

Általános információk, a diplomaterv szerkezete

A diplomaterv szerkezete a BME Villamosmérnöki és Informatikai Karán:

1. Diplomaterv feladatkiírás
2. Címoldal
3. Tartalomjegyzék
4. A diplomatervező nyilatkozata az önálló munkáról és az elektronikus adatok kezeléséről
5. Tartalmi összefoglaló magyarul és angolul
6. Bevezetés: a feladat értelmezése, a tervezés célja, a feladat indokoltsága, a diplomaterv felépítésének rövid összefoglalása
7. A feladatkiírás pontosítása és részletes elemzése
8. Előzmények (irodalomkutatás, hasonló alkotások), az ezekből levonható következtetések
9. A tervezés részletes leírása, a döntési lehetőségek értékelése és a választott megoldások indoklása
10. A megtervezett műszaki alkotás értékelése, kritikai elemzése, továbbfejlesztési lehetőségek
11. Esetleges köszönhetők
12. Részletes és pontos irodalomjegyzék
13. Függelék(ek)

Felhasználható a következő oldaltól kezdődő L^AT_EX-diplomatervsablon dokumentum tartalma.

A diplomaterv szabványos méretű A4-es lapokra kerüljön. Az oldalak tükörmagjával készüljenek (mindenhol 2,5 cm, baloldalon 1 cm-es kötéssel). Az alapértelmezett betűkészlet a 12 pontos Times New Roman, másfeles sorközzel, de ettől kismértékben el lehet tértíni, ill. más betűtípus használata is megengedett.

Minden oldalon – az első négy szerkezeti elem kivételével – szerepelnie kell az oldalszámnak.

A fejezeteket decimális beosztással kell ellátni. Az ábrákat a megfelelő helyre be kell illeszteni, fejezetenként decimális számmal és kifejező címmel kell ellátni. A fejezeteket decimális aláosztással számozzuk, maximálisan 3 aláosztás mélységen (pl. 2.3.4.1.). Az ábrákat, táblázatokat és képleteket célszerű fejezetenként külön számozni (pl. 2.4. ábra, 4.2. táblázat vagy képletnél (3.2)). A fejezetcímeket igazitsuk balra, a normál szövegnél viszont használunk sorkiegynítést. Az ábrákat, táblázatokat és a hozzájuk tartozó címet igazitsuk középre. A cím a jelölt rész alatt helyezkedjen el.

A képeket lehetőleg rajzoló programmal készítsék el, az egyenleteket egyenlet-szerkesztő segítségével írják le (A L^AT_EX ehhez kézenfekvő megoldásokat nyújt).

Az irodalomjegyzék szövegközi hivatkozása történhet sorszámozva (ez a preferált megoldás) vagy a Harvard-rendszerben (a szerző és az évszám megadásával). A teljes lista névsor szerinti sorrendben a szöveg végén szerepeljen (sorszámozott irodalmi hivatkozások esetén hivatkozási sorrendben). A szakirodalmi források címeit azonban mindenkor nyelven kell megadni, esetleg zárójelben a fordítással. A listában szereplő valamennyi publikációra hivatkozni kell a szövegben (a L^AT_EX-sablon a Bib^TEX segítségével mindenkor automatikusan kezeli). minden publikáció a szerzők után a következő adatok szerepelnek: folyóirat cikkeknél a pontos cím, a folyóirat címe, évfolyam, szám, oldalszám tól-ig. A folyóiratok címét csak akkor rövidítsük, ha azok nagyon közismertek vagy nagyon hosszúak. Internetes hivatkozások megadásakor fontos, hogy az elérési út előtt megadjuk az oldal tulajdonosát és tartalmát (mivel a link egy idő után akár elérhetetlenne is válhat), valamint az elérés időpontját.

Fontos:

- A szakdolgozatkészítő / diplomatervező nyilatkozata (a jelen sablonban szereplő szövegtartalommal) kötelező előírás, Karunkon ennek hiányában a szakdolgozat/diplomaterv nem bírálható és nem védehető!
- Mind a dolgozat, mind a melléklet maximálisan 15 MB méretű lehet!

Jó munkát, sikeres szakdolgozatkészítést, ill. diplomatervezést kívánunk!

FELADATKIÍRÁS

A feladatkiírást a tanszéki adminisztrációban lehet átvenni, és a leadott munkába eredeti, tanszéki pecséttel ellátott és a tanszékvezető által aláírt lapot kell belefűzni (ezen oldal helyett, ez az oldal csak útmutatás). Az elektronikusan feltöltött dolgozatban már nem kell beleszerkeszteni ezt a feladatkiírást.



Budapest University of Technology and Economics
Faculty of Electrical Engineering and Informatics
Department of Control Engineering and Information Technology

Comparison of convolution and transformer-based image processing neural networks

BACHELOR'S THESIS

Author
István Péter

Advisor
dr. Bálint Kiss
Ádám Gyula Nemes

December 8, 2022

Contents

Kivonat	i
Abstract	ii
1 Overview of the Literature	4
1.1 Metrics in object detection	4
1.2 Fully Convolutional Architectures	4
1.2.1 Two-stage detectors	4
1.2.2 One-Stage detectors	4
1.2.2.1 The YOLO architecture	4
1.3 Transformer based approached	4
1.3.1 The Transformer	4
1.3.2 Visual Transformers	5
1.3.2.1 The Detection Transformer	5
1.3.3 Explainability	5
1.4 Comparison	5
2 Practical Applications: Training on a Specific Task	6
2.1 Datasets	6
2.1.1 Preparing the filtered COCO dataset	6
3 Higher Order Applications: Multiple Object Tracking	8
3.1 The problem and possible approaches	8
3.2 Metrics [WORK IN PROGRESS]	9
3.3 SORT, theoretical background [WORK IN PROGRESS]	9
3.4 Choosing the data	9
3.5 Data Exploration	10
3.6 Benchmark	11
3.7 Designing the measurement	12
3.7.1 Running the detections	13

3.7.2	Calculating precision and recall	15
3.7.3	Running SORT	17
3.7.4	Calculating tracking performance	18
3.7.5	Calculting benchmark scores	18
3.8	Conclusion	18
	Bibliography	20

HALLGATÓI NYILATKOZAT

Alulírott *Péter István*, szigorló hallgató kijelentem, hogy ezt a szakdolgozatot meg nem engedett segítség nélkül, saját magam készítettem, csak a megadott forrásokat (szakirodalom, eszközök stb.) használtam fel. minden olyan részt, melyet szó szerint, vagy azonos értelemben, de átfogalmazva más forrásból átvettettem, egyértelműen, a forrás megadásával megjelöltem.

Hozzájárulok, hogy a jelen munkám alapadatait (szerző(k), cím, angol és magyar nyelvű tartalmi kivonat, készítés éve, konzulens(ek) neve) a BME VIK nyilvánosan hozzáférhető elektronikus formában, a munka teljes szövegét pedig az egyetem belső hálózatán keresztül (vagy autentikált felhasználók számára) közzétegye. Kijelentem, hogy a benyújtott munka és annak elektronikus verziója megegyezik. Dékáni engedéllyel titkosított diplomatervek esetén a dolgozat szövege csak 3 év eltelte után válik hozzáférhetővé.

Budapest, 2022. december 8.

Péter István
hallgató

Kivonat

Jelen dokumentum egy diplomaterv sablon, amely formai keretet ad a BME Villamosmérnöki és Informatikai Karán végző hallgatók által elkészítendő szakdolgozatnak és diplomaternevnek. A sablon használata opcionális. Ez a sablon L^AT_EX alapú, a *TeXLive* T_EX-implementációval és a PDF-L^AT_EX fordítóval működőképes.

Abstract

This document is a L^AT_EX-based skeleton for BSc/MSc theses of students at the Electrical Engineering and Informatics Faculty, Budapest University of Technology and Economics. The usage of this skeleton is optional. It has been tested with the *TeXLive* T_EX implementation, and it requires the PDF-L^AT_EX compiler.

Introduction

Computer vision and its subtasks

Computer Vision is a branch of Artificial Intelligence aimed at deducing scene understanding and other higher-order information from visual input like images, videos, or more specialised sensor data like LIDAR point clouds etc. Possible subtasks in Computer Vision are classification of images, detection of objects in images and video, segmentation, pose estimation of specific entities, tracking etc. This thesis focuses on two specific tasks: object detection and multiple object tracking. Both are widely discussed topics, having decades of research behind them.

In short, object detection has the aim to find all relevant objects on images, usually indicating their location as bounding boxes that fit the contours of the object. Since the mid 2010s, deep neural networks saw an exponential rise in popularity and performance, by fixing numerical problems that previously plagued these kinds of machine learning models, and having access to a larger amount of data and more computing power than ever before. Particularly in vision applications, convolutional filters used to find certain features could be adapted as neural network layers, and the filters themselves became trainable parameters. This new approach gave rise to the VGG and AlexNet architectures, among others, kicking off the era of Convolutional Neural Networks (CNN). Today, almost all intelligent computer vision applications use some sorts of CNNs, so it is natural that the most popular and performant object detectors are also built on CNNs. Previously, systems using hand-crafted features like the Deformable Parts Model (DPM) were widely used.

Multiple object tracking works on videos, and aims to determine whole trajectories of objects of interest. The main challenge for multiple object trackers besides detection is preserving object identity, guaranteeing that the same object was tracked for the whole duration of the trajectory.

Goal and motivation

This thesis was made with the goal of comparing two paradigms in modern object detection architectures: the older, more established, Fully Convolutional Network-based (FCN) detectors versus the relatively recent Transformer-based implementations.

The comparison itself, considering only the classic detection metrics used in the field, is an active topic in the research community that spawned a myriad of analyses, varying in depth and scope. My goal wasn't to make yet another analysis of the kind, I wanted rather to measure performance in another computer vision task where results of the object detection are merely inputs to the solution, but the quality of the detection can make or

break the performance of the whole system. This task of my choice was multiple object tracking (MOT).

In the Project Laboratory course of the previous semester, I implemented a version of a well-known multiple object tracker called the Simple Online Realtime Tracker (SORT) for a specific task involving multiple fisheye cameras placed in a parking space. In that context, the object detector, that was based on the You Only Look Once (YOLO) architecture, was considered as a given, provided to me by the company that hosted my topic.

Starting from that setup, and following the suggestion of my advisor at the company, I considered to extend the scope of my examination of the field by comparing multiple possible object detectors, with special emphasis on a new emerging approach: using the Transformer architecture in vision tasks.

For the new project, I have chosen to work on public MOT databases, but staying in the topic of road traffic analysis. I worked with the UA-DETRAC database, that consists of videos taken from fixed camera positions over highways and highly frequented roads. For the tracker, I kept using SORT due to it being a simple yet highly performant solution. Moving away from the custom implementation, I chose to use the official implementation written by the authors of the SORT research paper. I evaluated the tracker with seven detectors, two Detection Transformer (DETR) and five YOLOv5 variants, the main difference between models of the same type being model size, as in number of parameters, which influences inference time as well. The benchmarking system for the UA-DETRAC database is currently not available, so I had to write my own implementation of the benchmark, based on the metrics proposed by the paper that introduced the UA-DETRAC benchmark itself. In this thesis, I will present the final results of this assessment.

As for the theoretical parts, I gave a brief overview of the model architectures, summarizing the research papers that led to the development of the two most popular branches of object detectors. I analyze available implementations of particular models, exploring the available possibilities to train on custom data, called transfer learning. I also review popular training datasets, specially the Microsoft Common Objects in Context (MS COCO or simply COCO) dataset and benchmark, considered the most popular, large-scale and richly annotated training database for object detection, classification and semantic segmentation.

Thesis structure

The first chapter of the thesis contains a more detailed introduction to the problem of object detection, focusing on standard metrics for assessment as well. After that, I review the modern fully convolutional object detectors, differentiating between two-stage and one-stage detectors, but only going into details in the case of the latter. Among the one-stage detectors, I chose the YOLO architecture for exploration, following its main versions, ideas, and overall architecture. Moving away from FCNs, I summarize the history of the Transformer, first introduced in natural language processing, its adaptation to computer vision and finally the DETR, the first end-to-end object detector, where the separate non-maximum suppression step from FCNs is replaced by a novel training loss.

The second chapter is the overview of available implementations in the most popular machine learning frameworks, as well as training databases for object detection, and the possibility of training the implementations on custom data, a common use case in the industry.

The third, largest chapter contains a theoretical overview of the MOT problem, citing a recent literature review. Then I explore the chosen benchmark and dataset, the UA-DETRAC, detailing the metrics used for evaluation. Then, I summarize the theoretical background of the SORT architecture, followed by the description of my own work: the measurement, its implementation and results.

Chapter 1

Overview of the Literature

In this chapter I am going to review the theoretical background for the two competing paradigms I cover: the fully convolutional, one-stage detector, whose most prominent variant is the You Only Look Once (YOLO) architecture, and the Transformer-based Detectection Transformer (DETR). For the former, I will explain in some detail chosing it over its competitors of the same kind, for example the Single Shot Detector (SSD).

In the case of the Tranformer-based category, I chose, for the sake of simplicity, the DETR architecture over its later successors, like DINO or Deformable DETR. The changes introduced in **its paper (insert citation)** are important enough to be discussed on their own, but I will mention the improvements achieved by the successors whenever the state-of-the-art is concerned.

Likewise, I have chosen the YOLOv5 for in-depth comparison as the DETR's counterpart, mainly because it is a contemporary of the latter (both being introduced in 2020), but mentioning the latest improvements introduced by YOLOv7 as well.

1.1 Metrics in object detection

1.2 Fully Convolutional Architectures

1.2.1 Two-stage detectors

1.2.2 One-Stage detectors

1.2.2.1 The YOLO architecture

1.3 Transformer based approached

1.3.1 The Transformer

The Transformer architecture has been introduced in the *Attention is All You Need* [5] paper in 2017, originally intended for Natural Language Processing (NLP) tasks, more specifically sequence transduction problems, like translation.

At the time, the attention mechanism and some variants of the encoder-decoder architecture was already widely used in the state-of-the-art, along with convolutional layers, Long

Short Term Memory (LSTM) cells or Gated Recurrent Units (GRU). The Transformer was a successful attempt at replacing the latter three with trainable versions of the attention mechanisms called *Multi-Head Attention*.

In the Transformer model, the bulk of the learning happens at the weights of the linear transformations that establish the **heads** of the Attention layers, as the Attention layer itself does only mathematical operations on its input.

The article mentions that attention mechanisms and encoder-decoder based architectures have already been used at the time in the state-of-the art models. The novelty of the Transformer was getting rid of the convolutional, or traditionally recurrent components, and relying almost solely on the attention mechanism, namely a slightly modified version of it: the *multi-head self-attention*.

1.3.2 Visual Transformers

1.3.2.1 The Detection Transformer

The main advantage of the Detection Transformer is its capacity for every region to attend to every other region. In the fully convolutional case, this is done with hierarchical convolutions that together define large *receptive fields*.

1.3.3 Explainability

1.4 Comparison

Chapter 2

Practical Applications: Training on a Specific Task

2.1 Datasets

The Microsoft COCO: Common Objects in Context is an immensely popular dataset for numerous computer vision tasks (classification, object detection, instance segmentation, collectively called object recognition). It was introduced in 2014 in the paper *Microsoft COCO: Common Objects in Context* [3], aiming to improve upon already existent visual datasets like ImageNet and PASCAL, and striving to be a benchmark of scene understanding. It contains labeled data for 91 object classes, captured in their natural habitat (hence *context*). It was later updated in 2017, a notable change being the introduction of *stuff* labels (among the already existent *thing* labels), for objects with no clear boundaries, like sky and grass. These were introduced as panoptic segmentation labels.

The official site of the dataset¹ mentions that the creators of COCO have partnered with the developers of open source tool FiftyOne in providing a software to facilitate downloading, visualizing, and evaluation on COCO, so I will be using it for initial data exploration. Another tool endorsed by the creators is the COCO API.

I conducted initial data exploration on the COCO dataset with the FiftyOne tool (launching an application session from a Python shell as described in their tutorials), as seen in figure 2.1. The success of the COCO dataset lies in its richness and difficulty: some training images, as seen in figure 2.2, are caught in so called *non-iconic views*, where visual features can be ambiguous and they should be interpreted in context to recognise the object.

2.1.1 Preparing the filtered COCO dataset

¹<https://cocodataset.org>

Figure 2.1: Exploring the train split of the MS COCO 2017 dataset in the FiftyOne tool. There are 43867 instances of labeled cars and 9973 instances of labeled trucks, with 14714 images containing either of them.

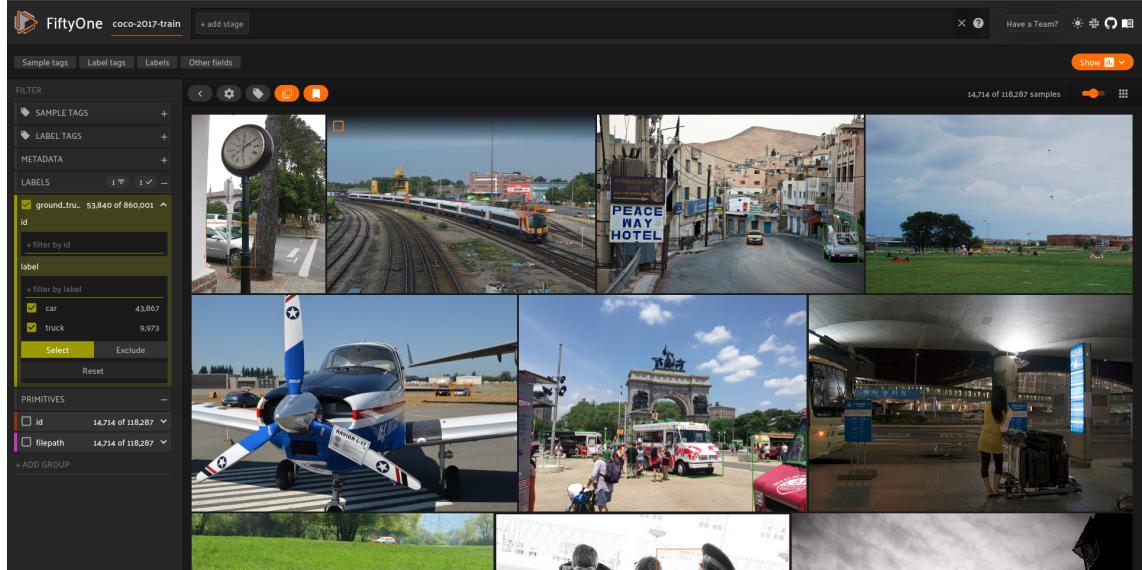


Figure 2.2: Some difficult examples: objects in overexposed regions of the image, crowd labels consisting of very small (e.g aerial) images of cars (orange) and trucks(green), detection based on a skewed reflection of the object, small parts of it.



Chapter 3

Higher Order Applications: Multiple Object Tracking

As the final aim of my thesis, I am going to inspect the performance of the chosen object detectors on a higher-order problem, namely multiple object tracking. First I am going to review the problem itself, and the common metrics used to measure its performance.

After this, I will elaborate on the object tracking method I chose as the basis, showing how it incorporates the aforementioned detectors, and the way the performance of the latter is expected to influence tracking performance.

3.1 The problem and possible approaches

Multiple object tracking (MOT) consists of determining the *trajectory* of given kinds of objects in a video stream. In the current context, the trajectory of a single, unique object is a series of bounding boxes with an identifier, one box for every frame in which the object is visible. The rectangular bounding boxes are expected to fit the object silhouette as tightly as possible.

Along the years, several approaches have been developed for the problem, but with the advent of highly accurate object detectors (most notably the breakthrough of convolutional neural networks from the mid-2010s and onward), the detection-based paradigm has become one of the most popular.

The approach consists of detecting objects on each new frame, then assigning them to previously found trajectories, if the object has been seen before, or registering a new trajectory otherwise. This is generally called the *association step*. In some scenarios, when the objects of interest are densely packed, this can pose a considerable difficulty.

Among some popular solutions to this association problem are the *proximity-based* (the Simple Online and Realtime Tracking (SORT) architecture [1] is one example) and the *feature-based* (the DeepSORT architecture [7] incorporates such methods using *deep appearance descriptors*) assignments. The former considers each already existent trajectory, takes their previous location, or the predicted location for this time step, and does the assignment based on some spatial distance between these and current detections. The latter does overarching associations based on visual appearance features, not necessarily restricted to the appearance observed in the last few frames.

The weakness of the proximity-based approach is when the density of objects is high, their movement highly dynamic and their trajectories intertwined. In this case, feature-based methods might excel. The weakness of the feature-based methods might show when they confuse similar, but distinct objects, or when they cannot account for some drastic, fast changes in appearance (like changes in lighting, orientation, deforming, or partial occlusion). In these situations, assignment to last known locations can help. Thus, these two approaches are not, and should not be exclusive.

The introduction above focuses on paradigms and aspects mostly relevant for my thesis, and narrows down the field somewhat. For a comprehensive, recent overview of the problem, see the literature review at [4]. It goes beyond just the detection-based approach, and also formalises the general problem.

Finally, it is worth mentioning that the most common multi-object tracking targets are pedestrians, faces and vehicles, and thus most popular MOT datasets consist of these. In this thesis, I am going to tackle tracking vehicles in road traffic footages.

3.2 Metrics [WORK IN PROGRESS]

When it comes to define metrics to measure a tracking solution's performance, there does not seem to be one trivial, immediately obvious score that can quantify the overall adequacy. When choosing metrics, I had to look for what the most well-known and respected benchmarks measured.

One of, if not the most popular multiple object tracking benchmark is the MOTChallenge¹, introduced in [2].

TODO: The MOT15 data format.

3.3 SORT, theoretical background [WORK IN PROGRESS]

3.4 Choosing the data

I will evaluate the model's performance for multi-object tracking on the UA-DETRAC dataset². The dataset contains 100 videos (60 for training, 40 for testing) of road traffic captured at different locations in China. The total length of the video footage is around 10 hours, stored frame by frame (as separate 960 pixel by 540 pixel JPEG images), at the rate of 25 frames per second.

The annotations contain information about vehicle type, illumination, scale (proportional to the square root of the bounding box area), occlusion ratio (the measure by which other objects occlude the vehicle) and truncation ratio (the degree of the bounding box lying outside the frame). Information about weather conditions e.g. rainy, cloudy, sunny etc. is also included.

¹<https://motchallenge.net/>

²<https://detrac-db.rit.albany.edu/>

3.5 Data Exploration

At the time of writing this thesis, the test and train images, grouped into sequences that form videos, can be accessed through the Download page of the official site³ as DETRAC-train-data.zip. However, the tracking annotations for the train and test sets cannot be downloaded, as clicking on the links triggers a popup prompting to log in first. As the login functionality currently does not work, I had to look for alternative ways to access the data.

Fortunately, after a short search I have found a GitHub repository owned by the Georgia Tech Database Group. Their Exploratory Video Analytics System (EVA) repository contains, among others, a guide on how to download the UA-DETRAC dataset⁴, along with a bash script serving the same purpose.

Through those links, I could download the training annotations. Sadly, the test annotations, claimed on the official UA-DETRAC website to be released, were still nowhere to be found, but I figured the 60 sequences (or even a subset of them) should be enough to evaluate tracking performance. The integrity of the measurement wouldn't have been compromised either, as I wasn't planning on doing detector training or hyperparameter tuning on the train set. There were two kinds of training annotation formats provided: XML and MAT.

The XML annotations are meant for detection training, and contain additional information like vehicle category, weather conditions during filming and bounding box scale. I did not use this data directly, as I used the models pre-trained on the COCO dataset, but inspecting this data confirms the vehicle categories supported by the DETRAC dataset: *car, van, bus, other*. The corresponding COCO object categories are *car, truck, bus*, so that is what I will be looking for when running detection on the images, and ignoring all other classes.

The MAT annotations are files in MATLAB serialization format, containing trajectory and position information for all tracked entities. For every image sequence, there is a MVI_NNNNN.MAT file. The image sequences themselves are consecutive frames under Insight-MVT_annotation_Train/MVI_NNNNN, after unzipping DETRAC-train-data.zip.

Initially I inspected the MAT files' inner structure (see figure 3.1) in GNU Octave, MATLAB's open-source and somewhat compatible counterpart. I found that each file contained 5 matrices:

1. X : An $N \times T$ matrix, where T is the number of trajectories, and N is the number of frames. Given the frame i and trajectory j , $x_{i,j}$ denotes the x coordinate of the bottom center of the bounding box⁵, or 0 if trajectory j is not present in that frame.
2. Y : An $N \times T$ matrix, similar to X , but it contains the y coordinates of the bounding boxes' foot position.
3. W : contains the width of the boxes.
4. H : contains the height of the boxes.
5. $frameNums$: A row vector of length N , containing the 1-based indices of the frames.

³<https://detrac-db.rit.albany.edu/download>

⁴https://github.com/georgia-tech-db/eva/tree/master/data/ua_detrac

⁵I found this out initially through trial and error, when trying to visualize bounding boxes in Python, because I did not know this to be a common format for specifying bounding boxes. Later, I found it mentioned in <https://detrac-db.rit.albany.edu/FAQ> as *foot position*.

The screenshot shows two Octave windows. The top window is titled 'gtInfo [1x1 struct]' and displays a table of field names and their sizes: X [664x52 double], Y [664x52 double], H [664x52 double], W [664x52 double], and frameNums [1x664 double]. The bottom window is titled 'gtInfo.X [664x52 double]' and shows a 664x52 table. The columns are labeled 1 through 9 and the rows are labeled 1 through 4. The data in the table is as follows:

	1	2	3	4	5	6	7	8	9
1	673	581.5	562.5	522	568	757	931.5	0	0
2	675	582	563	522	568	756	928	0	0
3	680.5	582.5	563	522	569	754.5	923	0	0
4	682	583	563.5	522.5	569	752	921.5	0	0
4									

Figure 3.1: Label data exploration in octave

3.6 Benchmark

The dataset and the benchmark is described in [6]. The article also proposes an evaluation protocol for multi-object tracking. A key point is the joint analysis of detection and tracking performance, analysing the effects of the chosen model's precision/recall values (and the underlying confidence threshold setting that influences both) in relation with the tracking performance, as reflected by the MOTA and MOTP score. These relationships are visualized on the PR-MOTA and PR-MOTP curves (See figure 3.2).

The authors argue that, as it is not fair, nor enough to compare the performance of two object detectors based on different points on the PR curve, it is also not enough to determine the maximum point on the PR-MOTA curve, as a good tracker must produce good scores in a wider range of settings. The whole range of the curve must be taken into account in some form, thus the need for a new metric, Ω^* , or the *PR-MOTA score*:

$$\Omega^* = \frac{1}{2} \int_{\mathcal{C}} \Psi(p, r) ds$$

where Ψ is the MOTA score across the whole dataset at precision p and recall r , and we calculate the (approximate value of the) signed area under the PR-MOTA curve as an integral along the PR curve \mathcal{C} (for every $(p, r) \in \mathcal{C}$). Dividing by 2 ensures that the score stays in the interval $(-\infty, 100\%]$. Similar metrics can be defined for the MOTP, FP, FN, IDS, MT and ML scores.

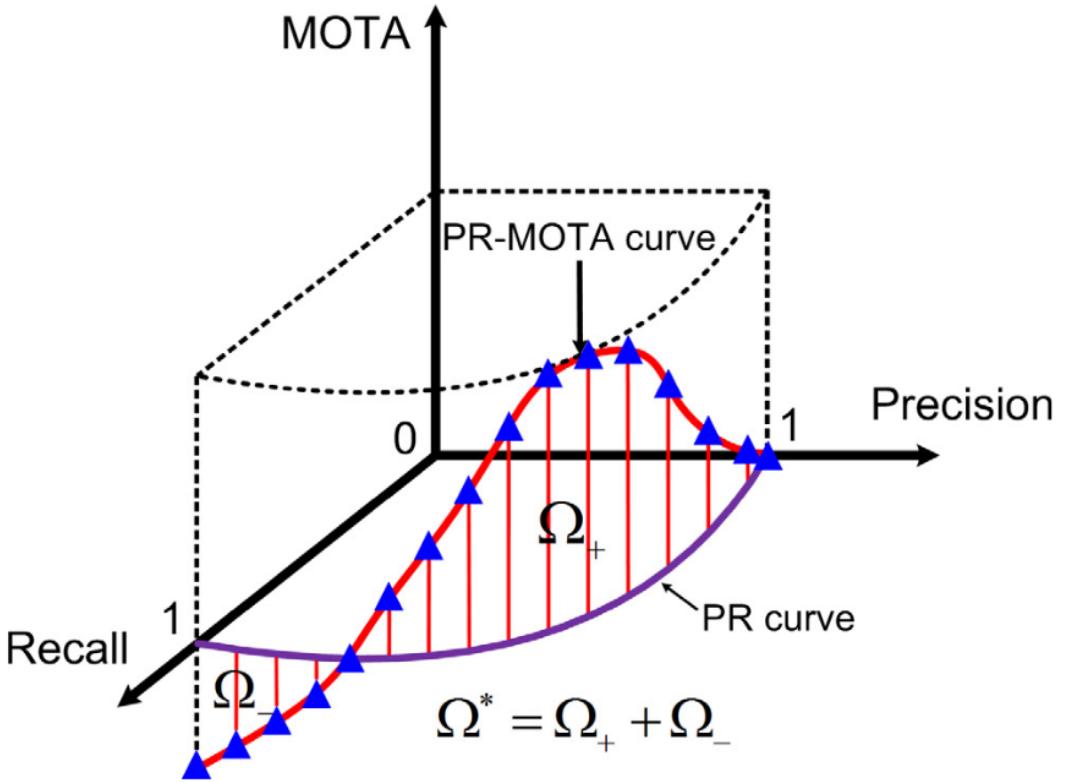


Figure 3.2: Visualization of the PR-MOTA curve. Image taken from [6]

3.7 Designing the measurement

For the measurement, I chose 7 detectors in total: two DETR variants, one with the ResNet50 and one with the ResNet101 backbone, pre-trained on the COCO 2017 dataset. The other 5 were the YOLOv5n (nano), YOLOv5s (small), YOLOv5m (medium), YOLOv5l (large), YOLOv5x (extra large). These differ in the size, as in number of parameters, and this influences the inference times as well.

Each detector was evaluated at 10 confidence thresholds: from 0.0 to 0.9, incremented by 0.1.

To calculate the value defined by the UA-DETRAC benchmark, I had to break the process into multiple steps and assess the best tool for each subtask.

As input data, I had the UA-DETRAC training videos, as frame sequences, and the annotations in MATLAB format.

For my tracker of choice, I could clone the public GitHub repository⁶, and run the tracker, provided I put the detections in MOT2015 format for each sequence under the data folder.

The rest of the steps had to be implemented by me. First, I had to create a model executor that returned the detections from each frame of the video sequences, and served as a adapter-wrapper around both the DETR and the YOLOv5 models, to hide the differences in preprocessing. I invoked this this on all combinations of models, confidence thresholds

⁶<https://github.com/abewley/sort>

and sequences. For each unique trio of these, one file has been saved in the MOT15 detection format.

The detections themselves, along with the annotations were enough to calculate the precision-recall curve for each model, aggregating the values for all sequences. For evaluation I used the standard IoU metric for ground truth-prediction distance, with IoU threshold 0.7 (In the code, `max_iou` is set to 0.3, because the library I used considers $1 - IoU$ when matching ground truth with prediction. More on that in the relevant subsection). This is not to be confused with the confidence threshold, which is a property of the detector that can be set, and varies. This is a global setting which means that a good prediction can be considered a match to the ground truth box only if their IoU distance is at least 0.7, a reasonable expectation according to my visual intuition.

Next, I ran the SORT script provided in the official repository for each model separately, on every sequence and confidence threshold combination. These were automatically saved in the MOT15 tracking annotation format.

Having the tracking output and the ground truth annotations, I ran the tracking evaluation on every model and confidence threshold combination, aggregating MOTA values for all sequences. The evaluation IoU threshold was set to 0.7, like at the detection evaluation.

Given both the precision-recall curves for every model (where both precision and recall are a function of the confidence threshold), and the MOTA values for every model and confidence threshold combination, I could finally calculate the PR-MOTA scores for each model.

A visual overview of the process above can be seen in figure 3.3. Detailing of the individual design choices made at each step can be found in the following subsections. At some of the steps, I implemented visualizations as a way to spot possible errors in implementation that could compromise the whole measurement.

The language I used was Python 3.10, the inference times were recorded under Linux, on a desktop PC with an AMD Ryzen 3800X processor, 32 Gigabytes of RAM and an RTX3060Ti graphics card with 8 Gigabytes of VRAM.

3.7.1 Running the detections

In order to evaluate the aforementioned models on images, I had to find the easiest way to load the pretrained weights for each model.

For the DETR, the `transformers` Python package by Huggingface⁷ provides the easiest API, with simple pre- and postprocessing (including confidence thresholding and converting from logits to probability distributions), and also provides a way to download pretrained weights. Under the hood, `transformers` uses a PyTorch `torch.nn.Module`, so sending it to the GPU is similar to all Torch modules. Also theoretically loading from Torch Hub was also a possibility, but the pre- and postprocessing tasks would have been messier.

As for the YOLOv5 model, it can be loaded directly from Torch Hub, and its input format is way more flexible, so lists of images (for the inference, I used batches of size 1), but even image links can be fed to it directly.

⁷<https://huggingface.co/docs/transformers/index>

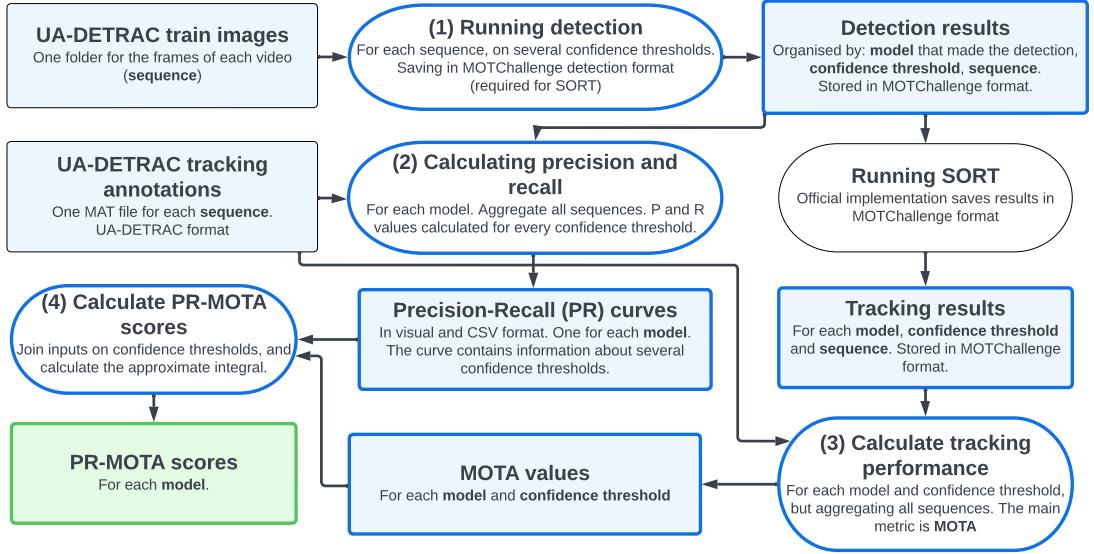


Figure 3.3: Flowchart visualization of my workflow. The blue rectangles denote data, rounded rectangles denote processing steps. Blue outline means that part of the workflow was implemented by me, or the data was generated in the process. No blue outline means either the data is from an external source, or the processing step involved using entirely external code.

To hide the differences between these two approaches, I implemented an adapter class for them, and also added visualization capabilities to spot possible errors in the interpretation of data (for example, assuming wrong bounding box format).

I also wanted to measure average inference times to show a picture about the real time capabilities of each detector. All inference happened on the GPU, so due to the asynchronous nature of the Torch CUDA operations, measuring elapsed time on CPU was out of the question. The appropriate time measuring method, as suggested on the official Torch forums⁸ was through the `torch.cuda.Event` object. I took the elapsed GPU time between the moment of feeding the inputs until receiving the logits, but this didn't include copy times between the RAM and VRAM, neither the postprocessing steps (calculating probability distributions with softmax, confidence thresholding).

Timing results can be seen in table 3.1. All code to reproduce the results can be found on the GitHub repository of my thesis⁹.

	yolov5n	yolov5s	yolov5m	yolov5l	yolov5x	detr-resnet-50	detr-resnet-101
avg (ms)	7.38	7.38	8.99	12.00	20.88	43.98	64.53
std (ms)	1.16	0.24	0.24	0.18	0.27	0.80	0.66
FPS	135.5	135.5	111.2	83.3	47.89	22.73	15.49

Table 3.1: Average inference times on GPU and their standard deviation (expressed in milliseconds) for each model, after running on every sequence.

⁸<https://discuss.pytorch.org/t/how-to-measure-time-in-pytorch/26964>

⁹See <https://github.com/peter-i-istvan/bsc-thesis>, under tracking/run_detection.py.

3.7.2 Calculating precision and recall

Next, I had to calculate precision and recall curves for the detections. To do this, for every model and confidence threshold pair, I had to sum all true positives, false positives and false negatives in every sequence. For the theoretical definition of these, not only in the general meaning, but in the case of object detection, where assignment between ground truth and predicted boxes is not always trivial (the Hungarian bipartite graph matching algorithm solves this ambiguity), see chapter one.

For the concrete implementation, I took a ‘sideways’ approach, but one that could be mostly reused later in calculating the MOTA score as well. The `motmetrics` Python package¹⁰ provides handy tools for calculating MOT metrics, like those in CLEAR MOT. Specifically, it gives us an *accumulator* object that must be initialized before evaluating on a sequence, then for each frame, its `update()` function must be called to update its state. In the arguments, one must provide ground truth IDs and predicted IDs, as well as a distance matrix that expresses the distance between ground truth box i and prediction box j . In theory, `motmetrics` is distance-agnostic, which means that any kind of distance can be supplied, but I used the IoU box distance metric, which is one of the most popular evaluation metrics both for object detection and multiple object tracking. `motmetrics` provides functions for calculating all popular metrics, so incorporating IoU calculation into my code was easy.

In its update step, the accumulator takes the input arguments, and in the first parts, it is not at all concerned with the IDs provided, but calculates the best ground truth box-to-prediction assignment with the Hungarian algorithm. This is the very exact action that is needed at detection evaluation as well, this is why we ‘piggy-backed’ tracking evaluation. Based on this assignment, it generates MATCH, MISS and FP events, which correspond to true positives, false negatives and false positives respectively.

In the next steps, if we provided valid prediction IDs (a.k.a our hypotheses for the identities of the tracked objects), the accumulator would also record possible mistakes with regard to identity confusion, like identity swaps. However, considering that the MOTChallenge format defines -1 as the ID field when saving only detections, not tracking results, technically we tell the accumulator that all found boxes correspond to our object -1, which makes the rest of this step absolute rubbish, but it does not influence the events already recorded in the first step, and in the first steps of all future update calls.

After finishing a sequence, the script extracts the number of relevant events, and adds them to all the previously calculated occurrences of true positives, false negatives and false positives for that model-confidence threshold pair. At the end of this, we will have a mapping counting all of these for every sequence. Then I iterate through all model-confidence threshold pairs, calculate the precision and recall values given by the formulas:

$$P = \frac{TP}{TP + FP}$$

$$R = \frac{TP}{TP + FN}$$

Finally, I save them as CSV files. The result can be seen in tables 3.2 and 3.3.

¹⁰<https://pypi.org/project/motmetrics/>

CT	D101	D50	YL	YM	YN	YS	YX
0.9	0.52	0.47	0.93	1.0	1.0	0.90	0.64
0.8	0.45	0.41	0.54	0.61	0.83	0.61	0.49
0.7	0.40	0.37	0.54	0.51	0.68	0.52	0.46
0.6	0.36	0.33	0.51	0.46	0.53	0.45	0.44
0.5	0.33	0.30	0.46	0.42	0.42	0.39	0.41
0.4	0.30	0.28	0.40	0.37	0.34	0.32	0.36
0.3	0.28	0.26	0.33	0.31	0.28	0.27	0.30
0.2	0.25	0.23	0.30	0.28	0.25	0.24	0.27
0.1	0.21	0.21	0.30	0.28	0.25	0.24	0.27
0.0	0.19	0.18	0.30	0.28	0.25	0.24	0.27

Table 3.2: The precision curve of the models. CT is confidence threshold, D101 and D50 is DETR-Resnet 101 and 50, while YL, YM, YN etc. are the yolo versions.

CT	D101	D50	YL	YM	YN	YS	YX
0.9	0.63	0.57	0.03	0.01	0.00	0.02	0.12
0.8	0.69	0.65	0.41	0.34	0.13	0.36	0.46
0.7	0.71	0.68	0.59	0.52	0.41	0.56	0.60
0.6	0.73	0.69	0.69	0.63	0.52	0.66	0.69
0.5	0.73	0.71	0.76	0.72	0.60	0.72	0.76
0.4	0.74	0.71	0.81	0.79	0.66	0.77	0.81
0.3	0.75	0.72	0.84	0.84	0.70	0.80	0.84
0.2	0.75	0.72	0.86	0.86	0.72	0.81	0.86
0.1	0.76	0.73	0.86	0.86	0.72	0.81	0.86
0.0	0.76	0.73	0.86	0.86	0.72	0.81	0.86

Table 3.3: The recall curve of the models. CT is confidence threshold, D101 and D50 is DETR-Resnet 101 and 50, while YL, YM, YN etc. are the yolo versions.

Running this took 1 hour and 8 minutes on my machine. For implementation, see my GitHub repository¹¹, where the curves are also available, saved in CSV format.

Finally, I wrote a small script that plots this¹² for each model. The results can be seen in 3.4 and 3.5.

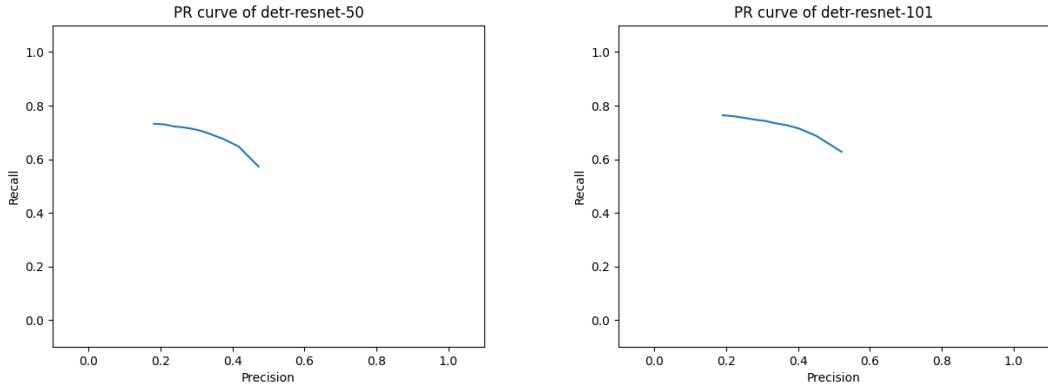


Figure 3.4: The PR curves of the two DETR models.

3.7.3 Running SORT

I ran SORT by cloning its repository, applying a small patch to account for division by 0 at empty detection files (happened with YOLOv5n models for some sequences, at high confidence thresholds), installing the dependencies and issuing the following command:

```
python sort.py --seq_path DETECTIONS_ROOT --phase MODEL --max_age 3
```

The main hyperparameters of SORT are `min_hits`, `max_age` and `iou_threshold`, all of them can be supplied to the script by command line parameters.

Minimum hits means the minimum number of consecutive times a new trajectory must be confirmed (by detection ‘hits’ that get assigned to it) to consider it a real one. Its default value is 3, and I left it at that.

Maximum age is the longest consecutive time that a trajectory without detection (if the detector somehow lost the object) can be kept alive. The position predictions are still updated, so the tracker looks for the object a little bit further apart at each frame, in the direction and considering the magnitude of the last known velocity. The default value is 1, but I set it to 3 because it seemed reasonable (based on my experiences in the Project Laboratory course, where similar maximum ages lead to better results).

IoU threshold is the minimum IoU value between the hypothesized current bounding box location of the object (predicted by the Kalman filter) and any detection bounding box that can be matched with it. The default value is 0.3, and I left it at that.

During the consecutive runnings of the script, I recorded the elapsed time and FPS reported by the script for each model. The results are confirming the belief that SORT is a very fast tracker, with the main factor for the tracking framerate in a real time scenario

¹¹See <https://github.com/peter-i-istvan/bsc-thesis>, under `tracking/detection_evaluation.py` for the code, `p_curve.csv`, and `r_curve.csv` for the curves in

¹²<https://github.com/peter-i-istvan/bsc-thesis> under `tracking/plot_pr.py`. The PNG plots are also there under the root directory.

(where detections are not precomputed) being the detection time (almost two orders of magnitude higher than the tracking time at each frame). For the results, see table 3.4.

	yolov5n	yolov5s	yolov5m	yolov5l	yolov5x	detr-resnet-50	detr-resnet-101
Time (s)	1313.99	1155.18	482.8	599.88	611.18	654.18	668.11
FPS	637	725	1710.2	1391.7	1369.1	1279.1	1253.5

Table 3.4: Statistics reported by the SORT script. The evaluation was done on 60 videos and on 10 confidence thresholds, making a total of 600 detection inputs. The time is the total evaluation time for all 600 inputs. The numbers are aggregated over confidence thresholds, but on lower values, tracking tends to take more time, as the consistent erroneous detections form a high number of false tracks.

3.7.4 Calculating tracking performance

After the tracking outputs were ready, I ran the tracking evaluation. The main goal was to calculate MOTA scores, average MOTA for every model-confidence threshold pair, across all sequences, to be precise. In the end though, I ended up calculating more metrics, to see a broader picture about the tracker performance.

For this, I used the `motmetrics` library, similarly to the detection evaluation. I processed each sequence with an accumulator objects, computing several summary metrics at the end. For every model, confidence threshold, sequence trio, I had a summary in the form of a Pandas dataframe, holding all the same metrics. I gathered all these, then aggregated across the sequences, receiving aggregate scores for each model and confidence threshold.

The scores I queried in for each sequence were the *MOTA*, *MOTP*, *number of identity switches*, the number of *mostly tracked trajectories* (80% or more of it was tracked), *mostly lost trajectories* (20% or less of it was tracked), and the *number of unique objects*. I aggregated the MOTA and MOTP by averaging (also saving minimum and maximum MOTA score), the rest by calculating the sum of their values.

insert tables when ready

insert MOTA - conf threshold plots when ready.

Running this took **TIME**. For implementation, see the GitHub repository¹³

3.7.5 Calculting benchmark scores

3.8 Conclusion

¹³<https://github.com/peter-i-istvan/bsc-thesis> under tracking/tracking_evaluation.py

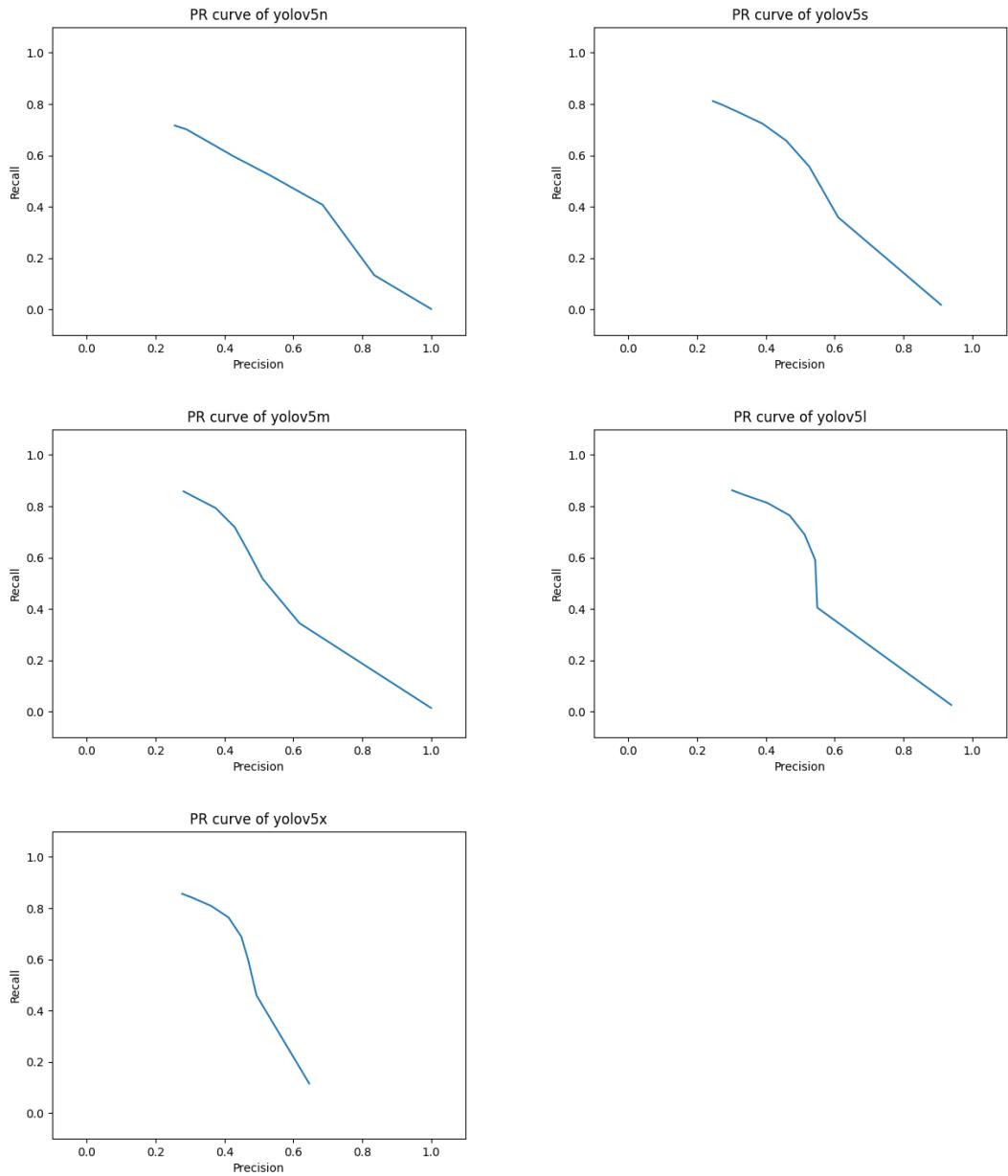


Figure 3.5: The PR curves of the five YOLO models.

Bibliography

- [1] Alex Bewley, Zongyuan Ge, Lionel Ott, Fabio Ramos, and Ben Upcroft. Simple online and realtime tracking. In *2016 IEEE International Conference on Image Processing (ICIP)*. IEEE, sep 2016. DOI: 10.1109/icip.2016.7533003. URL <https://doi.org/10.1109/2Ficip.2016.7533003>.
- [2] Laura Leal-Taixé, Anton Milan, Ian Reid, Stefan Roth, and Konrad Schindler. Motchallenge 2015: Towards a benchmark for multi-target tracking, 2015. URL <https://arxiv.org/abs/1504.01942>.
- [3] Tsung-Yi Lin, Michael Maire, Serge Belongie, Lubomir Bourdev, Ross Girshick, James Hays, Pietro Perona, Deva Ramanan, C. Lawrence Zitnick, and Piotr Dollár. Microsoft coco: Common objects in context, 2014. URL <https://arxiv.org/abs/1405.0312>.
- [4] Wenhan Luo, Junliang Xing, Anton Milan, Xiaoqin Zhang, Wei Liu, and Tae-Kyun Kim. Multiple object tracking: A literature review. *Artificial Intelligence*, 293:103448, apr 2021. DOI: 10.1016/j.artint.2020.103448. URL <https://doi.org/10.1016/2Fj.artint.2020.103448>.
- [5] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In I. Guyon, U. Von Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017. URL <https://proceedings.neurips.cc/paper/2017/file/3f5ee243547dee91fb053c1c4a845aa-Paper.pdf>.
- [6] Longyin Wen, Dawei Du, Zhaowei Cai, Zhen Lei, Ming-Ching Chang, Honggang Qi, Jongwoo Lim, Ming-Hsuan Yang, and Siwei Lyu. UA-DETRAC: A new benchmark and protocol for multi-object detection and tracking. *Computer Vision and Image Understanding*, 2020.
- [7] Nicolai Wojke, Alex Bewley, and Dietrich Paulus. Simple online and realtime tracking with a deep association metric. In *2017 IEEE International Conference on Image Processing (ICIP)*, pages 3645–3649, 2017. DOI: 10.1109/ICIP.2017.8296962.