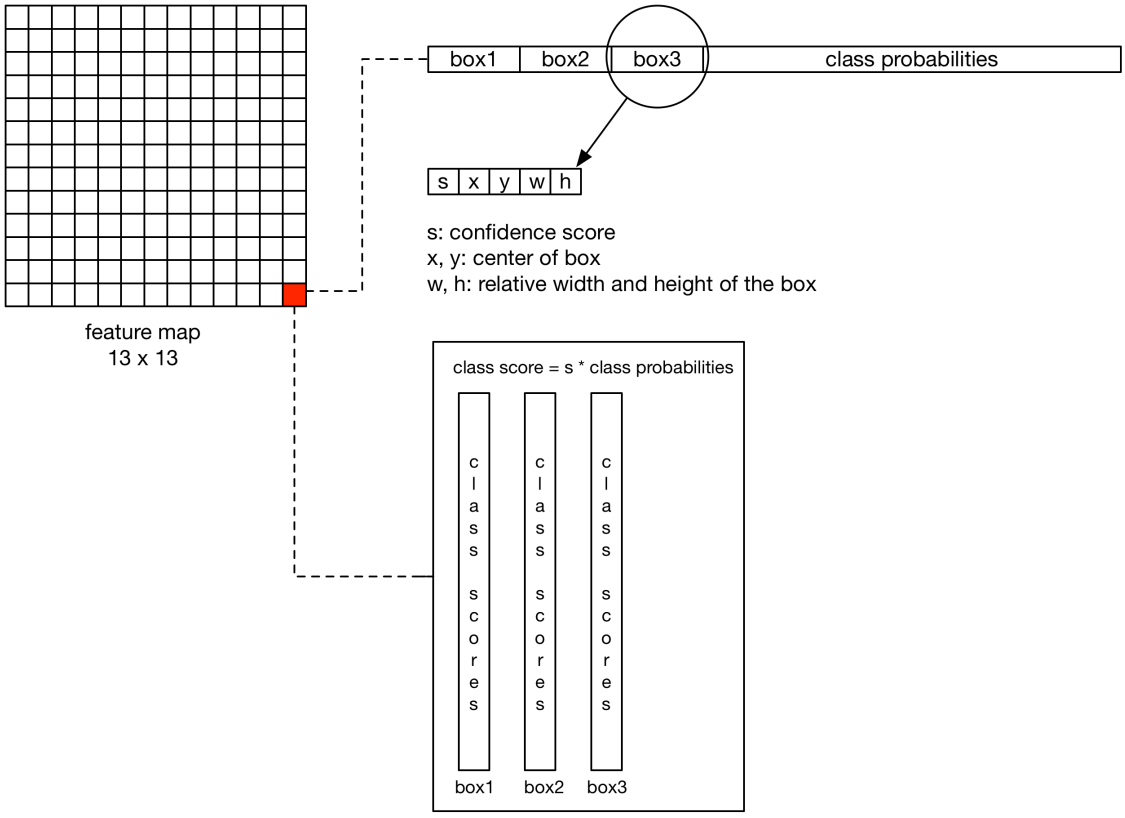


Figure 42: Calculation process of each cell in the feature map.

3. Mark the box with maximum score as `bbox_max`, then use it to compare with all the remaining boxes `bbox_cur` on the intersection over union metric.

4. if $IoU(\texttt{bbox\_max}, \texttt{bbox\_cur}) > 0.5$ then set the score to be zero. Otherwise, keep it unchanged.

4.3.3 Recognizer Specialized Framework Instantiation

Person recognizer is the most significant component in our specialized framework, without it we cannot reach our final goal covering the stage with more than one camera. In Section 3.3.3, we proposed the generic design of the frozen spot for a common recognizer. For our specific purpose, we need the capacity to re-identify the same person across multiple cameras, so what we need is actually a person recognizer instance (hot spot). Just like the way we did for the detector, we follow the same idea and come up with the instantiation plan shown as Figure 44. The only difference

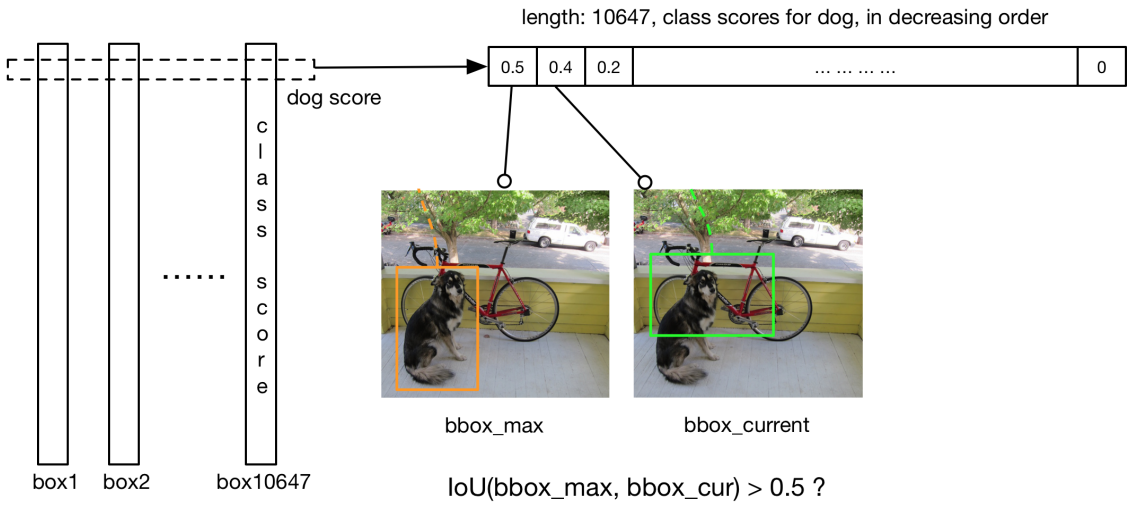


Figure 43: Non-maximum suppression process.

here is that the database invoked in this case, we have to create the database first before we use the recognizer, otherwise, there is nothing to be recognized. There is a variety of ways to create the database. The corresponding logic should be added into the `attachDatabase` method. And the comparison between the the input and the records within the database should be written in the method `lookupDatabase`. In this case, we are using a deep learning-based model, so we need to use the model to compute the descriptor for each given record and store them in the database then compare the distance between query and gallery descriptors. More detail can be found in Section 4.3.3.3.

In order to keep the consistency with the person detector and fill the research gap described in Section 2.3.7, our implementation of the recognizer is written in Python and built on top of TensorFlow and Keras. In the following paragraph, we first introduce our network structure, then explain our training process which includes data pre-processing, loss function, optimizer and some important hyper-parameters. In the end, we will discuss the methodology that we used to perform inference by using the trained model.

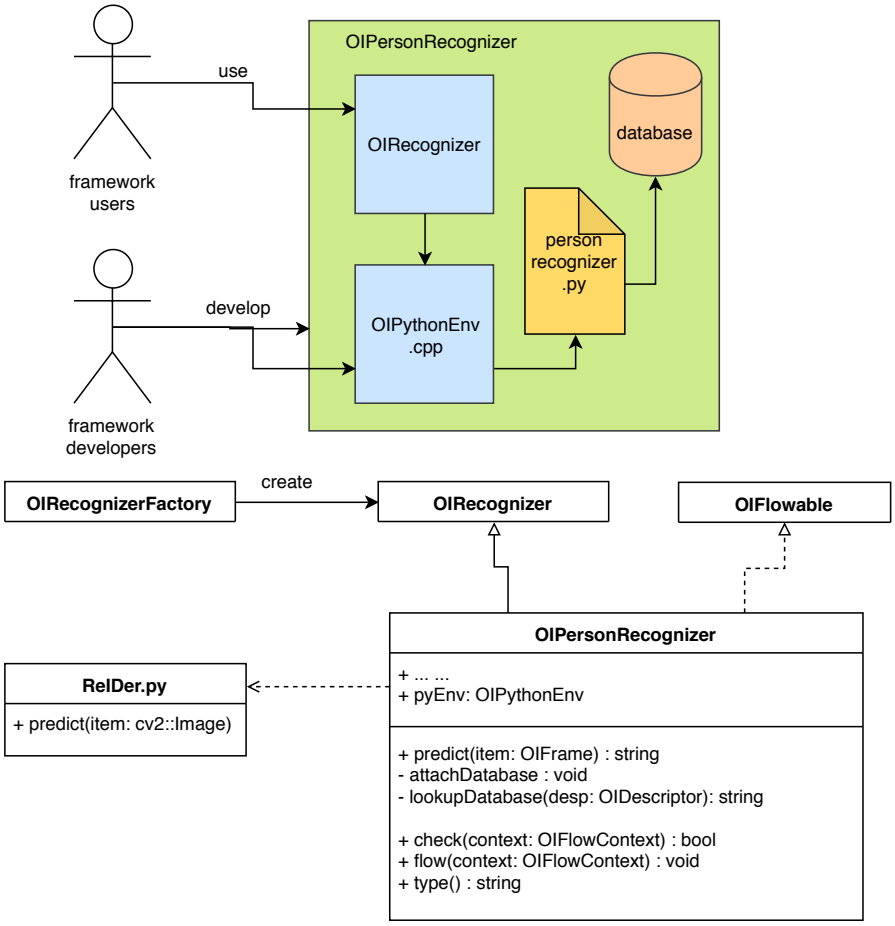


Figure 44: Instantiation of the recognizer specialized framework.

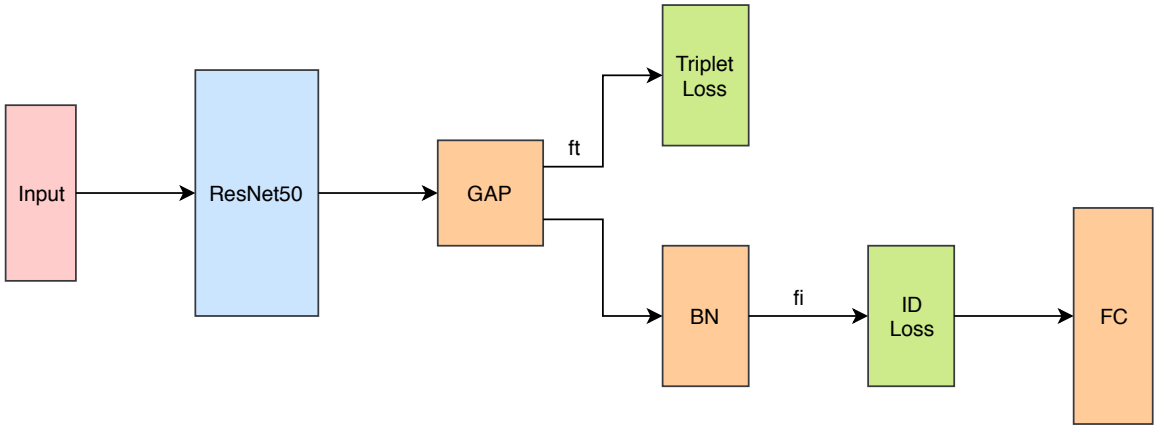


Figure 45: Implemented recognizer network architecture. The pink color represents input, blue means backbone network, orange represents a special layer and green means loss function layer.

4.3.3.1 Network Architecture

We implemented a network architecture shown as Figure 45, which is a combination of the identification model and distance metric-based model mentioned in Section 2.2. It employs a Residual-50 network as the feature extractor followed by a global average pooling layer (GAP) to flatten out the feature vector. Then in one branch, the extracted feature f_t was sent to calculate the triplet loss while in the other branch was used to obtain f_i after passing through a batch normalization layer (BN) to compute the identification loss. In the end, the fully connected layer (FC) is responsible for classification during the training time. According to [52], if we can get a higher spatial resolution before global pooling by changing the stride in the last convolutional layer from 2 to 1, which would not affect the number of parameters, obvious improvement can be obtained. Because of that, we modified our ResNet50 accordingly. Like most of the deep learning tasks, our model was also initialized with the weights pre-trained on ImageNet. From the implementation point of view, it is important to point out that when we are using any pre-trained weights, we need to make sure the input image respect to the format of the pre-trained model. In Keras, this can be done by invoking the `preprocess_input` method within the pre-defined model package.

4.3.3.2 Training

As we can see from Figure 45, during the training, our model will be guided by two loss functions: triplet loss and ID loss.

$$L = L_{triplet} + L_{ID} \tag{6}$$

where $L_{triplet}$ is defined as Equation 3, $L_{ID} = -\mathbf{y} \cdot \log(\hat{\mathbf{y}})$ is the cross-entropy loss.

Since the training set is too large to fit into memory in one shot, the mini-batch training strategy was adopted as well as the sampling method proposed in [23]. For each mini-batch, we randomly select P identities and for each identity random K images will be chosen. In our implementation, P is set to 16 and K is set to 4, which makes the batch size to become 64. This work is done by the class `RandomSampler`

whose UML diagram shown as Figure 46.

In order to prevent overfitting and enhance the generalization ability of the model, a data augmentation technique named random erasing [68] was applied to each image individually on-the-fly when constructing each mini-batch of data. Just like its name suggests, it will randomly replace a portion of the pixel's intensity within the image with some random values. Besides that three more steps pre-processing were applied to enlarge the dataset. These pre-processing methods are commonly used in the image-based deep learning problem, they are implemented in the file named `preprocess.py`.

- Pad 10 pixels around the image
- Randomly crop the image back to the size before padding
- Flip the image horizontally with 0.5 probability

For the optimizer, we used the build-in Adam algorithm provided by Keras but with warm-up learning rate setting [13]. Precisely, the learning rate has been scheduled as Equation 7, where t is the current epoch. Last but not least, according to [33], they trained their model for 120 epochs and it is sufficient to obtain a good result. Also, from our training result shown by Figure 48, this number is enough for the model's convergence, so we set our total training epochs to 120 as well.

$$\mathrm{lr}(t) = \begin{cases} 3.5 \times 10^{-5} \times \frac{t}{10} & \text{if } t \leq 10 \\ 3.5 \times 10^{-4} & \text{if } 10 < t \leq 40 \\ 3.5 \times 10^{-5} & \text{if } 40 < t \leq 70 \\ 3.5 \times 10^{-6} & \text{if } 70 < t \leq 120 \end{cases} \tag{7}$$

From the implementation point of view, the training program can be illustrated by Figure 47. Firstly, we defined a set of configuration variables and the structure of the model. Then we pass the configuration to the model, attach it with defined loss functions, optimizer and necessary callback functions then run the model in training mode. During the training time, the data will be retrieved from the dataset, sampled

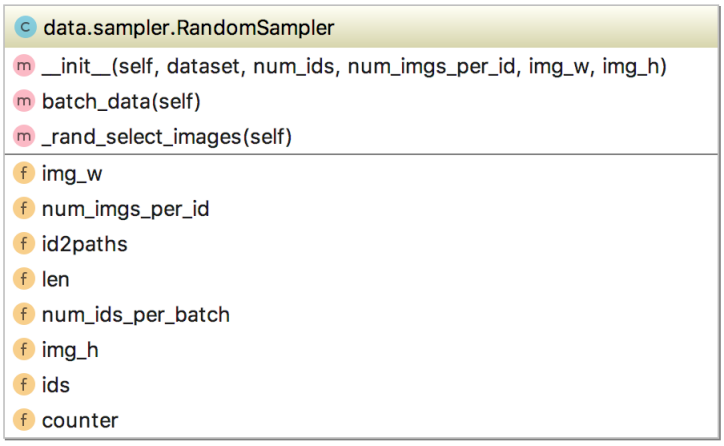


Figure 46: UML class diagram of `RandomSampler` class

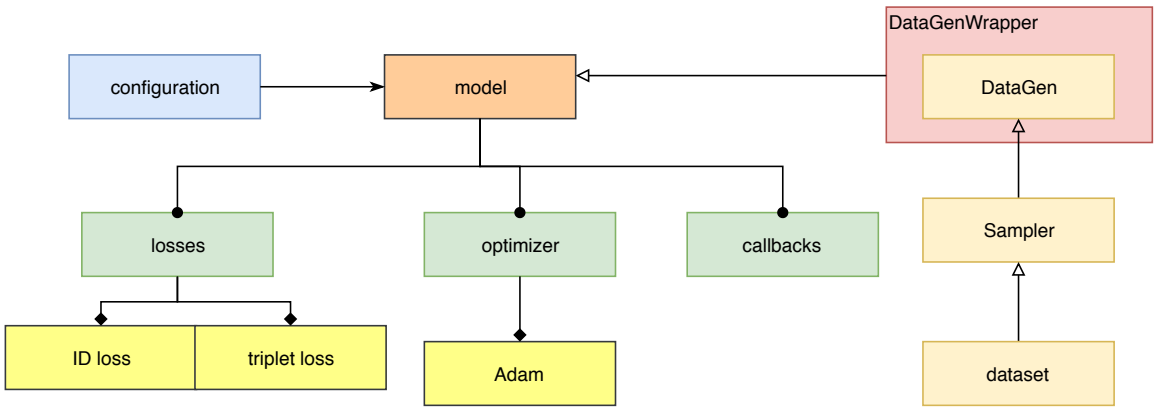


Figure 47: Code structure of the ReID training program

by the `Sampler`, applied data argumentation by `DataGen` and wrapped up to be a Python generator object by `DataGenWrapper`.

With these setting and after training, the result can be visualized by Figure 48, we can see clearly that the model starts to converge at around 50^{th} epochs. And from the training accuracy figure, we can find that there is a steep increase between 10^{th} and 20^{th} epochs.

4.3.3.3 Inference

We explained the training process above, let's take a look at the inference procedure. Assume we have a query image q and a set of gallery images G. The model is denoted by M and the output of model will be f_i then we have:

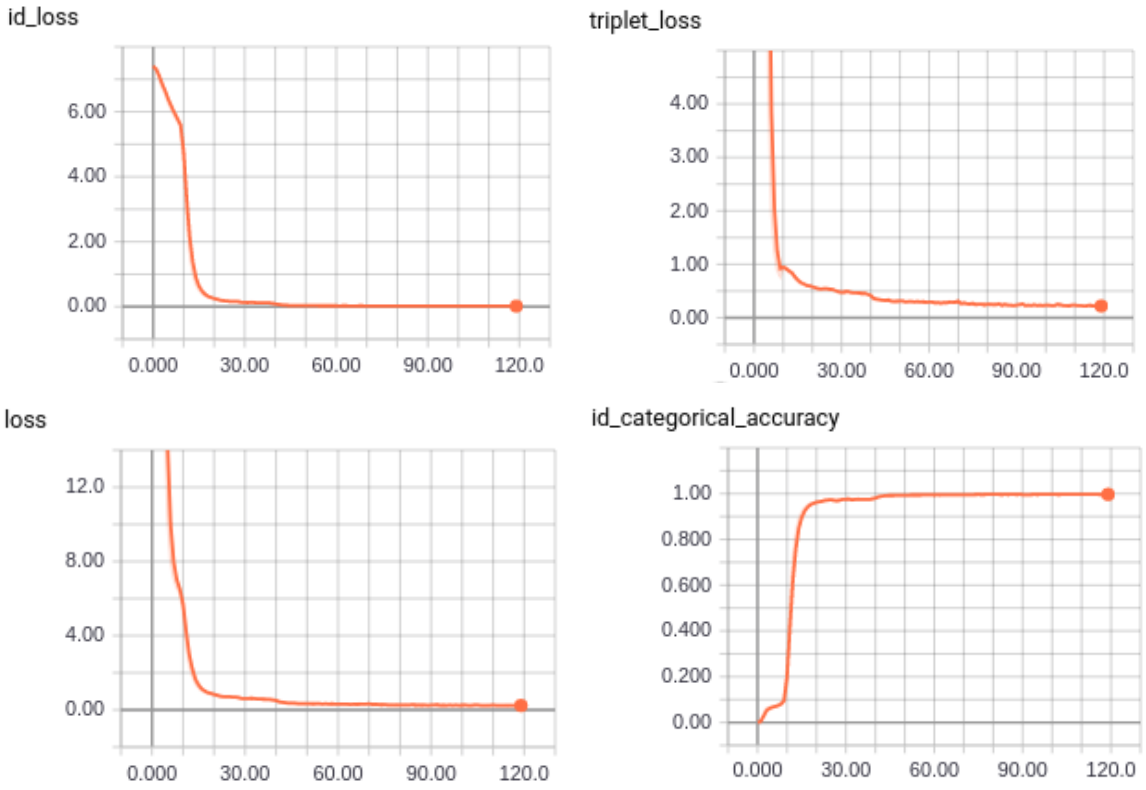


Figure 48: Training visualization diagram. Upper left: training identification loss curve. Upper right: training triplet hard loss curve. Lower left: total training loss curve. Lower right: training classification accuracy.

$$f_i^{query} = M(q) \quad \text{and} \quad f_i^G = M(G)$$

Once we have the feature descriptor, then for each $f \in f_i^G$ we calculate the distance D_i between f and f_i^{query}:

$$D_i = distance(f, f_i^{query}) \;\; f \in f_i^G \tag{8}$$

Finally, the gallery image which gives the minimal distance will be the one whose identity needs to be returned:

$$id = \arg\min(D_i)$$

If the reader wants to reproduce the result, attention needs to be paid on one

issue that the dimension of f_i^{query} and f_i^G are different so when we try to feed the single query image into the model, we have to explicitly add one more axis to it. What's more is that, according to Equation 8, a *distance* function is used to perform comparison between f and f_i^{query}. The most popular distance function for two vectors comparison are: (1) euclidean distance and (2) cosine distance. According to [33], using (2) can obtain a better result. Our implementation offers both two methods, but for simplicity we don't follow exactly the cosine distance calculation. We have the function `euclidean(f1, f2)` for the standard Euclidean distance but `L2(euclidean(f1, f2))` which is equivalent to cosine distance.

4.3.4 Tracker Specialized Framework Instantiation

In the plan of OpenISS framework, there are three kinds of trackers that are needed: skeleton tracker, facial landmark tracker and gesture tracker. In this thesis, we focus on the skeleton tracker. Skeleton tracking is the processing of depth image data to establish the positions of various skeleton joints on a human form. This information can be useful in many cases, as mentioned in Section 2.2.5, the approach proposed by [35] takes skeleton as the starting point. A lot of research has been conducted in this area and a lot of methods both conventional and deep learning-based have been developed. We will not focus on developing our own approach but to implement an existing one and design it as an instance of the specialized framework to allow users to integrate other approaches or develop their own approach easily.

Like what we did for the device abstraction, here we design a similar mechanism which is a hierarchical architecture shown as Figure 49 that can support real-time skeleton tracking with good extensibility without any GPU device needed. From the implementation point of view, firstly, we abstract the common functionalities of the skeleton tracker to create an abstract superclass `OITracker`, which serves as a contract exposed to the users and hides its implementation complexity. Our implementation is based on a middleware of OpenNI2 named NiTE2 mentioned in Section 2.3.4. So we create a concrete subclass which inherits from `OITracker` named `OINiTETracker`. It can be seen as an adapter that adapts the NiTE tracking

algorithm to our framework's data structure. Let's go deeper, starting from the method `readFrame(OITrackerFrame)`. Every time a new frame is read from the device, this method will be invoked. Its job is to perform skeleton detection to take the data from the NiTE2's data structure and move them to several OpenISS framework's data structures which will be eventually packed into `OITrackerFrame`. There are three specialized framework-specific data holder classes and their responsibilities listed below:

- `OIUserData` contains all the data for each single detected skeleton.
- `OIUserMap` contains a 2d array with the same resolution as the depth image, where the background indicated by 0 and the detected skeleton for each person indicated by their user id.
- `OISkeleton` contains a hashmap which the key is the joint type and the value is the position of that joint.

4.3.5 ReID Context Instantiation

As mentioned in Section 3.2.3, we have the pipeline module designed in the core framework. In this chapter, we have `OISkeletonTracker`, `OIPedestrianDetector` and `OIPersonRecognizer` both implementing the `OIFlowable` interface to serve as filters within a pipeline. In order to build up a ReID pipeline for our final goal, what we are still missing is a concrete class for the `OIFlowContext` which will be used as the data holder for intermediate result generated during the execution of the pipeline. For this purpose, we define a class `OIReIDFlowContext` which extends the abstract class `OIFlowContext`. For ReID task, the pipeline will need to contain the concrete implementation of a device, a detector, a recognizer and a viewer. Also, all the input and output of these components are being defined in the previous sections. So within the concrete context class, we need to define the way we store the temporary results. For example, the data frame captured from the device, the bounding boxes used to mask out the detected person, as well as the way we query them from the latter filter

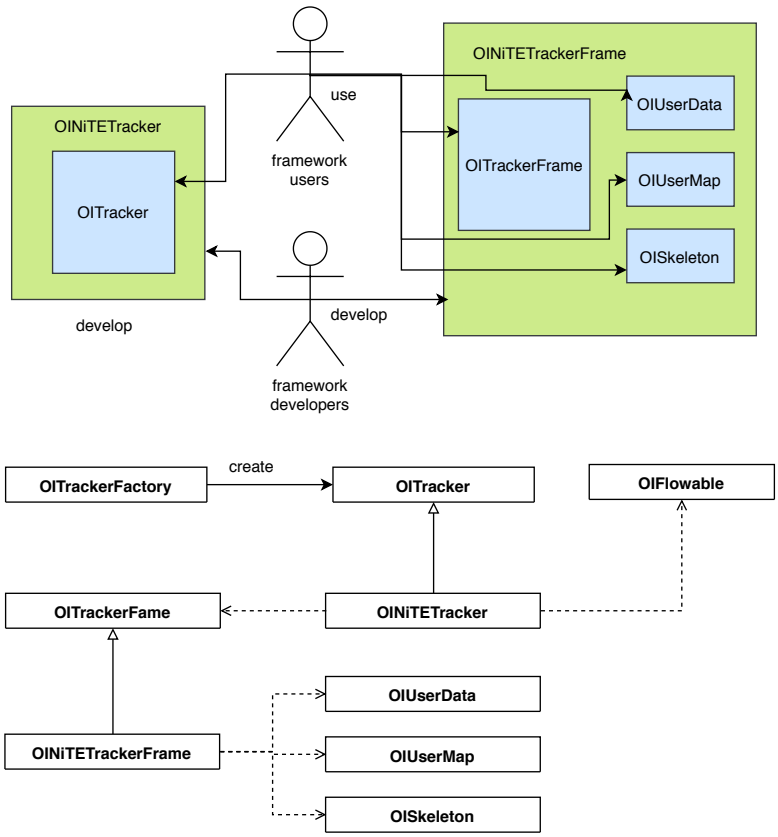


Figure 49: Instantiation of the tracker specialized framework.

within the pipeline. In this case, since we have the `type` of each filter and all of them are distinct from each other, we use their type as a key and create different logic when the `query` or `save` get called within their respective implementation as a pipeline filter.

4.4 Summary

In this chapter, we describe the instantiation and implementation for the frozen spots within both the core and specialized frameworks. A lot of focus is put on discussing two deep learning-based models' development (detector and recognizer) as well as how to integrate them into the specialized framework with the desired APIs exposed. In the next chapter, we will see how can we make use of these hot spot classes to form an application which can eventually address our research problem.

Chapter 5

Applications

In this chapter, we will explain how to make use of the framework instance created in Chapter 4 to build a person re-identification application to realize our main goal of tracking the same person across multiple cameras. Besides the main ReID application, we will also introduce other applications: skeleton tracking, camera calibration, image alignment and green screen image, which are all built on top of our OpenISS framework instance. With the applications added, the architecture of our current system can be shown by Figure 50.

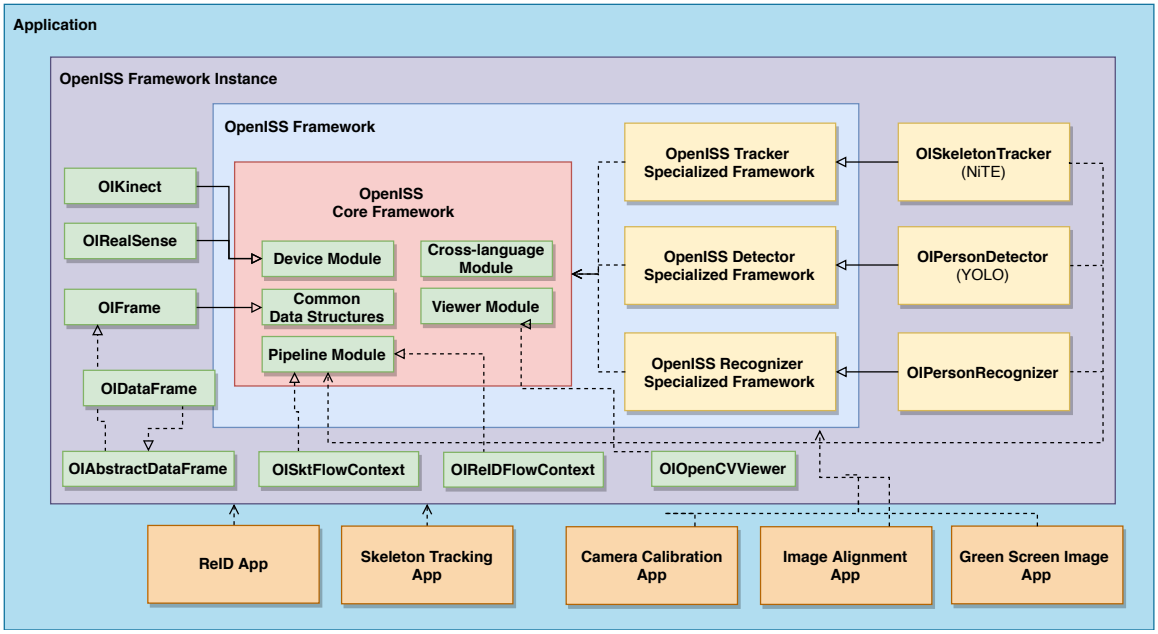


Figure 50: Applications built on top of our framework instance.

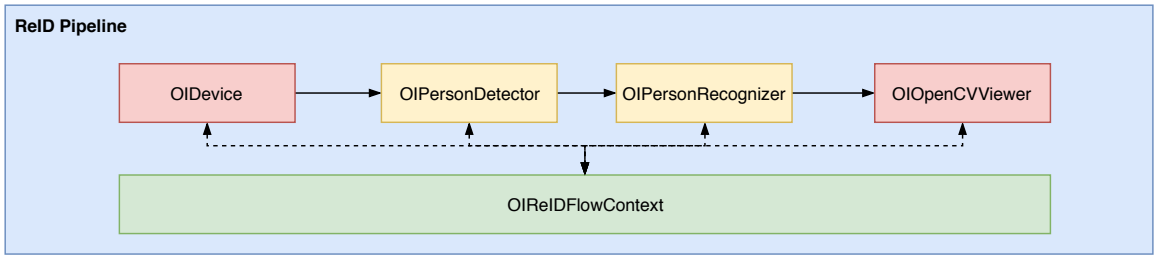


Figure 51: Person re-identification pipeline, the solid line arrow represents the pipeline execution order and the dash line arrow represents the data flow.

5.1 ReID Application

Until now, what we described in the previous chapters is the design and implementation of the framework itself. But it cannot solve our concrete ReID problem directly, since they are just pieces of solution for each subproblems. If we would like to use it for a specific task, we need a way to organize these fragments so that they can operate following our expectations. Our ReID application is the software we built using our framework instance to address the ReID task which includes person detection and person recognition.

As mentioned in Section 1.3, the person ReID task can be divided into two portions: one is person detection and the other is person retrieval. We have already shown how to create two specialized frameworks for each of these two parts. Intuitively, what we need to do next is to arrange them in a suitable order to make them work properly together. More specifically, the ReID application workflow can be described as following:

1. The raw data flow into the device module from the core framework being encapsulated as an instance of `OIFrame`.

2. The frame then being passed to the person detector which is a concrete implementation of the abstract `OIDetector` class invoking the deep learning-based model implemented in Python via the cross-language module in the core.

3. The output of the detector specialized framework will be a list of bounding boxes which can be used to mask the data frame, each of the masked result

contains a presence of a person.

4. These results will flow into the recognizer specialized framework which again depends on cross-language module since the corresponding model is also implemented in Python to compute and compare the descriptors among the database.

The process described above make a lot of sense but it requires the application developer to know how all these device module, person detector and person recognizer work, what are their input and output, etc. In most of the case, the application developers don't really care about these internal details, what they want just a working application that can achieve their goal. As a framework solution, we should provide such functionality along with ease of use through abstraction. The application developers tell the framework what they want and the framework takes care of the internal process and return the result directly. That is where the pipeline module in the core framework comes to the picture.

As mentioned in Section 3.2.3, the class `OIPipeline` serves as an execution engine for a list of filters. Filters are the classes which can live within a pipeline instance. They are required to implement the `OIFlowable` interface. All their input and output are stored inside the instance of the abstract class `OIFlowContext`. For the ReID task, we proposed a pipeline instance shown as Figure 51. Now, with the pipeline instance, the application developer doesn't need to worry about the details anymore (i.e. what is the return values of the person detector and how to handle them and pass them to the next filter) since the pipeline will take the program's flow of control. What the user needs to do is to tell the framework how to assemble these filters within the pipeline. That is done by creating an instance of a specific filter with type `OIFlowable` then invoke the `push` method defined in `OIPipeline` class and pass that filter as parameter. The steps we describe here can be accurately expressed by Algorithm 2.

So with the pipeline module, after assembling the pipeline instance, there is only one step to fire the ReID application. The user of our framework simply invokes one

Algorithm 2: ReID application procedure

```
devF ← create a device factory
detF ← create a detector factory
recogF ← create a recognizer factory
db ← create a database for recognizer

noEcsPressed = True
device = devF.create("name of the device")
detector = detF.create("name of detector")
recognizer = recogF.create("name of recognizer")
recognizer.attachDatabase(db)

reidContext = new OIReIDFlowContext
reidPL = new OIPipeline(reidContext)
reidPL.push(dev)
reidPL.push(detector)
reidPL.push(recognizer)
reidPL.push(new OIOpenCVViewer)

while noEcsPressed do
    reidPL.flow(reidContext)
    if isEcsPressed then
        noEcsPressed = False
```

function defined in the core, precisely, the `flow` method within the class `OIPipeline`. Then the framework gives the corresponding result back without any other user interaction or external programming required. What we described above is actually the beauty of a framework solution. It frees the user from knowing the temporary result to allow them to focus on their own application development. Also, it is a key feature of a framework solution. The system takes the control of the program which is known as inversion of control. With the pipeline module, or we can say, the framework solution, the user just needs to tell what they want and the framework will take care of the rest and return the result directly. Also, it enables a framework developer to design each specialized framework modularly and make these components reusable. Because they don't belong to any specific task anymore and may be used in any other pipeline instances.

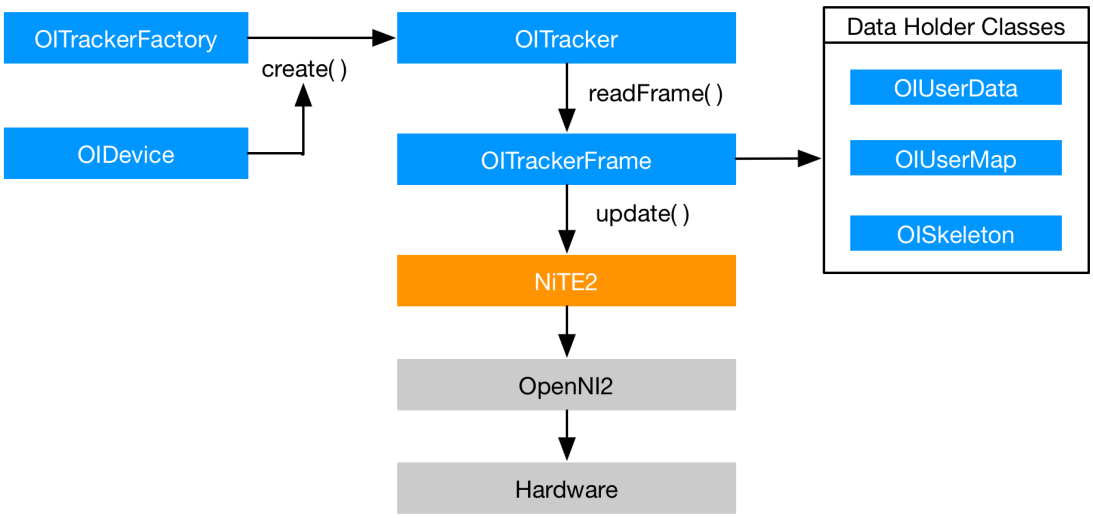


Figure 52: Tracker module interaction diagram, the boxes in blue are our framework's components, the box in orange is the concrete tracking algorithm implementation, the boxes in gray are the low level components.

5.2 Skeleton Tracking Application

As mentioned in Section 1.5.4, we would like to keep the functionality of skeleton tracking originally designed for ISSv2. To achieve that, we defined a set of frozen spots in Section 3.3.1 and create a specific hot spot for these frozen spots by adapting the implementation from NiTE2 in Section 4.3.4.

The workflow of the tracker instance is depicted in Figure 52. Firstly, the tracker factory `OITrackerFactory` will take a concrete class of `OIDevice` and create the instance of a concrete tracker but return a reference of its superclass `OITracker`. Secondly, the tracker will aggregate the information from NiTE2 and update the data holder within the concrete class of `OITrackerFrame`, in our case, `OINiTETrackerFrame`. Finally, the gathered data will be sent to the viewer and draw the skeleton out for the user. In Figure 52, the box in orange (NiTE2) is one of the possible implementations. It can be replaced by any other implementation by passing different indicators to the tracker factory.

In Section 4.3.4, we showed that the `OISkeletonTracker` class implements the `OIFlowable` interface which means that the skeleton tracking application will work in the same manner as our ReID application. With the pipeline mechanism introduced,

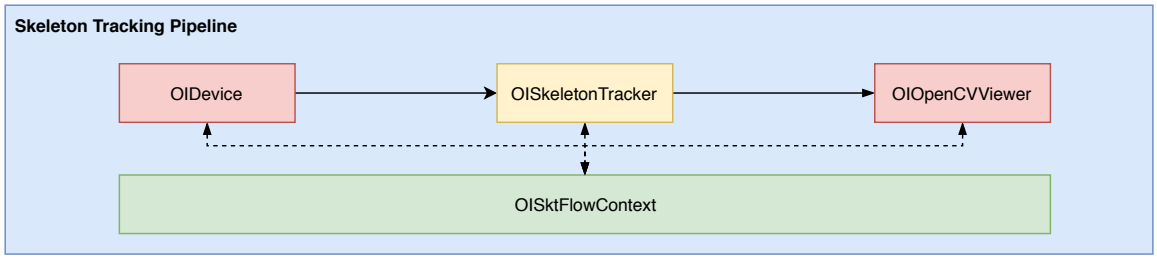


Figure 53: Skeleton tracking application pipeline. The solid line arrow represents the pipeline execution order and the dash line arrow represents the data flow.

our first step is to create a corresponding skeleton tracking pipeline shown as Figure 53, then we instantiate the a device filter as input, a tracker filter to perform tracking and a viewer filter for display. Next, as what we did for ReID application, we have to create a concrete `OIFlowContext` for the skeleton tracking task named `OISktFlowContext`. Finally, we invoke the `flow` method of the pipeline instance. The procedure can be described as Algorithm 3.

Algorithm 3: Skeleton tracking application procedure

```
devF ← create a device factory
tkF ← create a tracker factory

noEcsPressed = True
device = devF.create("name of the device")
tracker = tkF.create("name of the tracker")

sktContext = new OISktFlowContext
sktPL = new OIPipeline(sktContext)
sktPL.push(dev)
sktPL.push(tracker)
sktPL.push(new OIOpenCVViewer)

while noEcsPressed do
    sktPL.flow(sktContext)
    if isEcsPressed then
        noEcsPressed = False
```

Sample Name	Description
`calib.cpp`	Camera calibration application, which can be used to calibrate camera by inputting checkerboard images.
`kinect_capture.cpp`	Minimum Kinect application, it can stream both color and depth image of the scene in real time and display on the screen.
`kinect_sklt.cpp`	Skeleton tracking application using Kinect devices.
`rs_capture.cpp`	Minimum RealSense application, it can stream both color and depth image of the scene in real time and display them.
`rs_align.cpp`	Image alignment application, it can align the depth image to the color image captured by RealSense camera and also filter out the background based on the distance value.
`yolo.cpp`	Person detection application based on YOLO v3 algorithm built on top of OpenISS APIs.
`reid.cpp`	Person re-identification application.

Table 3: Available applications provided by OpenISS.

5.3 Other Applications

Besides the main person re-identification and the skeleton tracking application, we have also implemented some other applications to show the usability of our framework. All the available samples can be found under the path `OpenISS/samples/`. Currently, we have the sample applications shown as Table 3. Like most of the popular frameworks did, the samples not only prove our framework is useful but also serves as the learning material for the application developers who want to build software using our framework to learn how to use our APIs. In this section, we will examine some of the applications with their supported theory behind and the implementation detail.

5.3.1 Camera Calibration

Camera calibration, is one of the basic functionality of a computer vision-related library. Because of the limitation of manufacturer craft, the intrinsic matrix which is important for a certain camera application among of computer vision tasks varies across cameras. Also, the pose of camera which is described by the extrinsic matrix is another significant parameter as well. Camera calibration is a way to obtain both intrinsic and extrinsic matrices as well as fixing the distortion issue of the cameras.

5.3.1.1 Pinhole Model

Most of the cameras are using the pinhole model to capture images. The pinhole model can be shown as Figure 55, where $P(x, y, z)$ is a 3D point in the real-world, $P'(x', y')$ is the corresponding point in the 2D image plane and f' is the focal length. What a camera does can be imagined as a mapping relation between the real world 3D point and the image's 2D point, illustrated as Equation 9. If we know the real world poin'st P coordinate, by applying trigonometric calculations, we can calculate the coordinate of P' using Equation 10.

$$P = \begin{bmatrix} x \\ y \\ z \end{bmatrix} \rightarrow P' = \begin{bmatrix} x' \\ y' \end{bmatrix} \tag{9}$$

$$\begin{cases} x' = f' \frac{x}{z} \\ y' = f' \frac{y}{z} \end{cases} \tag{10}$$

5.3.1.2 Distortions Removal

Due to the fact that pinhole cameras may introduce distortion to images, we need to understand what is the kind distortion, then figure out how to fix it. There are mainly two kinds of distortions, radial and tangential distortion, illustrated by Figure 54. According to [61], radial distortion can be solved by Equation 11 and tangential distortion can be solved by Equation 12. Then eventually, we need to find five parameters to describe this model, formulated as Equation 13.

$$\begin{aligned} x_{\text{corrected}} &= x \left(1 + k_1 r^2 + k_2 r^4 + k_3 r^6\right) \\ y_{\text{corrected}} &= y \left(1 + k_1 r^2 + k_2 r^4 + k_3 r^6\right) \end{aligned} \tag{11}$$

$$\begin{aligned} x_{\text{corrected}} &= x + \left[2p_1 xy + p_2 \left(r^2 + 2x^2\right)\right] \\ y_{\text{corrected}} &= y + \left[p_1 \left(r^2 + 2y^2\right) + 2p_2 xy\right] \end{aligned} \tag{12}$$

$$\text{Distortion coefficients} = \begin{pmatrix} k_1 & k_2 & p_1 & p_2 & k_3 \end{pmatrix} \tag{13}$$

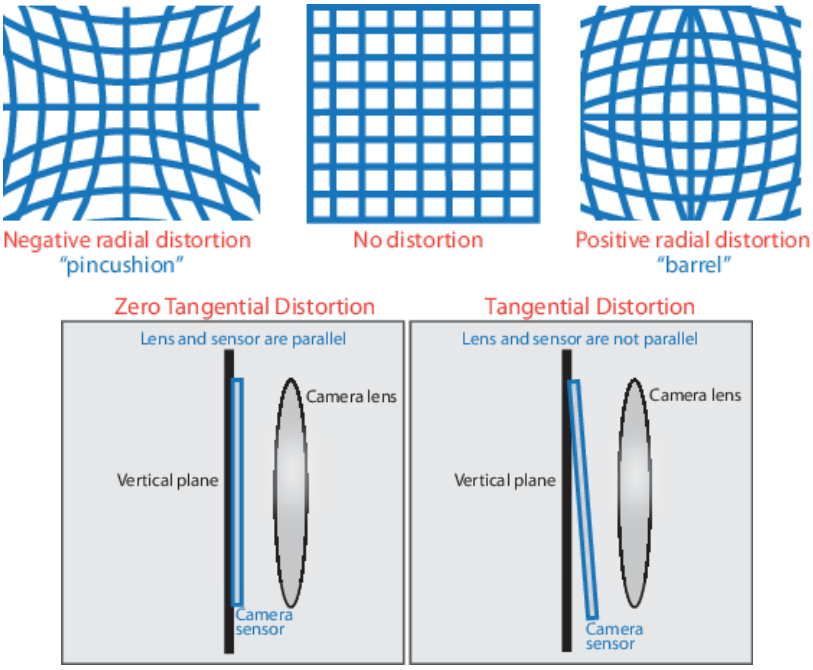


Figure 54: Two kinds of distortion models, the first row represents radial distortion, the second row represents tangential distortion [5].

5.3.1.3 Intrinsic and Extrinsic Matrices

Observed from Equation 10, division is not a linear transformation which is not convenient for calculation. So we move the coordinate from Cartesian to homogeneous to make the formula linearly computable, also assuming optical center at (u_0, v_0), pixel shape is square, no skew exists and no restriction to the camera pose. Then their relation can be concluded by Figure 56 and formulated by Equation 14 where K is the intrinsic matrix, E is the extrinsic matrix, R is the rotation matrix and $\bar{t}$ is the translation vector.

$$P' = \begin{bmatrix} f & 0 & u_0 \\ 0 & f & v_0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} r_{11} & r_{12} & r_{13} & t_x \\ r_{21} & r_{22} & r_{23} & t_y \\ r_{31} & r_{32} & r_{33} & t_z \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} = KEP = K[R \ \ \bar{t}]P \tag{14}$$

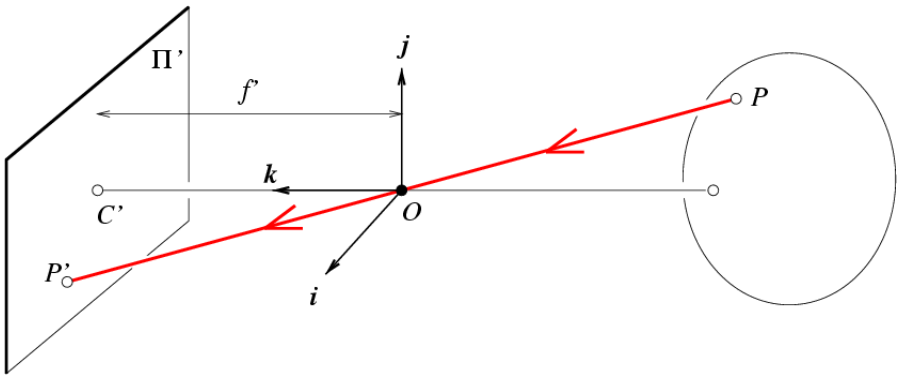


Figure 55: Pinhole camera model.

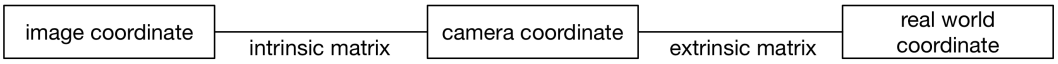


Figure 56: Relation between three coordinates and camera matrices.

5.3.1.4 Solving Distortion coefficient, Intrinsic and Extrinsic Matrices

In the camera calibration problem, we need to perform a reverse computation in which P and P' are known, the unknown being the intrinsic and extrinsic matrices. So we need to provide some sample images with well-defined pattern (typically a checkerboard). Then we detect the corners of the image whose positions are known. Finally, we solve the equation system to get our target K and E as well as the distortion coefficient.

We already have OpenCV as our dependencies and it has the functionality that can help us to solve those equations. So what we need to do just input a set of images with the pre-defined pattern. We break the implementation into several methods, the detailed explanation for each of them are listed below. A sample result can be found through Figure 57.

1. `prepareFileName`, takes a directory that contains all the images used for calibration as input, extracts their file path and puts them into a vector.

2. `prepareObjChessboardCorners`, prepares the pre-defined pattern of the corner and stores them into a vector.

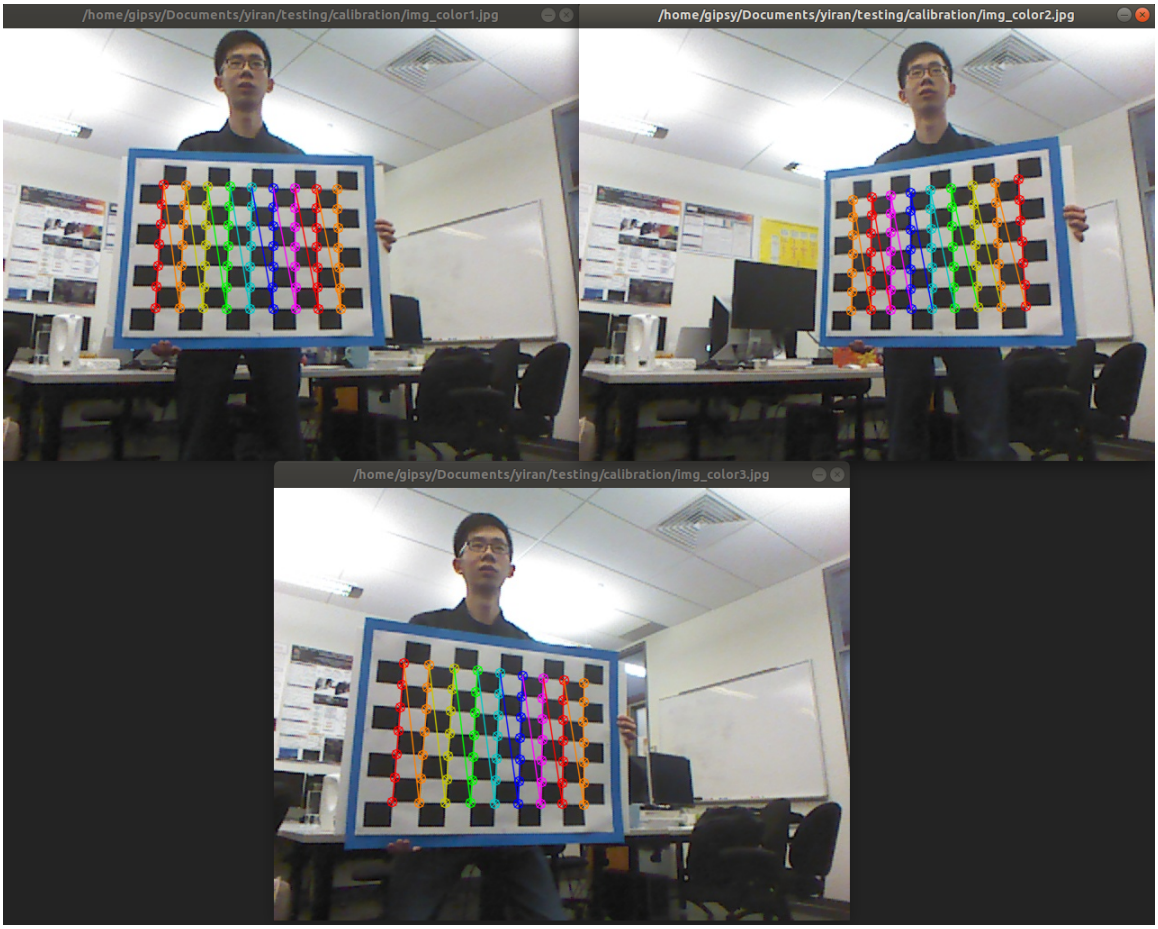


Figure 57: Example usage for camera calibration application.

3. `loadTestingImgAndFindCorner`, invokes OpenCV to load the image into memory and uses Haris-Corner detector to find a certain amount of corner points within all the loaded images.

4. `runCalibration`, takes the pre-defined corners and the detected corners as input applying the theory described in the previous section and solve the equation to find the distortion coefficient, intrinsic and extrinsic matrices.

5.3.2 Image Alignment

Since in our solution, we target the depth cameras as the input device, in such case, we will have not only the normal RGB image but also the depth image (an image where the value in each pixel is the distance of the object away from the depth sensor).

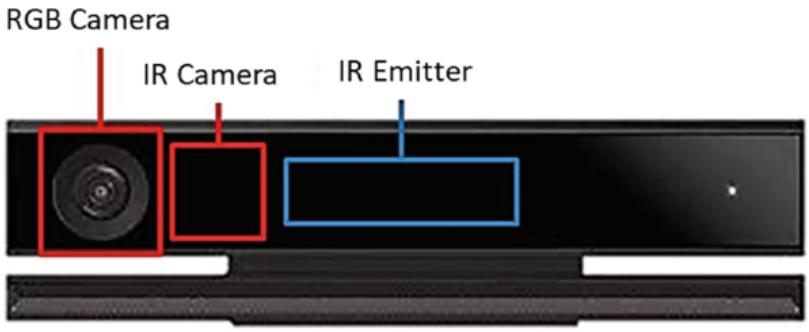


Figure 58: Kinect v2 sensor front with cameras and emitter positions.

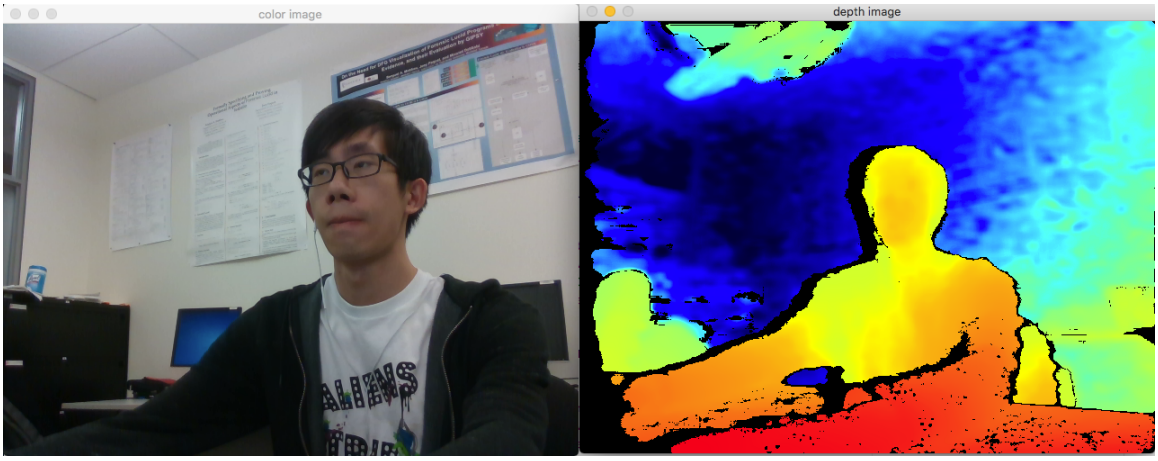


Figure 59: Raw color image and depth image without alignment, we can clearly see that there is displacement between these two images.

Take Kinect v2 as an example: from Figure 58 we found that the RGB and depth (IR) sensor are not at the same position on the camera housing which means that these two images for the same scene cannot be mapped pixel to pixel directly. An sample image pair before alignment can be shown as Figure 59. We can see that the person is closer to the image right-edge in the left image than that in the right image.

In this case, what we actually want to do is to align the depth image coordinates to the color image coordinates. More precisely, given the two types of image taken for the same scene at the same time, one is the depth image $I_{depth} = (a, b, depth)$ and the other is the color image $I_{color} = (m, n, intensity)$. For each pixel in I_{depth}, the alignment problem is to find the corresponding point, where $realworld(m, n) = realworld(a, b)$, in I_{color} and expand it to be $(m, n, intensity, depth)$.

Assuming that we already know the matrices of both depth and color cameras

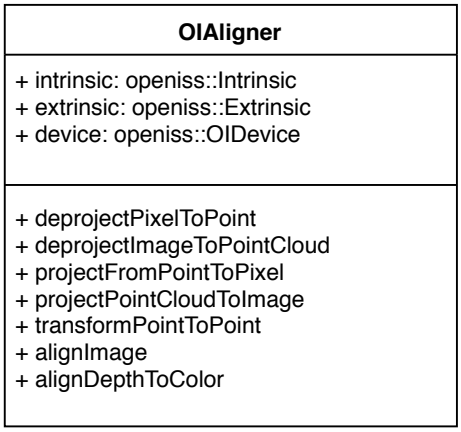


Figure 60: UML class diagram of `OIAligner` class

denoted by $K^{depth}_{intrinsic}$, $E^{depth}_{extrinsic}$, $K^{color}_{intrinsic}$ and $E^{color}_{extrinsic}$, then we loop over all the pixels in the depth image and try to re-project them onto the color image plane. For each pixel in the depth image $p_{depth}(x, y)$, we perform the following operations. In our solution, this work is done by `OIAligner` class shown as Figure 60.

1. With $K^{depth}_{intrinsic}$ and $E^{depth}_{extrinsic}$, we can project $p_{depth}(x, y)$ back to the real world coordinate to get $P(x', y', z')$.

2. With $K^{color}_{intrinsic}$ and $E^{color}_{extrinsic}$, we capture the re-projected point $P(x', y', z')$ and compute its corresponding point $p_{color}(x'', y'')$ in the color image plane.

5.3.3 Green Screen Image

Green screen image, is a technique which is widely used in the film industry. The originally idea is that we shoot a clip in front of a green backdrop then we apply whatever background we need to replace the green part of the captured image. In our case, we use this technique for background removal. Basically, it allows us to remove useless information of the scene depending on the depth value which maybe useful in some situations. For example, for the person recognition task, some models may expect the input just to be the person itself without any other noise, then this functionality becomes extremely helpful.

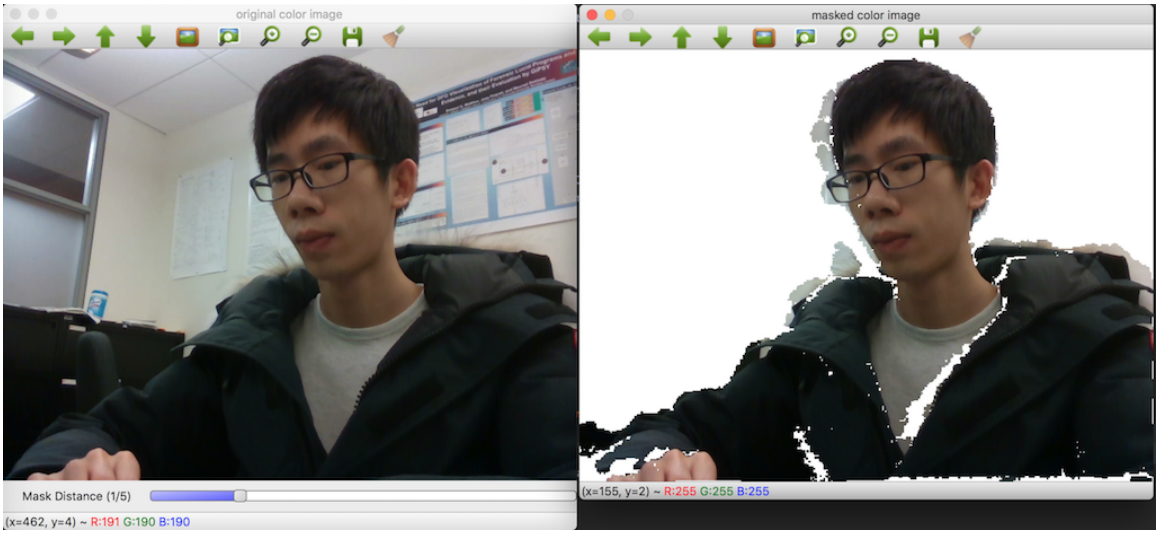


Figure 61: An example of the green screen image, left is the original color image and the right is the image after background removal.

In our implementation, we provide a GUI for the users to determine only to keep the information based on a depth value threshold. This function deeply relies on the two functionalities we mentioned above: camera calibration and image alignment. Only when the image is aligned, we are able to filter out the pixel whose deep value is larger than the threshold. An example of the application can be shown as Figure 61.

5.4 Summary

In this chapter, we introduced the applications which make use of our framework instance proposed in Chapter 4. In the ReID application, we mainly focus on the pipeline design architecture provided by our framework, which is the most valuable point of our framework in this case. With such a mechanism, we can integrate components into the framework easily, while maintaining a good modular design and these components can be reused for various tasks. Then we described three more applications which are common and basic within computer vision libraries frameworks. The camera calibration allows us to obtain more accurate intrinsic and extrinsic matrices, the image alignment enables us to map the depth image to the color image pixel by pixel and the green screen image makes use of the previous

two and provides background removal functionality that proves useful in various situations. In the next chapter, we will evaluate our framework solution according to our proposed requirements and demonstrate how well our solution can achieve using common metrics acknowledged by the community.

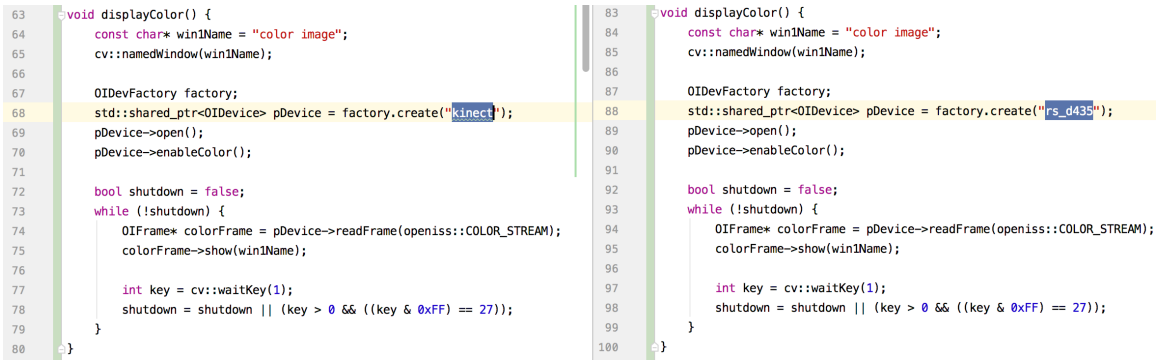
Chapter 6

Results and Evaluation

In this chapter, we will first evaluate our framework solution to show that we achieve the goal we set up at the very beginning in Section 1.4. Also, both the functional and non-functional requirements listed in Section 1.5 are fulfilled and the scenarios are realized by the applications we described in Chapter 5. As we are demonstrating that we can achieve the goal, we also want to analyze how well we can do it. So we describe the common metrics which are acknowledged by the research community. Then we employ exactly the same approach to evaluate our algorithms (or models) and report our result with respect to these metrics.

6.1 Framework Evaluation

We proposed a framework approach in Chapter 3 to address the limitations we have identified in Section 1.2 and fulfill the requirements listed in Section 1.5. In this section, we will examine the requirements and scenarios one by one to show how our solution addresses them. While doing so, we will also demonstrate the advantages of our framework solution and show that this approach makes it more valuable and interesting than just solve it by a more simple non-framework solution.



```
void displayColor() {
    const char* win1Name = "color image";
    cv::namedWindow(win1Name);

    OIDevFactory factory;
    std::shared_ptr<OIDevice> pDevice = factory.create("kinect");
    pDevice->open();
    pDevice->enableColor();

    bool shutdown = false;
    while (!shutdown) {
        OIFrame* colorFrame = pDevice->readFrame(openiss::COLOR_STREAM);
        colorFrame->show(win1Name);

        int key = cv::waitKey(1);
        shutdown = shutdown || (key > 0 && ((key & 0xFF) == 27));
    }
}
```

```
void displayColor() {
    const char* win1Name = "color image";
    cv::namedWindow(win1Name);

    OIDevFactory factory;
    std::shared_ptr<OIDevice> pDevice = factory.create("rs_d435");
    pDevice->open();
    pDevice->enableColor();

    bool shutdown = false;
    while (!shutdown) {
        OIFrame* colorFrame = pDevice->readFrame(openiss::COLOR_STREAM);
        colorFrame->show(win1Name);

        int key = cv::waitKey(1);
        shutdown = shutdown || (key > 0 && ((key & 0xFF) == 27));
    }
}
```

Figure 62: Effort needed to switch between different cameras, left is the code for Kinect and right is the code with the same effect for RealSense.

6.1.1 Device Switch and New Device Addition

As mentioned in Section 4.3.1, we have a device module that provides an abstraction layer for various physical devices that would enable the switch among different devices easily and also will not prohibitively increase the complexity of adding a new device.

Device switch: For both Kinect and RealSense cameras, we provide a minimum working sample code for the users to see how to use them (as a starting point). The code can be found under the path `sample/kinect/kinect_capture.cpp` and `sample/rs435/rs_capture.cpp`. From these two files, we can find that the only differences between them are just one line of code (even we can say just one word changed to switch between Kinect and RealSense), shown by Figure 62. This clearly demonstrates that our solution fulfills **FR1** proposed in Section 1.5.1.

New device addition: Since we have an abstraction layer on top of each concrete device implementation, we have to admit that when we try to add a new device it will introduce a little bit more work compared to we don't have any abstraction. But if we look at the convenience (one word changed for switching between Kinect and RealSense device) the abstraction layer brings to us we believe everyone will agree that introducing a minimal amount of additional work is necessary. Under our design of the framework, a developer needs to following steps to add a new device. It is evidence which can clearly demonstrate that we fulfill **FR2** proposed in Section 1.5.1.

1. Create a new class inheriting from `OIDevice` and implement all its pure virtual

methods under the directory `src/`.

2. Since we employed the factory design pattern, you still need to add the logic to instantiate the appended device in `OIDeviceFactory` class as well as the device-related memory deallocation when it hs become useless.

6.1.2 Back-end Abstraction

As explained in Section 1.4, we want the OpenISS framework to serve as a back-end of our previous work ISSv2. The image alignment application we introduced in Section 5.3.2 is a good showcase for it. In this application, no matter what kinds of camera you are using, the low-level APIs like `OIDevice` will abstract the physical device and provide the operands (e.g. intrinsic, extrinsic matrices and the data for each frame) needed for the image alignment computation. Then the aligned frame will be encapsulated as an OpenISS data structure `OIDataFrame` and returned for further usage. These processes, abstractions and encapsulations, are the core ideas of a back-end system which provides a set of usable and convenient unified APIs for the front-end to request without worrying about the complexity and implementation details. The same applies to other applications mentioned in Section 5.3. They all proved that the back-end abstraction is properly implemented and usable which means the requirements are in fact satisfied. But we have to admit that one more step needs to be done to complete the back-end abstraction. Since the ISSv2 was developed in Java and our current solution is written in C/C++. A Java wrapper that can expose the functionalities from our framework to ISSv2 is still missing.

6.1.3 Person Re-identification and Skeleton Tracking

In Section 1.5.3 and Section 1.5.4, we stated the scenarios of person re-identification and skeleton tracking. From these two scenarios, we extracted requirements related to these two functionalities. In Section 3.3, we described the design of our framework solution demonstrating its ability to address the ReID and tracking tasks in general. In Section 4.3.2, Section 4.3.3 and Section 4.3.4 we described an instance of our

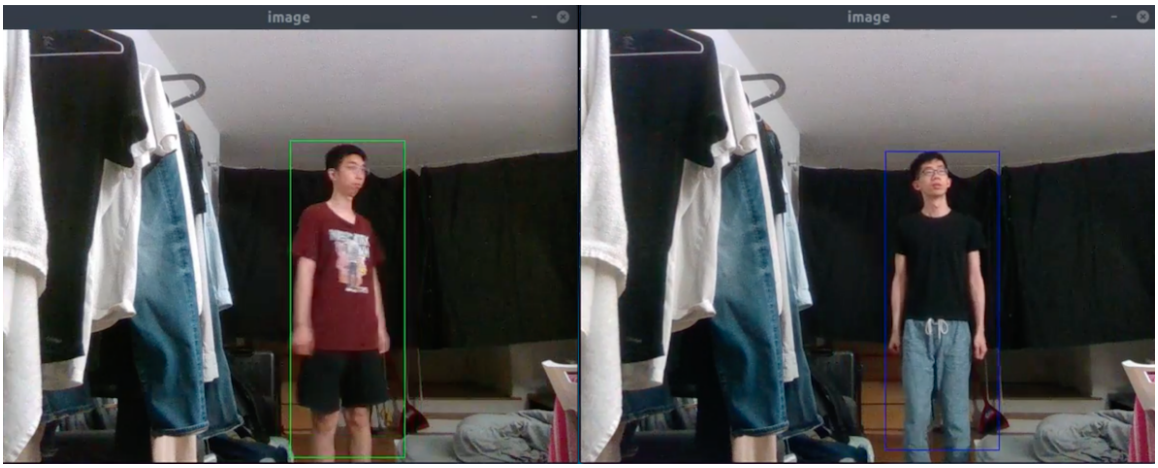


Figure 63: Sample result from our person re-identification application. The bounding box in different colors represents distinct identities recognized by our solution. In this example, the system finds two identities matched, the green box represents one on the left and the blue box represents the other one on the right.

framework which provides the functionalities that can solve our specific person ReID and human skeleton tracking problems. In Section 5.1 and Section 5.2, we described the application built on top of our framework demonstrating how we use our solution to address the problems mentioned above.

The ReID application described in Section 5.1 clearly demonstrated that it can detect the appearance of person and match that particular individual in a pre-defined database, which exactly fulfill **FR4** and **FR5** proposed in Section 1.5.3. A sample result of skeleton tracking from our solution is depicted in Figure 63.

The skeleton tracking application that was described in Section 5.2 clearly demonstrated that it can detect the appearances of persons and extract their skeleton points then keep tracking them, which exactly fulfills **FR6** proposed in Section 1.5.3. A sample result of skeleton tracking from our solution is depicted in Figure 64.

6.1.4 Real-time Response

Since we want our solution to address the problem in a live artistic performance production, real-time response is a non-functional requirement for our solution. As explained in Section 1.5.6, we set the real-time response baseline to be 10 FPS. To

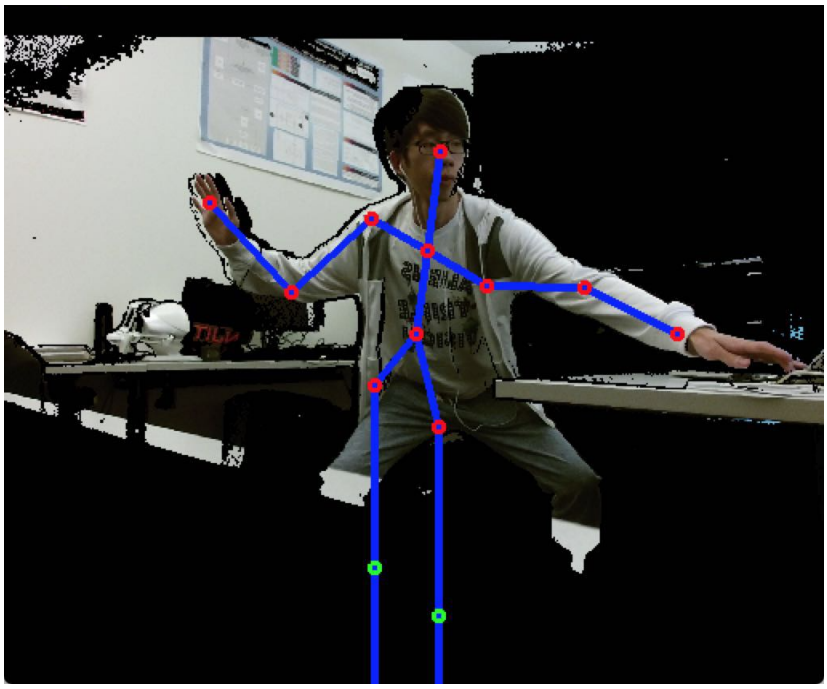


Figure 64: Sample result from our skeleton tracking solution. Red points represent high confidence and green points represent less confidence.

evaluate the whole system's processing ability, we perform an experiment described as below:

1. Create an database containing two target identities. Each of them has 3 images captures from camera $C1$.

2. Attach another camera $C2$ to the system and run the person ReID pipeline which includes streaming data from the physical device, performing person detection, person retrieval, and result visualization.

3. Measure the timestamp t_s before the pipeline start and the timestamp after the system process 500 frames t_d.

4. Calculate the frame rate using formula $fr = \frac{500}{t_e - t_s}$.

The result shown that the frame rate of our solution was 12 FPS which is slightly higher than the real-time baseline of 10 FPS. It is worthwhile to mention, a controlled

experiment, which only takes the data from the device and displays them one the screen, can reach a frame rate of 22 FPS using our solution.

Furthermore, we also want to evaluate the performance strictly from the ReID algorithm aspect which means without the time spending on data streaming and result visualization. For doing so, using the same testing protocol, we only measured the time spending on person detection and person retrieval. We performed the same process 1000 times and calculated the average. For each ReID operation, it takes about 0.066 second per image. More specifically, the person detection took about 0.051 second while the person retrieval took 0.015 second. So employing the frame rate formula, we can obtain the algorithm's time performance which is 15 FPS. By analyzing the time spent on each step, we concluded that the bottleneck is the person detection part which takes almost 3.5 more time than the time spent on the person retrieval part. The experiments described above were conducted on a local desktop machine, its specification can be found in Table 4, referred as `setting 2`.

6.1.5 Accuracy

In Section 1.5.6, we have identified a non-functional requirement for accuracy which stated that our ReID result should be comparable to the state of the art methods. Since there is no state of the art for the whole pipeline including person detection and person retrieval, we have to compare each part separately.

For the person detection model, ours is 5% less than the original YOLO v3 object detection algorithm on person category in term of mAP. The decrease of performance is due to the fact that we reduce the training labels to become only person and non-person which we believe it can improve the training and the whole ReID process time. Even the accuracy decreased, but it is still a considerably good result among the existing works. More details will be discussed in Section 6.2.1.4. For person retrieval model, ours is ranked 9^{th} and 7^{th} among the state of the art approaches on Market1501 dataset and DukeMTMC-reid dataset respectively regarding to top-1 accuracy. More details can be found in Section 6.2.2.4.

6.1.6 Extensibility

In Section 1.5.6, by analyzing the usage scenarios described in Section 1.5. We noticed that our solution should have extensibility as one of its non-functional requirements. The device switch and new device addition which we just explained in Section 6.1.1 is actually a good showcase for it. Also, our skeleton tracker instance implementation is another case. As explained in Section 4.3.4, we have a set of frozen spots defined for skeleton tracking, they are `OITracker`, `OITrackerFrame`, `OIUser`, `OIUserMap` and `OISkeleton`. When we want to integrate an existing tracker or develop a new one, the work will be just to create the corresponding hot spots, the process can be described as the following:

1. Create a concrete subclass inherited from `OITracker` and implement all the pure virtual methods.

2. Create a subclass of `OITrackerFrame` which is the wrapper of the data holder classes mentioned above, link it to `OITracker` class within its `readFrame` function.

3. Implement the logic for skeleton tracking algorithm for each coming image in the function named `update` in `OITrackerFrame`.

4. Add a new string as name to indicate the appended tracker and create an entry of it in the `OITrackerFactory` class.

What's more, the cross-language module and pipeline module are also showcases for extensibility. For the cross-language module, the way to add a new Python module (can be implemented with any Python-based deep learning framework) or any Python code-base can be described below:

1. Make a copy of your existing Python script, let's say with name "example.py", and all its dependencies into the `project_root/python/` folder and assume the function we want to expose is named "`func`".

2. Create a C++ wrapper class under the `project_root/src/` directory. The wrapper class should have a member variable with type `OIPythonEnv` initialized by the file name (example.py) and target function named (`func`). Then create a corresponding wrapper function, for example "`wrapper_func`", for the Python function "`func`".

3. Create helper functions to translate the parameter(s) and unpack the return value if needed.

4. In the application code, create an instance of the wrapper class that uses "`wrapper_func`" as "`func`".

For the pipeline module, the extensibility is reflected by the `OIFlowable` interface. Any class which can act or be expected to act as a filter (really common in a computer vision framework) can implement it and then be pushed into a pipeline instance. Such a filter is then independent from any other modules in the core or specialized framework which means it will not have any side effect on the existing implementation but just extend them. In order to create a new `OIFlowable` instance, the developer has to abide with the interface, to provide a concrete implementation of all abstract methods and to create a compatible instance of `OIFlowContext` to work with it. Furthermore, with the pipeline module, the flow of control becomes more simple and predictable. This enables the framework to provide more diversified processing power and to hack into each processing step providing more useful functionality. For example, if we have 10 kinds of person recognition algorithms and would like to know the performance of each of those in order to select the fastest one, then we can put two timer filters right next to the recognizer without changing the code of the recognizer itself. This is a perfect example of OCP (open-closed principle) which means that software entities should be open for extension, but close for modification, which is a cornerstone of software extensibility.

6.1.7 Usability

From Section 1.5.6, usability has also been identified as one of the non-functional requirements of our solution. In this thesis, by usability, we have two meanings. One is that the system is usable for our goals and applications, the other is that it should be also user-friendly.

To prove our framework is usable, in Chapter 5, we listed all the sample applications we provided along with our framework. These applications can serve as proof clearly show that our framework can be used to perform the following tasks:

- Capture data from various physical devices and display them.
- Use the captured data to perform person detection in real-time.
- Use the person detection result and pre-defined database to achieve real-time person re-identification.
- Calibrate the connected camera sensors.
- Align the captured depth image to the corresponding color image.
- Remove background for a captured image specified by a distance threshold.
- Allow C/C++ code to communicate with Python code
- Allow the user to assemble different filters for various kinds of task flexibly.

In term of the solution can be used with ease, currently, we don't have any experiment to prove it yet. As an adaptation, we counted the number of application level code we need for our scenarios. The minimum example we mention in Section 6.1.1 shows that in just 14 lines of code we can activate the device accessing the data and display them. Also from the person detector sample code `sample/yolo/main.cpp`, we found that to enable the whole person detection pipeline and display the result just takes 18 lines of code. What's more, for the ReID pipeline, using only 36 lines of code, we can query data from a device, perform the detection

and re-identification as well as comparison with the database and display the result. Last but not least, for advanced users who may want to integrate their custom deep learning model the procedure is also straightforward. With our cross-language invocation module, all you need to do just create a wrapper class then specify your Python script name and the functions you would like to expose. The person detection and person recognition described in Section 4.3.2 and Section 4.3.3 respectively are good showcases of it.

6.2 Person Re-identification Evaluation

As discussed in Section 1.3, the person ReID task can be divided into three subtasks: person detection, person tracking and person retrieval. In our solution, we have omitted person tracking since we performed detection on each incoming frame which has the same effect as tracking. The methods used to evaluate detection and retrieval algorithms have already been well-defined and acknowledged within the community. In this section, we will introduce these evaluation methods and report the results we have obtained by applying these evaluation methods on our implementation.

Before we move to the evaluation methods, we give the details of the environment settings we are using both for training and testing. We used two execution environments: one is a local desktop machine and the other is the Virya cluster provided by the faculty. The hardware specification of these two environments shown as Table 4. We will refer to them as `setting 1` and `setting 2` respectively in the following content. There is a slight difference between the software version installed in these two environments shown as Table 5.

6.2.1 Person Detection

6.2.1.1 Intersection Over Union

In order to measure the quality of the person detector we described in Section 4.3.2, we use the metric called "intersection over union(IOU)". It requires a ground truth

Setting	Name	Amount	Device
1(Desktop Machine)	Memory	1	7.7 GB
	Processor	1	Intel Core i5-3470 CPU @3.20GHz × 4
	Graphics	1	GeForce GTX 1070 Ti (8GB memory)
	OS	N/A	Ubuntu 18.04.1 LTS 64-bit
2 (Virya Cluster)	Memory	1	400 GB
	Processor	1	72-core CPU
	Graphics	8	Tesla V100 (32 GB memory)
	OS	N/A	Scientific Linux

Table 4: Environment hardware specification.

Software	Setting 1	Setting 2
Python	3.6.7	3.6.8
TensorFlow	1.12.0	1.13.1
Keras	2.2.4	2.2.4
Keras-application	1.0.6	1.0.7
Keras-preprocessing	1.0.5	1.0.9

Table 5: Environment software specification.

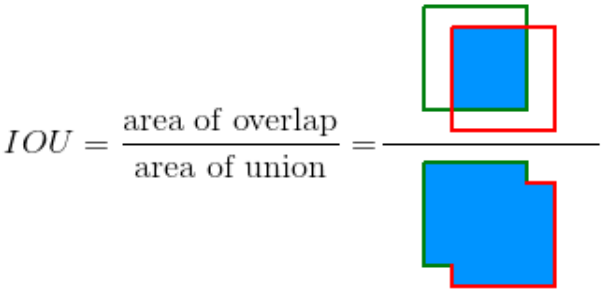


Figure 65: Intersection Over Union metric for object detection. The green box represents ground truth and the red box represents the detection result. The blue area on top represents intersection and the one on bottom represents union.

bounding box B_{gt} and a predicted bounding box B_p. By calculating the IOU metric, we can tell the bounding box the detector produced is valid or not. IOU is formulated as Equation 15, and can be visualized as Figure 65.

$$IOU = \frac{area(B_p \cap B_{gt})}{area(B_p \cup B_{gt})} \tag{15}$$

With the computed IOU result and a hyper-parameter *threshold* which is usually set to 50%, 75% or 90%, we can define the following four terms to describe a detection:

- True positive, a correct detection where $IOU \geq threshold$.
- False positive, a incorrect detection where $IOU < threshold$.
- True negative, does not apply.
- False negative, a ground truth not detected.

6.2.1.2 Precision-Recall Curve

By analyzing these results, the two metrics precision and recall can be used to describe how well the model performs in two different aspects. Precision Equation 16 is a fraction of relevant instances among the retrieved instances which can be used to measures how accurate is your prediction. Recall Equation 17 is a fraction of relevant instances that have been retrieved over the total amount of relevant instances. It can be used to measure how good you find all the positives.

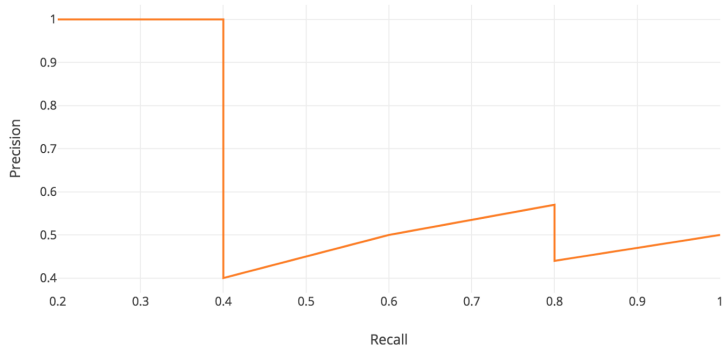
$$precision = \frac{\#\text{ true positive}}{\#\text{ true positive} + \#\text{ false positive}} \tag{16}$$

$$recall = \frac{\#\text{ true positive}}{\#\text{ true positive} + \#\text{ false negative}} \tag{17}$$

With precision and recall in hand, we can construct a Precision-Recall curve (P-R curve) which will be used to calculate our metric. The curve construction process can be described as the following:

1. Collect all the predictions that make for a particular class of objects. Rank them in decreasing order according to the confidence score given by the model.
2. Compare theses predictions with ground truth to see if they are correct or not.
3. Calculate the precision and recall using the given formula for each prediction in the ranked list from top to bottom.

An example of the P-R curve can be shown as Figure 66. Let us examine how the precision and recall are calculated. The first row of the right-hand-side table belongs



Rank	Result	precision	recall
1	true	1.0	0.2
2	true	1.0	0.4
3	false	0.67	0.4
4	false	0.5	0.4
5	false	0.4	0.4
6	true	0.5	0.6
7	true	0.57	0.8
8	false	0.5	0.8
9	false	0.44	0.8
10	true	0.5	1.0

Figure 66: An example of P-R curve. Left is the curve itself and right is the data used to plot this curve, assuming that there is a total of five positives in the data.

to the prediction whose confidence score is the highest among all the predictions for a particular class. From the table, we know that this prediction is true. Under the assumption that the total number of positive results is 5, according to Equation 16 and Equation 17, $precision = \frac{1}{1} = 1.0$ and $precision = \frac{1}{5} = 0.2$ We calculate the third row using the same pattern. We already have two results (rows) before so for the this row is: $precision = \frac{2}{3} = 0.67$ and $precision = \frac{2}{5} = 0.4$.

6.2.1.3 Average Precision

Since the P-R curve is often zigzag which is not easy for comparison, such as shown in Figure 66. We may want to apply a numerical transformation that can enable a more direct comparison. Average precision (AP) is such a metric. In general, it represents the area under the P-R curve and can be formulated as Equation 18. Precision and recall are always within $[0, 1]$, so to compute AP we can do an integration of the P-R curve within this interval.

$$AP = \int_0^1 p(r)dr \tag{18}$$

$$p_{\text{interp}}(r) = \max_{\widetilde{r} \geq r} p(\widetilde{r}) \tag{19}$$

Under the context of object detection, before we compute AP, for simplicity purpose we will first smooth the zigzag P-R curve by taking the maximum precision

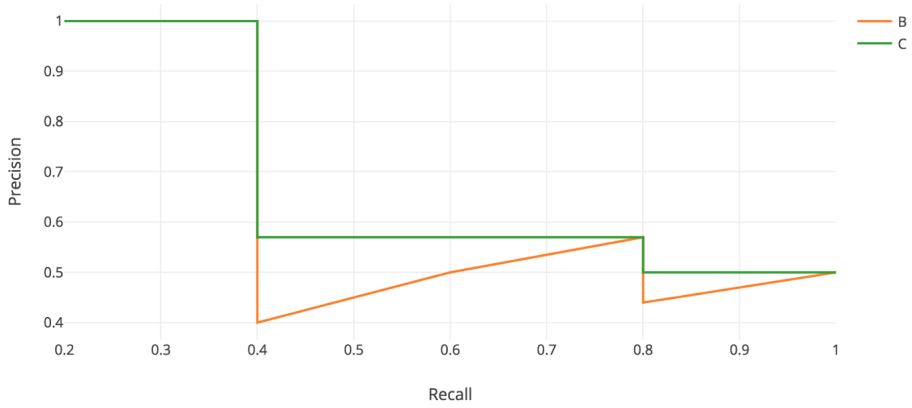


Figure 67: An example of smoothed Precision-Recall curve. The orange line is the original zigzag curve, green line is the smoothed curve.

at each recall level mathematically represented by Equation 19. After smoothing, the curve will look like Figure 67. Then we have two kinds of AP: interpolated AP and AP (Area Under Curve, AUC).

Interpolated AP

Starting from Pascal VOC2008 until VOC2010, an average for the 11-point interpolated AP is calculated. We first divide the recall value into 11 points $[0.0, 0.1, ..., 1.0]$, then we compute the average of maximum precision value for these 11 recall values. Precisely, the calculation can be shown by Equation 20.

$$\begin{aligned} AP &= \frac{1}{11} \sum_{r \in \{0.0,...,1.0\}} AP_r \\ &= \frac{1}{11} \sum_{r \in \{0,0,...,1,0\}} p_{interp}(r) \end{aligned} \tag{20}$$

AP (Area Under Curve, AUC)

For later Pascal VOC2010-2012, all unique recall values are sampled, whenever the maximum precision value drops. With such an operation, we are able to measure the exact area under the P-R curve after the zigzags are removed. The operation can be expressed using Equation 21.

$$\mathrm{AP} = \Sigma \left(r_{n+1} - r_n\right) p_{\mathrm{imerp}} \left(r_{n+1}\right) \tag{21}$$

where

$$p_{\text{interp}}\left(r_{n+1}\right) = \max_{\tilde{r} \geq r_{n+t}} p(\tilde{r})$$

6.2.1.4 Experimental Result

We trained the YOLO model to become a person detector using the VOC2012 dataset as explained in Section 4.3.2, then we perform an in-dataset validation. We randomly sample 5823 images from the dataset as validation set which never be used for training. In the randomly sampled validation set, we have total 4372 ground truth boxes. During the validation time, 3950 bounding boxes have been detected while 3423 of them have been located properly and 527 of them are wrong results, as shown in Figure 70. The IOU threshold is set to 0.5 according to the literature's common knowledge. By performing the detection using our model on the validation set, we have the Precision-Recall curve shown as Figure 68. Since we only have one class, the mAP will be exactly the person's AP (area under curve) which is 76.08% as shown on the top of the figure.

For comparison purposes, we used the same model definition training on the same dataset but with all 20 classes, the result are shown as Figure 69 and Figure 71. The result is obtained from the same sampled validation set. The only difference is the model used to perform the detection: one is the person detection model and the other is the object detection model with 20 classes supported. We can clearly see that the result from the object detection model about 5% outperforms the person detection model. That is also explainable since the person detection treats the problem like a binary classification problem. We only train it with person or non-person label while the object detection model takes it as multiple classes. With a certain amount of data, a multi-class detector can perform better than a binary detector. A straightforward example can be the following: Imagine we have a cat binary classifier and given a tiger image, the cat classifier may take it as a cat but if we have another well-trained tiger/cat/dog/fish/etc classifier, then it has a good chance to classify it properly.

One more thing that needs to be mentioned is that in Section 4.3.2 we claim that by theoretical analysis the object detection model should take more time to train.

In our experiment with `setting 2`, the object detection model takes 504 minutes to train while the person detection model takes 416 minutes to train, unlike our analysis which is one-tenth of its object detection sibling. That is because we train our model on a GPU. With parallel computation supported, it can perform the calculation concurrently.

With the reduction from object detection to person detection, we even lose 5% accuracy. It seems we can only reduce the training time and have nothing to do with the inference time, which is actually a bad trade. However, that consideration is not true. Arguably, if we still use the objection detector, after the detection process we may have a large number of bounding boxes being detected. Then we need to perform NNS to eliminate those overlapping boxes which is time consuming as well as the non-person boxes which has linear time complexity. It is actually a trade-off between time and accuracy. Since real-time performance is the hard constraint in our case, we give high priority to the time instead of accuracy.

6.2.2 Person Retrieval

We test our person retrieval implementation described in Section 4.3.3 through the following method:

1. We split all the identities we have in the dataset into two parts: One is the training set T and the other is validation set V. There is no overlapping between T and V which means $T \cap V = \emptyset$. Each identity has a certain number of images captured from different cameras.

2. Then we take all the images corresponding to the identities in V, further splitting them into query set Q and the gallery set G. For each image in Q there must be at least one corresponding image in G that shares the same identity label but captured from different cameras.

3. We use our trained model to extract feature descriptor f_i^g for all the images in the gallery to form a database D.

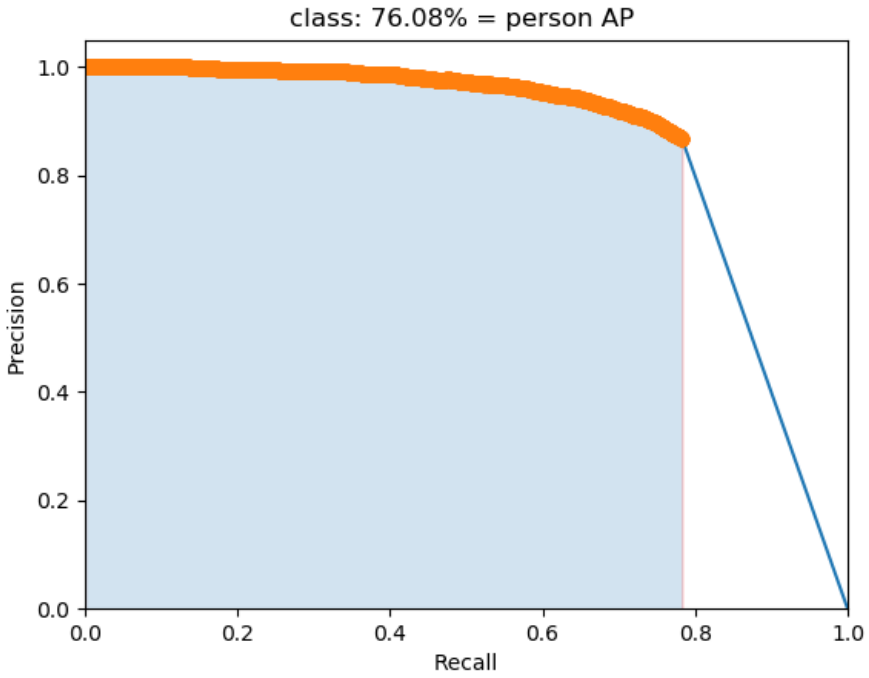


Figure 68: Person detection model's Precision-Recall curve on validation set.

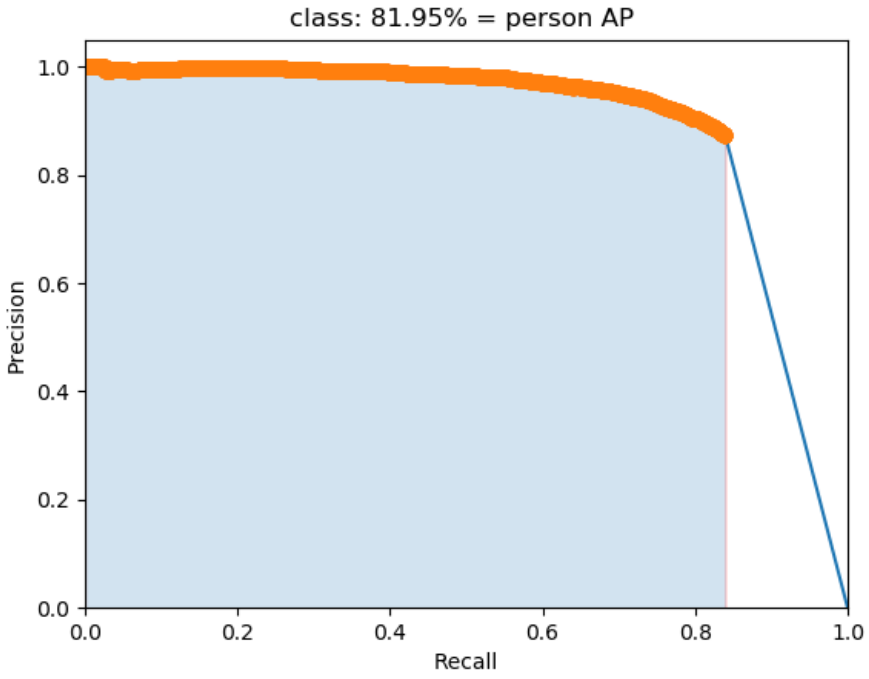


Figure 69: Object detection (with 20 classes supported) model's Precision-Recall curve on validation set.

Figure 70: Person detection result on validation set.

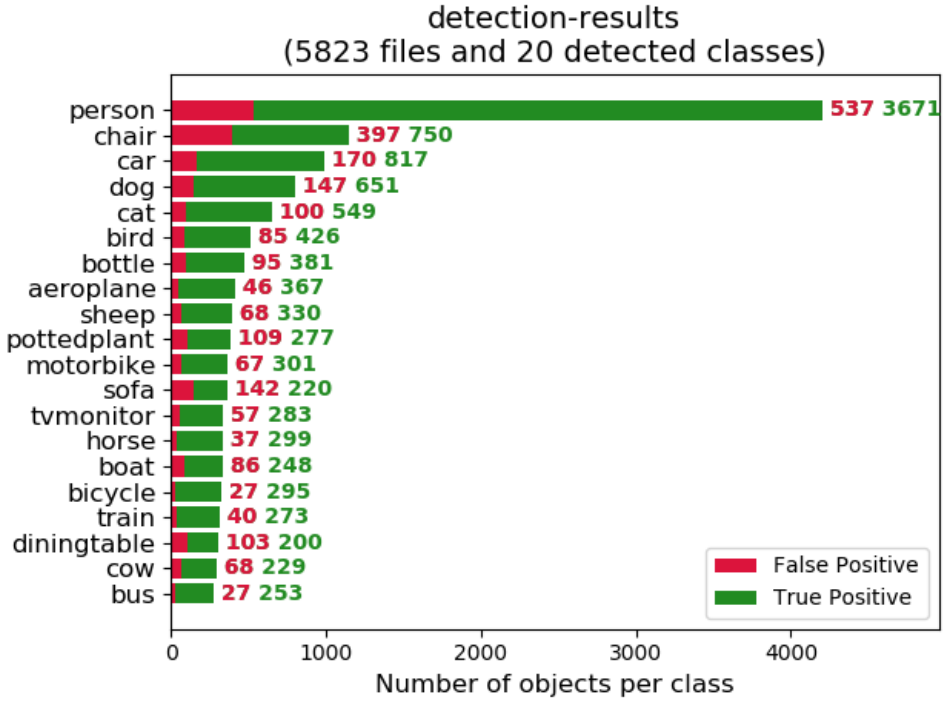


Figure 71: Object detection result on validation set.

4. For each image $q \in Q$, we also used the same model to extract the feature descriptor f_i^q, then sort all the descriptors in the database according to $distance(f_i^q, f)$ in increasing order.

5. We find the corresponding identities of the sorted descriptor and return them in the same order.

Once we have the result, we can take a look into two main metrics for the person retrieval task.

6.2.2.1 Cumulative Matching Characteristics (CMC)

Before we discuss cumulative matching characteristics, we explain two possible settings for the dataset:

- Single-gallery-shot setting: each gallery identify has only one instance (e.g. the CUHK03 dataset).
- Multi-gallery-shot setting: both query and gallery identity have more than one instance in their own dataset (e.g. the Market1501 dataset).

Consider a simple single-gallery-shot setting. For each query, the algorithm will rank all the gallery images according to their distance to the query in increasing order, then the CMC top-k accuracy is:

$$Acc_k = \begin{cases} 1 & \text{if top-k ranked gallery samples contain the query identity} \\ 0 & \text{otherwise} \end{cases} \tag{22}$$

which is a shifted step function. The final CMC curve is computed by averaging Equation 22 over all the queries. In this thesis, we applied a new splitting method [67] to the CUHK03 dataset like the Market1501 and the Duke-MTMC dataset did which originally are multi-gallery-shot setting to ensure the result is comparable. Under this setting, the query and gallery sets could have the same camera views, but

		Actual	
		Positive	Negative
Predicted	Positive	True Positive	False Positive
	Negative	False Negative	True Negative

Figure 72: Possible result for classification problem.

for each individual query identity, their gallery samples from the same camera are excluded.

6.2.2.2 Mean Average Precision

In order to explain the mean average precision (mAP), we will first need to introduce several related concepts. In the classification problem, we commonly use the following four terminologies to describe the result (even if the name is the same as the object detection one, the meaning is different here) shown as Figure 72.

- True positive, the model correctly predicts the positive class.
- True negative, the model correctly predicts the negative class.
- False positive, the model incorrectly predicts the positive class.
- False negative, the model incorrectly predicts the negative class.

The definition of these four concepts vary between the objection detection and ReID tasks. But the formula to calculate precision and recall remain the same Equation 16 for precision and Equation 17 for recall. With the knowledge about the P-R curve and AP (AUC) described in Section 6.2.1.2 and Section 6.2.1.3, we can finally give definition to mean average precision (mAP). It is the mean of average precision. Assuming we have N queries during the evaluation time then

$$mAP = \frac{AP}{N} = \frac{\int_0^1 p(r)dr}{N} \tag{23}$$

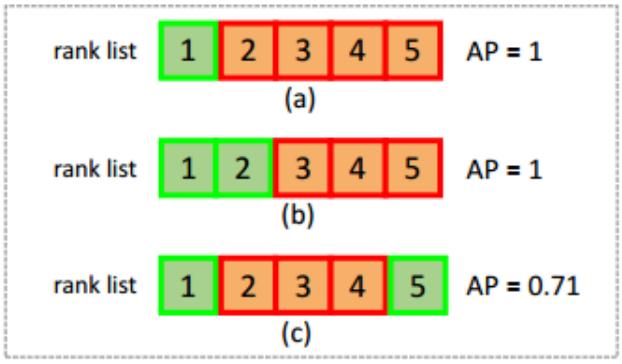


Figure 73: Simple example for the necessity of AP. The green box represents a match while red box means a mismatch. The CMC result for all three cases are 1, but AP will be [1, 1, 0.71] respectively.

We apply mAP to evaluate retrieval task firstly proposed in [64]. The author argued that in multi-gallery-shot setting, CMC cannot provide a fair comparison between two rank lists, a simple example can be found in Figure 73. Because that, for a ReID system, we would like to know how accurate the prediction can be, not just the accuracy among the top k result.

6.2.2.3 Dataset

Before we go into our experiment and results, let us take a look at the publicly available datasets in this domain. There is a total of three datasets: Market1501 [64], CUHK03 [26] and Duke-MTMC [44], which have been employed in our implementation. We are going to introduce their properties, shown as Table 6. We provide an abstraction of the dataset illustrated by Figure 74 and currently encapsulate these three for our model's training, testing and experiment.

6.2.2.4 Experimental Result

We perform comprehensive experiments on the person re-identification task since it is the main goal of this thesis. Let us walk through them one by one. In order to give the reader with a sense of how well our model perform, we provide the state-of-the-art result obtained from two common datasets shown as Table 7 and Table 8. By

dataset	subset	# pids	# images	# cameras
Market1501	train	751	12936	6
	query	750	3368	6
	gallery	751	15913	6
CUHK03-NP	train	767	7368	2
	query	700	1400	2
	gallery	700	5327	2
Duke-MTMC	train	702	16522	8
	query	702	2228	8
	gallery	1110	17661	8

Table 6: Statistic for three popular ReID datasets

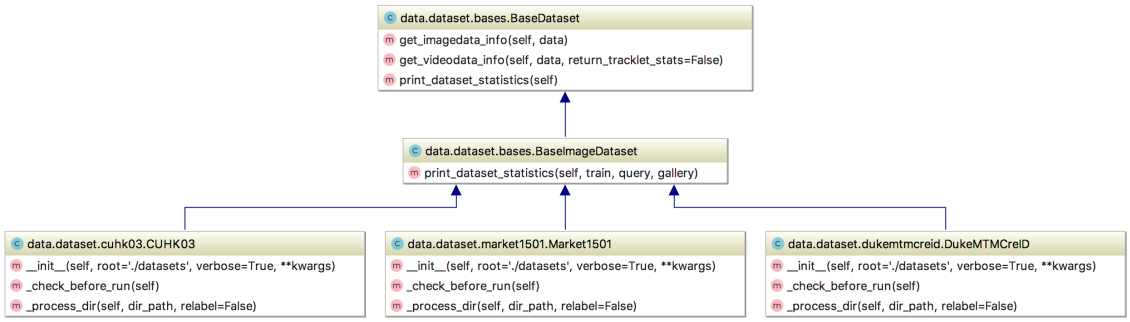


Figure 74: UML class diagram of ReID dataset abstraction

Rank	**Method**	**CMC**	**mAP**	**Year**
1	Auto-ReID	95.4	94.2	2019
2	DG-Net(RK)	95.4	92.49	2019
3	Parameter-Free Spatial Attention	94.7	91.7	2018
4	MGN	95.7	86.9	2018
6	DG-Net	94.8	86.0	2019
6	OSNet	94.8	84.9	2019
7	PCB + RPP	93.8	81.6	2017
8	PCB	92.3	77.4	2017
9	GLAD*	89.9	73.9	2017
10	Incremental Learning	89.3	71.8	2018

Table 7: State of the art result on Market1501 dataset [4].

Rank	Method	CMC	mAP	Year
1	Auto-ReID	91.4	89.2	2019
2	DG-Net(RK)	90.26	88.31	2019
3	Parameter-Free Spatial Attention	89.0	85.9	2018
4	MGN	88.7	78.4	2018
6	DG-Net	86.6	74.8	2019
6	OSNet	88.6	73.5	2019
7	PCB (RPP)	83.3	69.2	2017
8	PCB (UP)	81.8	66.1	2017
9	SVDNet + Random Erasing	79.3	62.4	2017
10	Incremental Learning	80.0	60.2	2018

Table 8: State of the art result on DukeMCMT-reid dataset [3].

observing these two tables, we find that all the results from Market1501 are better than that from DukeMCMT-reid which indicates the former dataset is in a sense easier than the latter.

Comparison between two different triplet loss functions: As mentioned in [23], there are two kinds of triplet loss functions Equation 5 and Equation 4. We do a comparison between these two loss functions and train the model with `setting 2` on the Market1501 dataset, the validation results can be summarized as Table 9. Surprisingly, triplet all loss perform better than triplet hard loss. That is in contrast with the result reported by [23]. So far, we don't have a sound explanation for it. Our guess is that it may be caused by the implementation where the paper's implementation is in Pytorch and ours is in TensorFlow. During the training CMC and mAP metrics, both got improved with time increasing which perform as our expectation. And the trend becomes more and more stable when the training time goes up.

Comparison between two environment settings: We also trained the model in two different settings in the same Market1501 dataset. The result are expressed in Table 10. Obviously, with the more powerful hardware in `setting 2`, the training time is much less than it on `setting 1`. Also, as we can see the results obtained from the same loss function are almost identical. The difference is limited to 0.001 which is under expectation. In such a computation task we cannot guarantee the result will

Triplet Loss	Epoch No.	CMC (top 5)	mAP
triplet all	40	[0.808, 0.877, 0.902, 0.902, 0.931]	0.597
	80	[0.898, 0.932, 0.949, 0.962, 0.966]	0.755
	120	[0.904, 0.941, 0.952, 0.964, 0.969]	0.769
triplet hard	40	[0.803, 0.871, 0.903, 0.920, 0.930]	0.602
	80	[0.868, 0.921, 0.945, 0.955, 0.962]	0.704
	120	[0.878, 0.924, 0.946, 0.955, 0.963]	0.715

Table 9: Validation result on the model trained with Market1501 dataset guided by two different loss functions.

Triplet Loss	Metric	Setting 1	Setting 2
triplet all	CMC (top 2)	[0.904, 0.947]	[0.904, 0.941]
	mAP	0.774	0.769
	training time (min)	244	133
triplet hard	CMC (top 2)	[0.871, 0.923]	[0.878, 0.924]
	mAP	0.705	0.709
	training time (min)	237	134

Table 10: Training result with two different settings on the same Market1501 dataset.

be exactly the same.

Comparison between the results obtained with the same dataset validation: As we mentioned in Section 6.2.2.3, we have an abstraction layer of the ReID dataset which enables us to train on various datasets with only a few modification (actually just passing different parameters). By making use of it, we perform training and validation on three different datasets with `setting 2`. We obtained the results shown by Table 11. From the results, we found that the performance on the CUHK03 dataset is poor. We thought that might be due to lesser training samples for the identity from distinct cameras (it has total two cameras only). And for the other two, since Market1501 is a little be easier than the DukeMTMC-reID dataset, the result we have currently is under expectation.

Comparison between results obtained from cross-dataset validation: Generalization is used to describe how well a model can handle an unseen style of data. In order to test the generalization capability of our models, we perform a cross-dataset validation experiment which means we train the model on *dataset*1

Dataset \Metrics	CMC (top 5)	mAP
Market1501	[0.904, 0.941, 0.952, 0.964, 0.969]	0.769
CUHK03	[0.502, 0.586, 0.641, 0.683, 0.721]	0.500
DukeMTMC-reID	[0.840, 0.884, 0.904, 0.919, 0.926]	0.704

Table 11: Training and validation result on three different datasets with ID and triplet all loss on `setting 2`.

Metrics / Dataset	M → D	D → M
CMC	[0.272, 0.339, 0.379, 0.404, 0.420]	[0.474, 0.548, 0.587, 0.618, 0.643]
mAP	0.150	0.211

Table 12: Cross-dataset validation result between Market1501 and DukeMTMC-reID dataset on `setting 2`. M → D represents the model trained on Market1501 dataset and tested on DukeMTMC-reID dataset.

then test it on *dataset*2. Since we have already found that the model trained with the CUHK03 dataset perform poor on this task from the previous comparison, we don't take it into consideration. The cross-dataset validation result is shown as Table 12 with the Market1501 and DukeMTMC-reID datasets. From the table, we found that the model trained on DukeMTMC-reID dataset can obtain a better generalization ability.

6.3 Summary

In this chapter, we described the evaluation methods we employed to examine the framework solution we proposed in Chapter 3 and Chapter 4. We first evaluated our two main models for person detection and person retrieval respectively with commonly used metrics in their domain, then we examined our framework by showing cases to prove that it meets the requirements we proposed in Section 1.5. In the next chapter, we will summarize our work, acknowledge the limitations we have and point out some potential research paths that can be done in the future.

Chapter 7

Conclusion and Future Work

In this chapter, we summarize what we have done within this thesis, and acknowledge the known limitations we have currently. At the end, we point out some potential research paths for other researchers who would like to follow up on our work.

7.1 Conclusion

In this thesis, we proposed a solution that can provide an abstract layer for various kinds of depth cameras and the functionality of tracking the same person across multiple cameras achieving the goal we set up in Section 1.4. The solution was designed and implemented in a software framework manner, since the key features of a framework perfectly fit to our need as explained in Section 3.1. Precisely, it contains a core framework which serves as infrastructure and three specialized frameworks for the detection, recognition and tracking tasks in general.

Within the framework's core, we provided modules for device abstraction, cross-language invocation, pipeline execution, framework-level common data structure and result visualization. As mentioned in Section 3.2, the device abstraction currently supports accessing data via a common API for three different kinds of cameras, Kinect v1, Kinect v2 and RealSense D435. The cross-language module allows the programs written in C/C++ to communicate with the one written in Python which is a common case for deep learning-based algorithms. The pipeline module enables the inversion

of control feature of our framework freeing the application developer from handling the complex but useless intermediate results. The common data structure defines the data format exchanged within the framework while the viewer module provides visualized results for the users.

For our specific demand, we instantiated the general detector, recognizer and tracker for person detection, person recognition and skeleton tracking tasks. For the detector instantiation described in Section 4.3.2, we re-trained the YOLO v3 network, reducing its scope from object detection to person detection achieving 76% mAP. For the person recognizer instantiation described in Section 4.3.3, we combined the identification model with the triplet model employing a ResNet-50 as backbone network to train a model achieving 90% top-1 accuracy. For the skeleton tracker instantiation described in Section 4.3.4, we ported the implementation from NiTE2 to our solution which can perform skeleton tracking in real-time without the use of a GPU.

To prove that our solution can satisfy the proposed scenarios listed in Section 1.5, we created concrete applications employing our framework instance in Chapter 5 and evaluated them in both the framework design and algorithm performance aspects. The result shown that all the requirements were fulfilled. In the context of performance, our solution, while keeping the real-time response requirement, can still achieve a comparable performance among the currently available approaches. The person detector is only 6% less than the original YOLO v3 in mAP metric and the person ReID model is ranked 9^{th} among all existing methods in the context of CMC and mAP metrics, more details were stated in Section 6.2.

7.2 Limitations

Even though the proposed solution in this thesis reach the goal we setup in Section 1.4 and fulfill the requirement we listed in Section 1.5, we have to admit that there are still a few limitations that currently exist in our solution:

- Our solution currently has not been tested with a real show yet.

- Our ReID application currently requires us to prepare an image database in advance and put them under a specific directory. There is no interface for the user to capture a database image on-the-fly.

- Our green screen application currently requires the user to select the filtering distance. Since we already have skeleton tracking, if we can combine them together we actually can achieve the green screen functionality automatically by making use of the depth values we get from the joint pixels.

- Our camera calibration application now can only calibrate color images, but most of the depth sensor have an IR camera. Since the image captured by the IR cameras is too bright, the corner detector cannot find the target easily. We should either add a pre-processing step in order to get a usable image or create different methods to calibrate the IR camera because the intrinsic and extrinsic matrices are useful for image alignment.

- Our current skeleton tracking application is based on a third-party library, we don't have skeleton extraction algorithm based on our framework's common data structure yet. It will restrict us that the skeleton tracking application can only apply to a subset of cameras which is not our original goal.

- Our current ReID model is mostly based on the appearance of the detected person. With such a model, it can work fine on a normal environment. But when put under some special environments like no or only dim lighting or tracking object which moves in a high speed, our model may likely fail.

- Our ReID model is currently trained on a single dataset which is minimally acceptable in order to measure our performance in an academic research context. But for real-life production we would need to focus on the performance by training the model on multiple datasets jointly to learn more generic features.

- Our framework requires a lot of dependencies, the environment configuration process for the framework developer is currently a little bit painful. We may need to develop some tools or scripts to help with the configuration.

7.3 Future Work

The research presented in this thesis constitutes a starting point for the OpenISS framework. The final goal is to make it can not only support our real-time performance production but also that it can serve as a research platform for people in computer vision, pattern recognition, deep learning and game development. Below, we list some additional features that we are working on or plan to work on.

Java API wrapper: As discuss in Section 4.1, our framework was written in C/C++, but our production ISSv2 was written in Java and on top of the Processing visual arts toolbox. In order to use OpenISS as the new back-end, we have to provide a Java wrapper for our APIs. This work is ongoing and partially done by our labmates Yuhao Mao, Jashanjot Singh and Chao Wang.

More devices support: At this moment, we only support three kinds of devices. But we always keep our eyes on the market, recently the new version of Kinect named Azure Kinect has been released, as well as two new cameras named D435i and T265 from RealSense. At the same time, RealSense also include OpenNI2 into their SDK which enable us to apply our skeleton tracking implementation described in **??** on all the RealSense cameras.

Comparison platform: One of our goal is to make OpenISS to serve as an algorithms comparison framework which defines a group of metrics for various research problem accordingly and enables the users to compare different algorithms under the a single controlled environment. Currently, we have the CMC and mAP metrics for the ReID task, we already achieved cross-datasets validation. We still plan to add more support for other tasks.

Full-platform support: Currently, our framework only works with Linux (Ubuntu distribution) and MacOS (without GPUs features, that is due to the hardware and their drivers limitation). We plan to add full support for Windows since it still the most popular OS in the world and most of our dependencies can be ported to it now.

Auto installation: The installation process our the framework currently is

manually and a little bit tricky. Some efforts have been made to automate the installations, but only for the Ubuntu System. We plan to script the installation process in CMake to enable dependencies downloading, building and installing automatically for all platforms.

Test with a real artistic show: Our solution currently has not been tested with a real live artistic show yet, we would like to integrate our solution with a real show to see how well it can perform during the performance.

Bibliography

[1] Frame rate. https://en.wikipedia.org/wiki/Frame_rate. Accessed: 2019-07-23.

[2] Software framework. https://en.wikipedia.org/wiki/Software_framework. Accessed: 2019-06-22.

[3] State of the art result on dukemcmt-reid dataset. https://paperswithcode.com/sota/person-re-identification-on-dukemtmc-reid. Accessed: 2019-06-29.

[4] State of the art result on market1501 dataset. https://paperswithcode.com/sota/person-re-identification-on-market-1501. Accessed: 2019-06-29.

[5] What is camera calibration? https://www.mathworks.com/help/vision/ug/camera-calibration.html. Accessed: 2019-05-29.

[6] Power score of different deep learning framework. https://towardsdatascience.com/deep-learning-framework-power-scores-2018-23607ddf297a, 2018.

[7] A Keras-YOLOv3 Team. A Keras implementation of YOLOv3 (Tensorflow backend). https://github.com/qqwweee/keras-yolo3, 2018.

[8] Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., Corrado, G. S., Davis, A., Dean, J., Devin, M., Ghemawat, S., Goodfellow, I., Harp, A., Irving, G., Isard, M., Jia, Y., Jozefowicz, R., Kaiser, L., Kudlur, M., Levenberg, J., Mané, D., Monga,

R., Moore, S., Murray, D., Olah, C., Schuster, M., Shlens, J., Steiner, B., Sutskever, I., Talwar, K., Tucker, P., Vanhoucke, V., Vasudevan, V., Viégas, F., Vinyals, O., Warden, P., Wattenberg, M., Wicke, M., Yu, Y., and Zheng, X. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org.

[9] Agarwal, S., Terrail, J. O. D., and Jurie, F. Recent advances in object detection in the age of deep convolutional neural networks. *CoRR abs/1809.03193* (2018).

[10] Ahmed, E., Jones, M., and Marks, T. K. An improved deep learning architecture for person re-identification. In *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (June 2015), pp. 3908–3916.

[11] Charles, P. Project title. https://github.com/charlespwd/project-title, 2013.

[12] Cheng, D. S., Cristani, M., Stoppa, M., Bazzani, L., and Murino, V. Custom pictorial structures for re-identification. In *Proceedings of the British Machine Vision Conference* (2011), BMVA Press, pp. 68.1–68.11. http://dx.doi.org/10.5244/C.25.68.

[13] Fan, X., Jiang, W., Luo, H., and Fei, M. Spherereid: Deep hypersphere manifold embedding for person re-identification. *CoRR abs/1807.00537* (2018).

[14] Galesso, S. Instance segmentation with mask r-cnn, 2017.

[15] Gheissari, N., Sebastian, T. B., and Hartley, R. Person reidentification using spatiotemporal appearance. In *2006 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'06)* (June 2006), vol. 2, pp. 1528–1535.

[16] Girshick, R. B. Fast R-CNN. *CoRR abs/1504.08083* (2015).

[17] Girshick, R. B., Donahue, J., Darrell, T., and Malik, J. Rich feature hierarchies for accurate object detection and semantic segmentation. *CoRR abs/1311.2524* (2013).

[18] Gray, D., and Tao, H. Viewpoint invariant pedestrian recognition with an ensemble of localized features. In *Computer Vision – ECCV 2008* (Berlin, Heidelberg, 2008), D. Forsyth, P. Torr, and A. Zisserman, Eds., Springer Berlin Heidelberg, pp. 262–275.

[19] He, K., Gkioxari, G., Dollár, P., and Girshick, R. B. Mask R-CNN. *CoRR abs/1703.06870* (2017).

[20] He, K., Zhang, X., Ren, S., and Sun, J. Spatial pyramid pooling in deep convolutional networks for visual recognition. *CoRR abs/1406.4729* (2014).

[21] He, K., Zhang, X., Ren, S., and Sun, J. Deep residual learning for image recognition. *CoRR abs/1512.03385* (2015).

[22] He, K., Zhang, X., Ren, S., and Sun, J. Identity mappings in deep residual networks. *CoRR abs/1603.05027* (2016).

[23] Hermans, A., Beyer, L., and Leibe, B. In defense of the triplet loss for person re-identification. *CoRR abs/1703.07737* (2017).

[24] Jin, H., Wang, X., Liao, S., and Li, S. Z. Deep person re-identification with improved embedding. *CoRR abs/1705.03332* (2017).

[25] Krizhevsky, A., Sutskever, I., and Hinton, G. E. Imagenet classification with deep convolutional neural networks. In *Proceedings of the 25th International Conference on Neural Information Processing Systems - Volume 1* (USA, 2012), NIPS'12, Curran Associates Inc., pp. 1097–1105.

[26] Li, W., Zhao, R., Xiao, T., and Wang, X. Deepreid: Deep filter pairing neural network for person re-identification. In *CVPR* (2014).

[27] Li, W., Zhao, R., Xiao, T., and Wang, X. Deepreid: Deep filter pairing neural network for person re-identification. In *2014 IEEE Conference on Computer Vision and Pattern Recognition* (June 2014), pp. 152–159.

[28] Lin, T., Goyal, P., Girshick, R. B., He, K., and Dollár, P. Focal loss for dense object detection. *CoRR abs/1708.02002* (2017).

[29] Liu, H., Feng, J., Qi, M., Jiang, J., and Yan, S. End-to-end comparative attention networks for person re-identification. *CoRR abs/1606.04404* (2016).

[30] Liu, L., Ouyang, W., Wang, X., Fieguth, P. W., Chen, J., Liu, X., and Pietikäinen, M. Deep learning for generic object detection: A survey. *CoRR abs/1809.02165* (2018).

[31] Liu, W., Anguelov, D., Erhan, D., Szegedy, C., Reed, S. E., Fu, C., and Berg, A. C. SSD: single shot multibox detector. *CoRR abs/1512.02325* (2015).

[32] Liu, X., Zhao, H., Tian, M., Sheng, L., Shao, J., Yi, S., Yan, J., and Wang, X. Hydraplus-net: Attentive deep features for pedestrian analysis. *CoRR abs/1709.09930* (2017).

[33] Luo, H., Gu, Y., Liao, X., Lai, S., and Jiang, W. Bag of tricks and A strong baseline for deep person re-identification. *CoRR abs/1903.07071* (2019).

[34] Mokhov, S. A., Song, M., Chilkaka, S., Das, Z., Zhang, J., Llewellyn, J., and Mudur, S. P. Agile forward-reverse requirements elicitation as a creative design process: a case study of llimitable Space System v2. *Journal of Integrated Design and Process Science* (2015–2016). Under review.

[35] Munaro, M., Basso, A., Fossati, A., Van Gool, L., and Menegatti, E. 3d reconstruction of freely moving persons for re-identification with a depth sensor. In *2014 IEEE International Conference on Robotics and Automation (ICRA)* (May 2014), pp. 4512–4519.

[36] Nanni, L., Ghidoni, S., and Brahnam, S. Handcrafted vs. non-handcrafted features for computer vision classification. *Pattern Recognition 71* (2017), 158 – 172.

[37] Neubeck, A., and Van Gool, L. Efficient non-maximum suppression. In *Proceedings of the 18th International Conference on Pattern Recognition - Volume 03* (Washington, DC, USA, 2006), ICPR '06, IEEE Computer Society, pp. 850–855.

[38] Pree, W. Meta patterns - a means for capturing the essentials of reusable object-oriented design. In *ECOOP* (1994).

[39] Redmon, J. Darknet: Open source neural networks in c. http://pjreddie.com/darknet/, 2013–2016.

[40] Redmon, J., Divvala, S. K., Girshick, R. B., and Farhadi, A. You only look once: Unified, real-time object detection. *CoRR abs/1506.02640* (2015).

[41] Redmon, J., and Farhadi, A. YOLO9000: better, faster, stronger. *CoRR abs/1612.08242* (2016).

[42] Redmon, J., and Farhadi, A. Yolov3: An incremental improvement. *CoRR abs/1804.02767* (2018).

[43] Ren, S., He, K., Girshick, R. B., and Sun, J. Faster R-CNN: towards real-time object detection with region proposal networks. *CoRR abs/1506.01497* (2015).

[44] Ristani, E., Solera, F., Zou, R., Cucchiara, R., and Tomasi, C. Performance measures and a data set for multi-target, multi-camera tracking. In *European Conference on Computer Vision workshop on Benchmarking Multi-Target Tracking* (2016).

[45] Schroff, F., Kalenichenko, D., and Philbin, J. Facenet: A unified embedding for face recognition and clustering. *CoRR abs/1503.03832* (2015).

[46] Song, M. Computer-assisted interactive documentary and performance arts in illimitable space. *CoRR abs/1212.6250* (2012).

[47] Song, M., and Mokhov, S. A. Dynamic motion-based background visualization for the *Ascension* dance with the ISS. [dance show, video], Jan. 2014. http://vimeo.com/85049604.

[48] Song, M., Mokhov, S. A., Mudur, S. P., and Bustros, J.-C. Demo: Towards historical sightseeing with an augmented reality interactive documentary app. In *Proceedings of the 2015 IEEE Games Entertainment Media Conference (GEM 2015)* (Oct. 2015), E. G. Bertozzi, B. Kapralos, N. D. Gershon, and J. R. Parker, Eds., IEEE, pp. 16–17.

[49] Song, M., Mokhov, S. A., Thomas, J., et al. Dynamic motion-based background visualization for the *Gray Zone* dance with the ISSv2. [dance show, video], Feb. 2015. https://vimeo.com/121177927.

[50] Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., and Salakhutdinov, R. Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research 15* (2014), 1929–1958.

[51] Su, C., Li, J., Zhang, S., Xing, J., Gao, W., and Tian, Q. Pose-driven deep convolutional model for person re-identification. *CoRR abs/1709.08325* (2017).

[52] Sun, Y., Zheng, L., Yang, Y., Tian, Q., and Wang, S. Beyond part models: Person retrieval with refined part pooling. *CoRR abs/1711.09349* (2017).

[53] Wei, L., Zhang, S., Yao, H., Gao, W., and Tian, Q. GLAD: global-local-alignment descriptor for pedestrian retrieval. *CoRR abs/1709.04329* (2017).

[54] Wen, Y., Zhang, K., Li, Z., and Qiao, Y. A discriminative feature learning approach for deep face recognition. In *Computer Vision - ECCV 2016 -*

14th European Conference, Amsterdam, The Netherlands, October 11-14, 2016, Proceedings, Part VII (2016), pp. 499–515.

[55] Wu, D., Zheng, S.-J., Zhang, X.-P., Yuan, C.-A., Cheng, F., Zhao, Y., Lin, Y.-J., Zhao, Z.-Q., Jiang, Y.-L., and Huang, D.-S. Deep learning-based methods for person re-identification: A comprehensive review. *Neurocomputing 337* (2019), 354 – 371.

[56] Wu, L., Shen, C., and van den Hengel, A. Deep linear discriminant analysis on fisher networks: A hybrid architecture for person re-identification. *CoRR abs/1606.01595* (2016).

[57] Wu, S., Chen, Y., Li, X., Wu, A., You, J., and Zheng, W. An enhanced deep feature representation for person re-identification. *CoRR abs/1604.07807* (2016).

[58] Xiao, T., Li, H., Ouyang, W., and Wang, X. Learning deep feature representations with domain guided dropout for person re-identification. *CoRR abs/1604.07528* (2016).

[59] Yi, D., Lei, Z., Liao, S., and Li, S. Z. Deep metric learning for person re-identification. In *2014 22nd International Conference on Pattern Recognition* (Aug 2014), pp. 34–39.

[60] Zhang, J., Bardakjian, S., Li, M., Song, M., Mokhov, S. A., Mudur, S. P., and Bustros, J.-C. Towards historical exploration of sites with an augmented reality interactive documentary prototype app. In *Proceedings of Appy Hour, SIGGRAPH'2015* (Aug. 2015), ACM.

[61] Zhang, Z. A flexible new technique for camera calibration. *IEEE Transactions on Pattern Analysis and Machine Intelligence 22* (December 2000), 1330–1334. MSR-TR-98-71, Updated March 25, 1999.

[62] Zhao, L., Li, X., Wang, J., and Zhuang, Y. Deeply-learned part-aligned representations for person re-identification. *CoRR abs/1707.07256* (2017).

[63] Zheng, L., Huang, Y., Lu, H., and Yang, Y. Pose invariant embedding for deep person re-identification. *CoRR abs/1701.07732* (2017).

[64] Zheng, L., Shen, L., Tian, L., Wang, S., Wang, J., and Tian, Q. Scalable person re-identification: A benchmark. In *Computer Vision, IEEE International Conference on* (2015).

[65] Zheng, Z., Zheng, L., and Yang, Y. A discriminatively learned CNN embedding for person re-identification. *CoRR abs/1611.05666* (2016).

[66] Zheng, Z., Zheng, L., and Yang, Y. Unlabeled samples generated by GAN improve the person re-identification baseline in vitro. *CoRR abs/1701.07717* (2017).

[67] Zhong, Z., Zheng, L., Cao, D., and Li, S. Re-ranking person re-identification with k-reciprocal encoding. In *CVPR* (2017).

[68] Zhong, Z., Zheng, L., Kang, G., Li, S., and Yang, Y. Random erasing data augmentation. *CoRR abs/1708.04896* (2017).

[69] Zhong, Z., Zheng, L., Zheng, Z., Li, S., and Yang, Y. Camera style adaptation for person re-identification. *CoRR abs/1711.10295* (2017).