FAKULTA INFORMAČNÍCH TECHNOLOGIÍ ČVUT V PRAZE

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

Název:	Detekce klíčových objektů na letištní stojánce z bezpečnostních kamer
Student:	Oliver Blaško
Vedoucí:	Ing. Marek Sušický
Studijní program:	Informatika
Studijní obor:	Znalostní inženýrství
Katedra:	Katedra aplikované matematiky
Platnost zadání:	Do konce letního semestru 2020/21

Pokyny pro vypracování

Pro rychlou obsluhu letadla na letištní stojánce je nutné správně synchronizovat servisní úkony. Aktuálně neexistují veřejná řešení, která by sledovala tyto úkony bez lidských pozorovatelů. Cílem této bakalářské práce bude navrhnout a implementovat systém, který by sám z kamerové nahrávky poznal klíčové objekty, které se podílí na jednotlivých akcích: dveře zavazadlového prostoru, pohyblivý most, cisterna na tankování, letadlo, vozíček se zavazadly, pushback vozítko, vozítko s pásem pro vykládku zavazadel, kontejner, obecné vozidlo. Jejich výskyt zobrazí přehlednou formou na časové ose.

Díky rychlému rozvoji v oblasti digitálního zpracování obrazu existuje řada algoritmů pro rozpoznávání objektů, nikoli však pro objekty typické pro letiště. V rámci této práce bude nutné vytvořit dataset z letištních nahrávek, provést rešerši, navrhnout úpravy algoritmů pro náš účel a demonstrovat jeho fungování na vzniklém datasetu.

Seznam odborné literatury

Dodá vedoucí práce.

Ing. Karel Klouda, Ph.D.
vedoucí katedry

doc. RNDr. Ing. Marcel Jiřina, Ph.D.
děkan

V Praze dne 12. února 2020

FACULTY
OF INFORMATION
TECHNOLOGY
CTU IN PRAGUE

Bachelor's thesis

Airport apron key object detection from surveillance cameras

Oliver Blaško

Department of Applied Mathematics
Supervisor: Ing. Marek Sušický

February 11, 2021

Acknowledgements

I would like to thank my supervisor Ing. Marek Sušický for his patience and guidance. Special thanks goes to Mgr. Tomáš Karella, for his great tips and feedback. The access to the computational infrastructure of the OP VVV funded project CZ.02.1.01/0.0/0.0/16_019/0000765 "Research Center for Informatics" is also gratefully acknowledged.

Declaration

I hereby declare that the presented thesis is my own work and that I have cited all sources of information in accordance with the Guideline for adhering to ethical principles when elaborating an academic final thesis.

I acknowledge that my thesis is subject to the rights and obligations stipulated by the Act No. 121/2000 Coll., the Copyright Act, as amended, in particular that the Czech Technical University in Prague has the right to conclude a license agreement on the utilization of this thesis as a school work under the provisions of Article 60 (1) of the Act.

In Prague on February 11, 2021 . . . . . .

Czech Technical University in Prague
Faculty of Information Technology

Citation of this thesis

Blaško, Oliver. *Airport apron key object detection from surveillance cameras.* Bachelor's thesis. Czech Technical University in Prague, Faculty of Information Technology, 2021.

Abstrakt

Nasledujúca práca sa venuje problému detekcie objektov so zameraním na objekty, ktoré sú typické pre prostredie letiska. Práca popisuje aktuálne riešenia tohto problému, trénovanie, vyhodnotenie a nasedenie modelu do aplikácie, ktorá dokáže úspešne detekovať objekty typické pre letisko.

Klíčová slova hluboké učení, strojové vidění, detekce objektů, bezpečnostní kamery, letišní stojánka, yolo, sledování objektů

Abstract

This work explores the field of object detection with attention to objects that are present in the airport apron environment. In addition, it describes the research, training, evaluation and deployment of a model into the application that is able to successfully detect key airport objects.

Keywords deep learning, computer vision, object detection, surveillance cameras, airport apron, yolo, object tracking

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Motivation

There has been significant progress in the computer vision and particularly in the field of object detection [18] in the last couple of years. Thanks to this progress, object detection models for basic day-to-day objects like dogs, cats, different kinds of fruit, cars, trains, etc., are surprisingly accurate [18]. However, there are still many different kinds of environments in life, where there is a need to detect objects that are somehow special and unique. One of these types of environments is an airport. More specifically, the airport apron, which is the part of an airport where aircraft are parked, unloaded or loaded, refilled, or boarded. Many unique objects are present in the airport apron - special types of airport vehicles, boarding bridges, parts of the airplane; these are all crucial objects that ensure the smooth and fast aircraft service. Apart from few startups, no publicly available solution could detect this type of objects without human observers. This thesis aims to design and implement a system that could accurately detect key airport apron objects from surveillance cameras that are usually present in every airport apron for security reasons.

1.2 Goals of the Thesis

This process can be boiled down to a few steps:

1. Creation and preparation of the data set from different airport surveillance camera recordings
2. Research of the current state-of-the-art models for object detection
3. Training of the proposed models adjusted for key airport apron object detection
4. Evaluation and analysis of trained models

5. Design and implementation of the system

Chapter 2

State-of-the-art

This chapter introduces the reader to the deep learning scientific field. It firstly describes the various problems and tasks of the field followed by the methods and models used for solving them.

2.1 Deep learning

One of the most prominent computer scientist and deep learning researcher Andrew Ng [39] states that *artificial intelligence* (AI) is the new electricity of our generation, and it will transform countless industries like transportation, healthcare, communication, and so many more in a way that electricity did change the life of the previous generations. Thanks to deep learning (not exclusively), we can slowly start observing this kind of progress with self-driving cars, AI systems that can diagnose cancer in the early stages, or chatbots that nowadays replace some customer service area. So what is this deep learning? The term *deep learning* (DL) as François Chollet defines in his DL book [1] is a new take on learning from data that emphasizes learning successive layers of increasingly meaningful representations. The deep in DL doesn't stand for any kind of a deeper understanding achieved by the approach, but rather for the idea of tens or hundreds of successive layers of representations, the so-called depth of the model. DL is simply a mathematical framework for extracting information from data.

2.2 Computer vision

The vision is arguably the most important sense that we humans have. It helps us recognize the objects, manipulate them, observe human emotions or navigate in the world. The ongoing research on image processing focuses on solving the same tasks using a computer. Thanks to the rapid advances in DL research, the goal of teaching computers to infer something about the real

world by observing image data is closer than ever before. The multidisciplinary field that tries to reach this goal is called *computer vision* (CV). CV tasks include methods for acquiring, processing, analyzing, and understanding digital images. The following section describes some of the most researched CV tasks.

2.3 Computer vision tasks

2.3.1 Image classification

Image classification (IC) is one of the most fundamental CV tasks. It attempts to comprehend the entire image as a whole, and the goal is to classify the image by assigning it a specific label from the pool of previously defined category labels. IC is usually performed on images containing only one object.

2.3.2 Image classification + localization

The task of IC combined with localization also operates with images containing a single object. In addition to IC, we expect the location of the object and category label. This is usually done by drawing a bounding box around the region where the object is present.

2.3.3 Semantic segmentation

The task of *semantic segmentation* (SEM) works with images containing multiple objects and it outputs a prediction of the category for every pixel. Similarly as with IC, SEM also works with categories that the images could contain. The SEM does not differentiate instances; it only cares about the pixels.

2.3.4 Object detection

Object detection (OD) is the task of detecting instances of objects of a certain class within an image. The input image can contain multiple objects, and the expected output is a prediction of bounding boxes around each detected object accompanied with the corresponding category of that object.

2.3.5 Instance segmentation

Instance segmentation (IS) can be considered as a combination of OD and SEM. The input is once again an image containing 0, 1, or multiple distinct objects. The IS combines previous outputs (object category and location) with the pixel segmentation. Each pixel is assigned by the detected object or background flag (0).

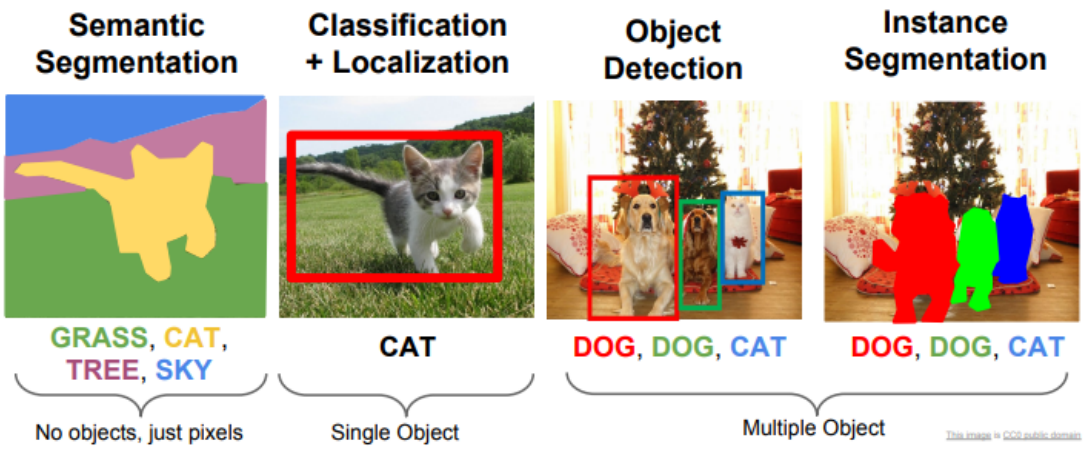


Figure 2.1: Illustration and comparison of the computer vision tasks. Justin Johnson[2]

2.4 Object detection methods

The popular website Paperswithcode [10] keeps track of the current state-of-the-art methods and models categorizes OD methods into two main types[10]: one-stage methods and two stage-methods. One-stage methods prioritize inference (detection) speed, and example models include YOLO [7], SSD [42]. Two-stage methods prioritize detection accuracy, and example models include Faster R-CNN [6], Mask R-CNN [40] and Cascade R-CNN [41].

2.4.1 Naive approach

We can look at the OD problem as a classification using a sliding window [3]. Sliding window is a common approach in CV - we take different regions of the image, and use a classifier (CNN for example) to classify which object is present in that particular region, or to classify the region as background. However with this approach comes one problem. These objects could appear in any location of the image, any size, and any aspect ratios. That means that we would have to select a huge number of regions, and it would be computationally very expensive. That's why this approach is never used in practice, however it is a good starting point, and it was a motivation for discovering architectures such as R-CNN.

2.4.2 R-CNN

To bypass the problem of selecting a huge number of regions described above Ross Girshick proposed a method [4] to use selective search to extract just ≈2000 regions that are likely to contain objects (also referred to as a regions

proposal). So rather than applying a classification network to every possible location and scale in the image, he applied the regions proposal method first and then fed the proposed regions into a CNN for classification. This approach ended up being much more computationally tractable. However, the region's proposal brought one problem. Each proposed region could be of a different size, and since CNNs require fixed input image size, the author needed to warp them into a fixed square before feeding the proposed regions. The CNN acts as a feature extractor and the output dense layer consists of the features extracted from the image. The extracted features are then fed into a SVM classifier [11] in order to predict the presence of the object. R-CNN problems: training of the network is slow (≈84hours) and it takes a lot of disk space [3], inference (detection) is slow as well (≈47sec) [3]. These two constraints sparked an idea to create Fast R-CNN.

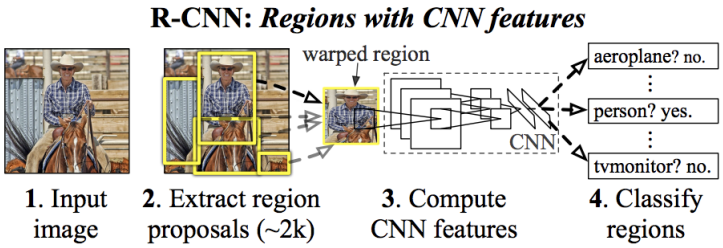


Figure 2.2: Illustration of the regions proposal method in R-CNN: [4]

2.4.3 Fast R-CNN

Instead of feeding each region proposal separately as in R-CNN, the authors of faster R-CNN [5] decided to provide the whole image into a CNN all at once, generating a convolutional feature map corresponding to that image. The network identifies the regions of a proposal using the feature map; the created regions are warped into a fixed size and fed to fully connected layers. This approach reduced the computational time significantly [5] since we don't do ≈2000 convolutional operations for each proposed region individually, but rather only once for each image. The improvement in training and also inference speed compared to R-CNN can be seen in the figure below 2.3.

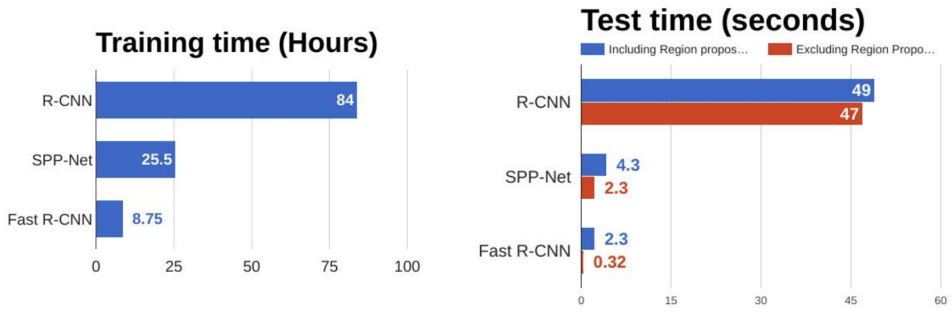


Figure 2.3: R-CNN vs. Fast R-CNN training and inference speed comparison, Source: [3]

2.4.4 Faster R-CNN

As can also be seen in the figure above 2.3, the Fast R-CNN's most time - consuming operation was the regions proposal (red line), which affected the overall performance of inference significantly. Faster R-CNN eliminates the region's proposal. The network learns the regions by itself [6]. Up until the creation of a convolutional feature map, Faster R-CNN works similarly as Fast R-CNN. But rather than using selective search, the regions are predicted by a separate network.

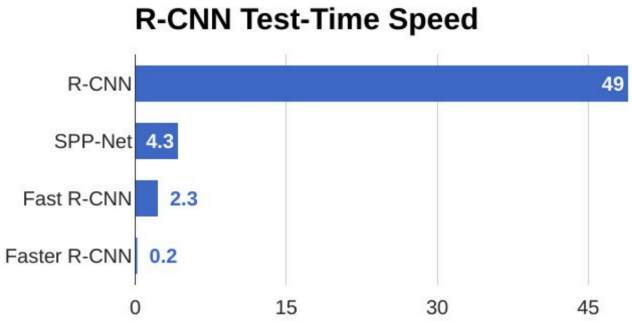


Figure 2.4: Inference speed comparison between R-CNN, Fast R-CNN and Faster R-CNN in seconds, Source:[3]

2.4.5 YOLO

All of the previously mentioned object detection methods did not look at the images as a whole but only on proposed regions with a high probability of object appearance. YOLO - You Only Look Once [7] method works differently. The authors thought of the image as a SxS grid. Within each grid, they created n bounding boxes centered at each grid cell, and the network predicts scores

for each class and offset values for the bounding box. Bounding boxes with a class probability above a certain threshold are selected and used to locate the object within an image. At the time of the release, YOLO worked faster than most object detection methods available at the time [7]. However, the downside of YOLO is that it struggles with small objects within the image [9]. Key takeaway: Faster R-CNN is slower but more accurate, YOLO is much faster, but not as accurate.

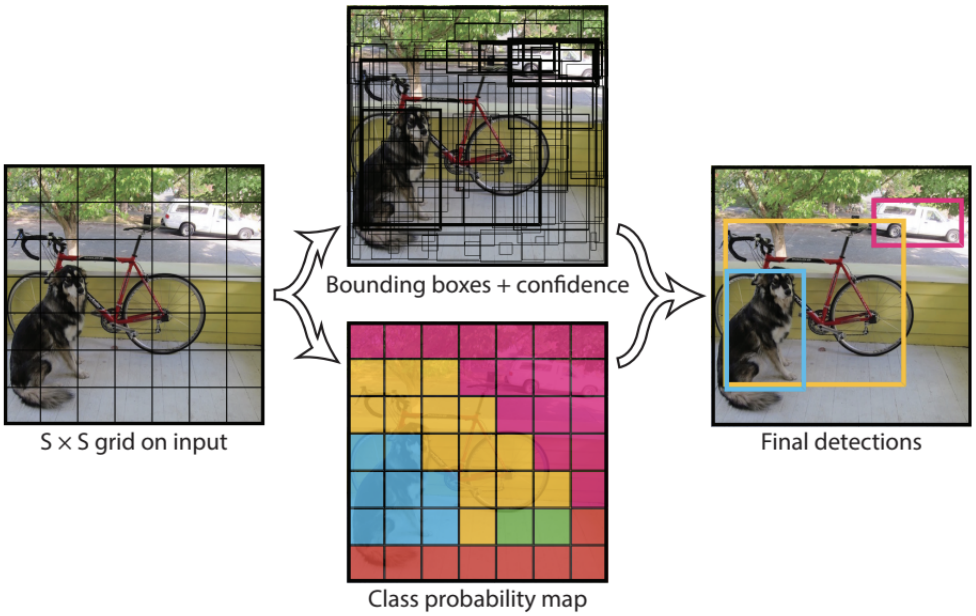


Figure 2.5: YOLO object detection. Source: [7]

2.4.6 YOLOV4

Many iterations of the Yolo model have been developed [7] [8] [18], each one improving its predecessor. As of this time, the most recent one is YoloV4 [18]. The authors of the YoloV4 improved Yolo model's performance by experimenting with different techniques and features which are said to increase CNN accuracy [18]. They categorized the experimental features and techniques into two categories:

1. Bag-of-Freebies - methods that only change the training strategy, therefore increase the training cost, not the inference cost. Examples include data augmentation, soft labeling [34].

2. Bag-Of-Specials - improvements in the network that impacts the inference time slightly, however with a return in increased performance. Ex-

amples: Experimenting with different activation functions [35] and using post-processing techniques like non-maximum suppression described in 2.7.2

By this experimental setup, the authors developed a model that runs twice faster than some of the other state-of-the-art models with a comparable performance [18]. The comparison is described in more detail later one in section 3.2.

2.5 Object tracking

Object tracking (OT) [25] is the process of:

1. Taking an initial set of object detections (in the form of bounding box coordinates)

2. Creating and assigning unique ID for each initial detection

3. Tracking each of the objects as they move around frames in a video, maintaining the assignment of unique IDs

OT assigns a unique ID to each detected object, thus counting the number of objects present in the video.

2.5.1 Centroid tracking algorithm

One way of achieving the object tracking can be the centroid tracking algorithm [25]. The algorithm can be split into a few incremental steps:

1. retrieves the initial detections from the starting frame using some object detector, and register detected objects.

2. incrementally retrieves the detections for every next frame:

   a) compares the new detections with detections in the previous frame by computing the Euclidean distance [36] between all previous and all current detections using the centroids of the two bounding boxes.

   b) associate centroids with minimum distance between subsequent frames and update the new (x,y) coordinates

   c) if there are more detected objects compared to previous frame, register the new ones

   d) if a tracked object was not associated with any other new detection for N subsequent frames, deregister it

2.5.1.1 The drawbacks of algorithm

When two objects overlap each other in the scene, the algorithm may swap the ids between the two, due to the metric used for associating the objects.

2.6 Metrics

In order to measure the trained model's performance, there need to be some quantification reflecting how well it works. There is a vast spectrum of different metrics suitable for different tasks. One of the popular metrics for measuring the performance of OD methods is mean average precision (mAP) [29]. mAP consists of multiple other metrics, so let's start by defining them:

2.6.1 Intersection over union

Intersection over union (IoU) is a metric that evaluates how similar is the predicted bounding box compared to the grounding truth bounding box. It's the ratio of the area overlapping boxes to the total combined area of the two boxes.

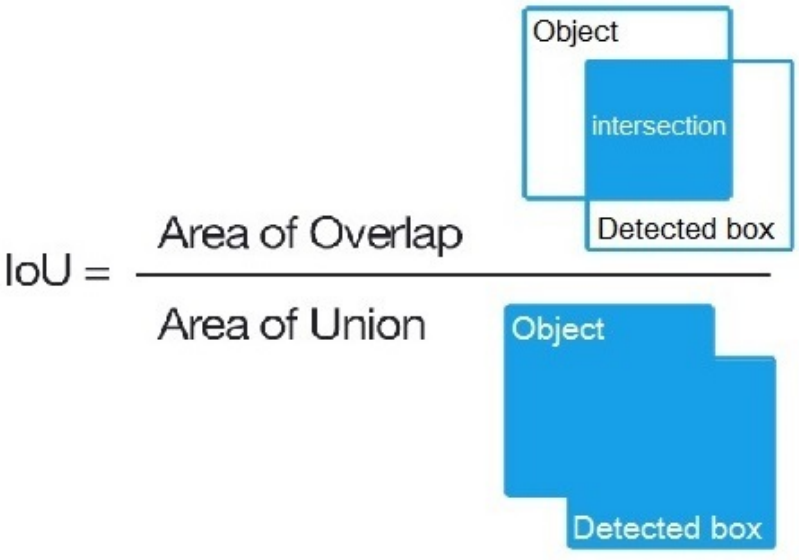


Figure 2.6: IoU metric ilustrated. Source: [16]

2.6.2 Precision and recall

Precision and recall are calculated using the true positives (TP), false positives (FP), and false negatives (FN) values. In the case of object detection, they can be defined as:

1. classify the OD as TP if $IoU => 0.5$

2. classify the OD as FP if $IoU < 0.5$

3. classify the OD as FN if the model failed to detect grounding truth completely

Keep in mind that different thresholds for the IoU could be used. Having said that the precision and recall are defined as:

$$PRECISION = \frac{TP}{TP + FP} \tag{2.1}$$

$$RECALL = \frac{TP}{TP + FN} \tag{2.2}$$

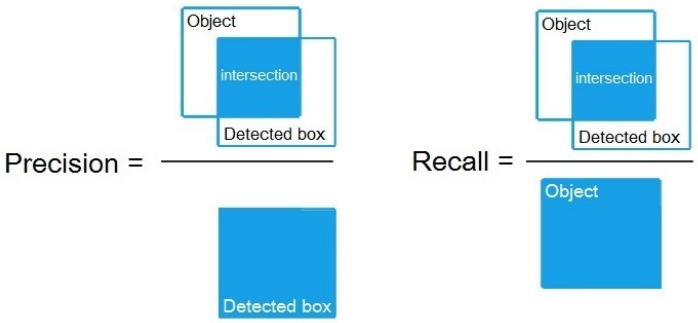


Figure 2.7: Illustration of precision and recall in the context of object detection. Source: [16]

2.6.3 Precision/Recall Curve (PR Curve)

A precision-recall curve is a plot of the precision (y-axis) and the recall (x-axis) for different thresholds [33].

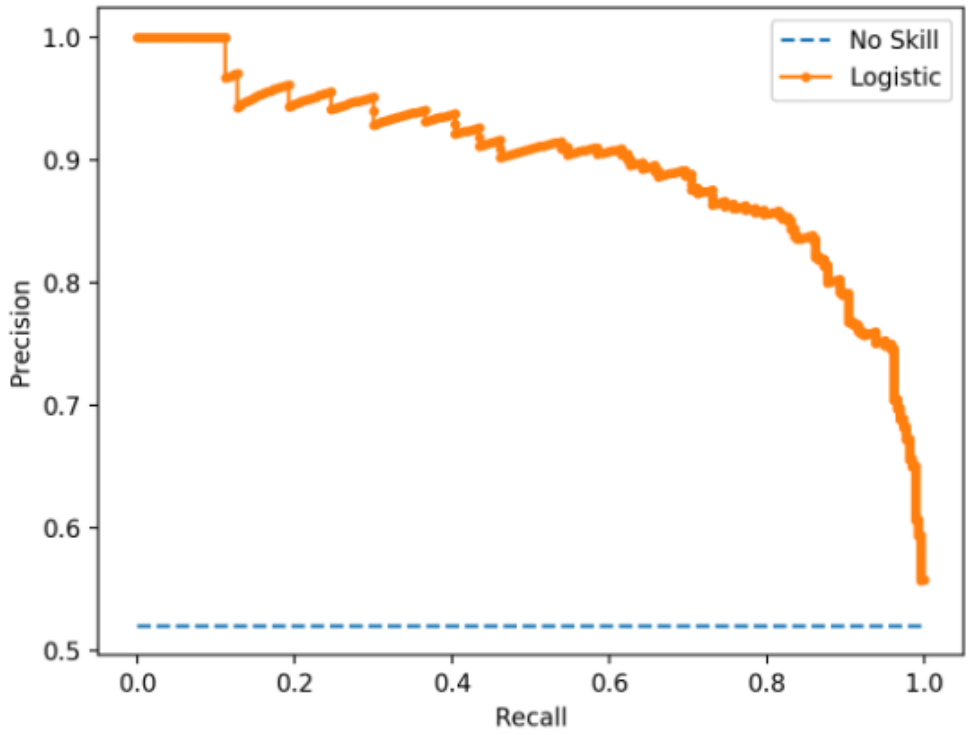


Figure 2.8: Example of precision/recall curve graph. Source: [33]

2.6.4 Average precision

Average precision (AP) captures the whole shape of the precision recall curve into a single number. AP is calculated as the weighted mean of precisions achieved at each threshold, with the increase in recall from the previous threshold used as the weight. [29]

2.6.5 Mean average precision

Finally, mAP (mean average precision) is simply defined as a mean value of average precisions for each class.

2.7 Other important computer vision techniques

2.7.1 Data augmentation

Data augmentation is one a regularization techniques that can be used for improving the generalization of OD models. Especially when there is a lack of data, it is a powerful technique that artificially creates variations in existing

images to expand the current data set. Transformation techniques like image mirroring, zooming, rotating, and color-shifting help increase the model generalization and avoid overfitting. Data augmentation can be categorized into groups:

1. Pre-processing augmentation that increases the size of the training data set before the actual training, this is usually done when there is a small training data set that needs an expansion
2. Real-time augmentation is usually used when we have large enough data set, but we want to improve the model generalization. During the training process, real-time augmentation uses the mentioned transformation techniques in mini-batches and feeds them to the model in real-time, so there is no need to store the augmented images on disk.

2.7.1.1 Data augmentation methods

1. Flipping: flipping the image vertically or horizontally
2. Rotation: rotates the image by a specified degree
3. Random cropping: randomly crops the image into a smaller chunk
4. Color shifting: adding/subtracting to RGB channels different distortions
5. Image blurring: deliberately lowering the quality of an image by adding blur (noise)

2.7.2 Non-maximum suppression

The problem that comes with OD is that the detector may find multiple detections of the same object. Non-maximum suppression (NMS) [26] is a CV technique that selects the most suitable detection out of many over-lapping defections. The process can be described as:

1. Discarding the bounding boxes with a probability below a certain threshold
2. While there is still some remaining bounding boxes:
    a) Pick the bounding box with the highest probability and mark it as the detection
    b) Discard all bounding boxes that overlap (IoU >= 0.5) with the previously marked bounding box.
    c) Discard the bounding box which was marked as detection

Chapter 3

Analysis and design

3.1 Describing the problem

The problem that this thesis is trying to solve can be categorized as a custom object detection problem. The state-of-the-art models described in section 2.4 usually detect only a fixed number of different everyday objects. The famous example, Common Objects in Context (COCO) [19] data set on which some state-of-the-art models are trained or evaluated consists, as the title suggests, of objects that humans interact with often. Examples of these objects include car, train, cow, sofa, bottle, etc. Using just the previously described models will not work in out case since the models probably never encountered objects like APR tank truck or push back truck and will not detect them. However, using transfer learning, we can capitalize on this previously learned knowledge and re-train these architectures on our custom data set, not starting from scratch but starting where they ended using the trained weights.

3.2 Choosing the model

As the title of the paper "YOLOv4: Optimal Speed and Accuracy of Object Detection" [18] suggests, YOLOv4 is considered the current state-of-the-art for the OD problem. The comparison that authors provided in the paper suggests [18] that YOLOv4 runs twice faster than some of the other state-of-the-art models with comparable performance. It also improves its predecessor YoloV3's AP and FPS by 10% and 12%, respectively [18]. The author Alexey Bochkovskiy also released a forked Darknet repository, which provides developers with a framework for training, testing and evaluating DL models. The newly released repository includes options for training the YoloV4 model and also has the possibility to start training on the previously learned weights, thus enables transfer learning. Taking these two factors into account We decided to start with the YoloV4 model. YoloV4 also has a "little brother", YoloV4-tiny [20]. The crucial distinction between the two is the network size. The size

reduction improves the inference speed 8 times while maintaining roughly 2/3 of performance on the COCO data set [20]. Besides, this blog post [21] shows that the decrease in performance can be even smaller on custom detection tasks. The inference speed will also be an essential factor in the overall application user experience, so we also decided to train YoloV4-tiny and compare the results between the two in the end.

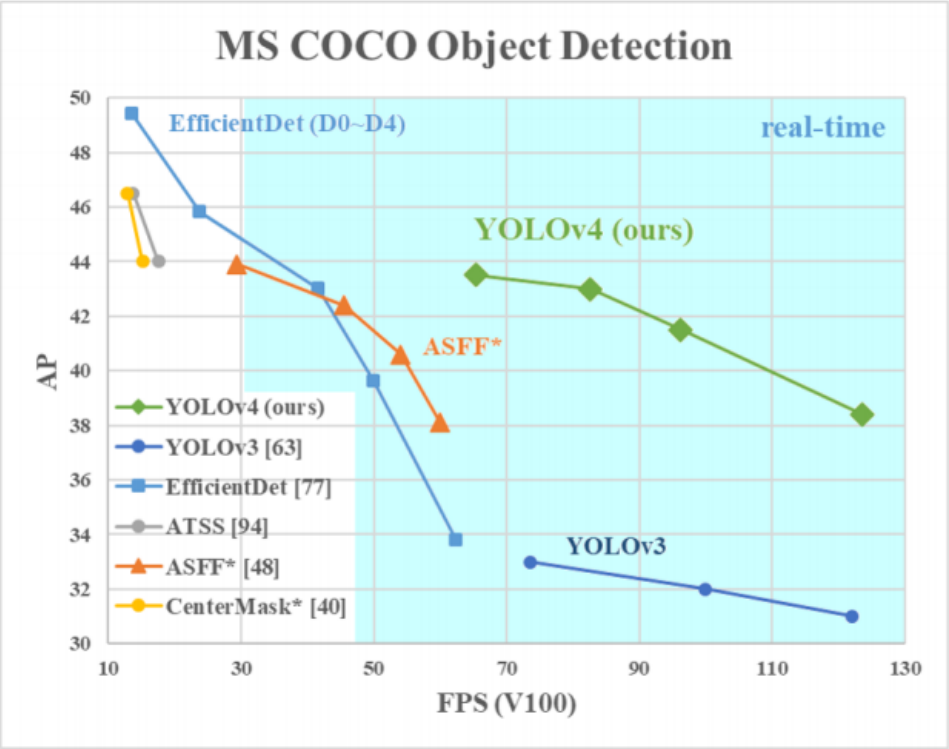


Figure 3.1: Comparison of the YoloV4 model and other state-of-the-art object detectors. Source: [18]

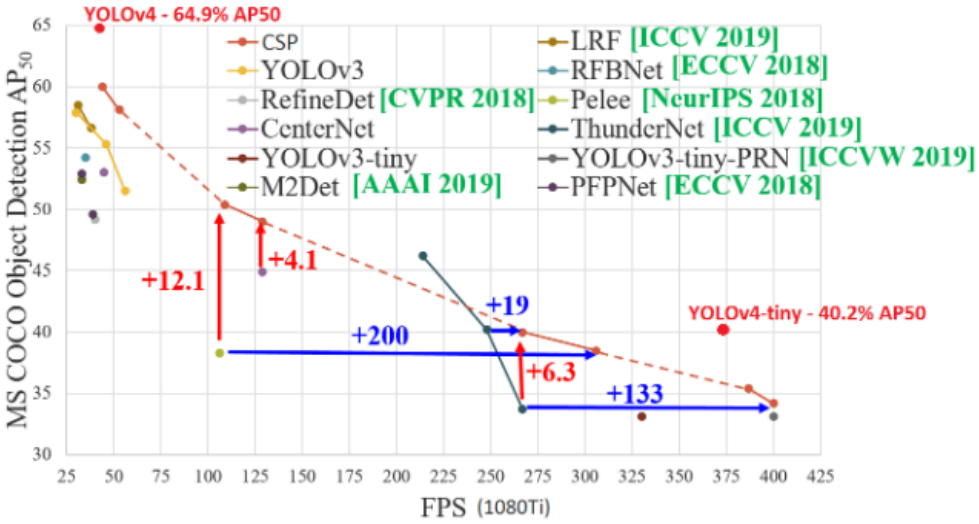


Figure 3.2: Comparison of the YoloV4 and YoloV4-tiny using AP and FPS. Source: [20]

3.3 Application design

3.3.1 Functional and non-functional requirements

3.3.1.1 Functional requirements

1. The user will be able to input a video file.
2. The user will be able to set the inference parameters (model to use, confidence).
3. The application will process the video and perform inference on it.
4. The application will let the user know about the estimated time to process the video.
5. The application will output the annotated video file.
6. The application will output a timeline graph of detected objects.
7. The application will render the results in the web-browser.
8. The user will be presented with the annotated video and timeline graph of detected objects.
9. The user will be able to pause/play/speed up the outputted annotated video.

3.3.1.2 Non-functional requirements

1. The application will consists of two parts: inference running application and visualization application.

2. The inference application will be written in Python.

3. The visualization application will be written in ReactJS.

3.3.2 Inference running application

This part of the application will handle the loading of the input video, preprocessing it, and running inference. One of the requirements of the application is to visualize the detected objects using the timeline graph. In more detail: It shows when exactly the object entered the scene when the object left the scene, and also if the previously lost object came back to the scene. This will be achieved by implementing some object tracking algorithm. Finally, it will output the annotated video with detected objects and the timeline graph. This part of the application will be written in Python because of the OpenCV library [22] that contains built-in methods for image processing and inference running. We'll also use the Altair statistical visualization library [23] for the timeline graph generation.

3.3.3 Visualization front-end application

This part of the application will serve for visualizing the results of the Python application. It will serve as a "view only" front-end application. The user will be presented with two pages, one rendering the annotated video (Detection page) and the second rendering the timeline graph (Dashboard page). The user will be able to switch between the two pages using the tab component. It will be written in ReactJS [24] since it's easy to setup JavaScript library for building user interfaces that also contains built-in video players with features such as video pausing or video speed up.

3.3.3.1 Wireframes

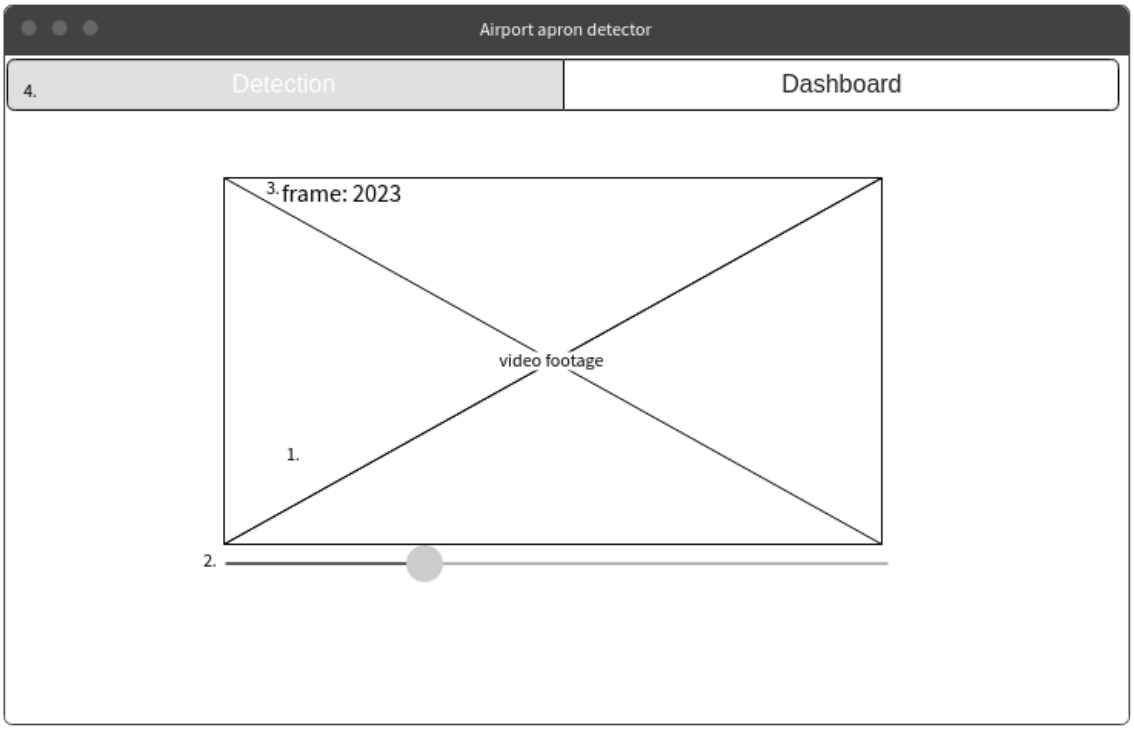


Figure 3.3: Detection page wireframe

1. region for video player

2. visualization of the video progress

3. number of the currently displayed frame

4. tabs component for changing the pages

Figure 3.4: Dashboard page wireframe

1. tabs component for changing the pages
2. timeline graph representing the detected object appearance in the video

Chapter 4

Data set

Firstly let's define basic terms to provide reader with a deeper understanding of the APR dataset.

4.1 Key terms

1. *Airport apron* (APR) is a defined area on an airport intended to accommodate aircraft for purposes of loading or unloading passengers or cargo, refueling, parking, or maintenance [12].

2. *Aircraft scene* the moment when aircraft is entering or leaving APR in a video.

3. *Vehicle scene* the moment when APR vehicle is entering, leaving APR, or performing its duty in a video, that includes refueling, loading, unloading, and other maintenance.

4.2 Raw footage

At this time, there is no publicly available solution for detecting key APR objects. The same goes for data set specialized for this task. Arguably this thesis's most important task was to acquire, clean and label raw video footage from airport security cameras. Thus creating an adequate data set for detecting key APR objects. The data set we made consists of two different sources described in more detail below.

4.2.1 Japan airport footage

Japan airport footage - still footage from a security APR camera. The footage was obtained from multiple days and consists of different weather conditions: day footage, night footage, sunny conditions, rainy conditions, etc. The

footage was provided by Profinit [14], the private company that supervised this thesis, and it's not publicly available. The footage is more static, and important scenes happen sporadically. We ended up with 2 hours and 30 minutes of footage containing 7 airport scenes and over 50 vehicle scenes, which translated to labeled 270 000 images.

Figure 4.1: Example of Japan source night footage. Source: [14]

Figure 4.2: Example of Japan source day footage. Source: [14]

4.2.2 Hongkong airport footage

To train a model that will be more universal and not specific to only one APR, we decided to add one more video source. It's similar footage to the Japan source, but different camera angles and slightly different looking APR vehicles. The footage comes from one day that consists of 3 aircraft scenes and multiple vehicle scenes. Compared to the Japan source, it's more dynamic, and vehicle scenes happen more often. Total of 43 minutes of footage, which translates to 50 421 labeled images. The footage was obtained from publicly available YouTube video, which can be accessed here: [13].

Figure 4.3: Example of Hongkong source footage. Source: [13]

4.3 Key objects

The system needs to be able to detect key APR objects. These objects represent crucial airport vehicles that ensure the safe and successful maintenance of the aircraft. Here is a more detailed description of all key APR objects that our system will detect:

1. **aircraft** - self explanatory

(a) Japan aircraft example 1. Source: [14]

(b) Japan aircraft example 2. Source: [14]

(c) Hong-kong aircraft example. Source: [13]

2. **cargo-door** - aircraft doors that are used for (un)loading of the luggage

(a) Cargo door example 1. Source: [14]

(b) Cargo door example 2. Source: [14]

3. **jet-bridge** - enclosed, movable connector which most commonly extends from an airport terminal gate to an aircraft

(a) Jet bridge example 1. Source: [14]

(b) Jet bridge example 2. Source: [13]

4. **tank-truck** - vehicle that is used for fueling the aircraft or supplying it with electricity

(a) Tank truck supplying electricity. Source: [14]

(b) Tank truck supplying fuel. Source: [14]

5. **cargo-truck** - special vehicle that is used for moving the cargo-boxes

(a) Cargo truck example 2. Source: [14]

(b) Cargo truck example 2. Source: [14]

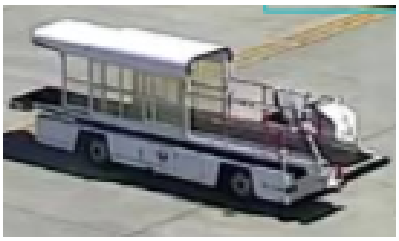
6. **push-back-truck** - vehicle that carries out the pushback procedure - aircraft is pushed backwards away from an airport gate by external power

(a) Push back truck example 2. Source: [14]

(b) Push back truck example 2. Source: [14]

7. **luggage-loading-truck** - vehicle that is used for (un)loading the luggage through the cargo-door

(a) Luggage truck example 1. Source: [14]



(b) Luggage truck example 2. Source: [14]

8. **cargo-box** - container used for storing luggage or other aircraft material

(a) Cargo box example 1. Source: [14]

(b) Cargo box example 2. Source: [13]

9. **basic-truck** - normal car that operates on APR

(a) Basic truck example 1. Source: [13]

(b) Basic truck example 2. Source: [13]

4.4 Image labeling

After the extraction of frames (images) from the raw video footage, they needed to be labeled.

4.4.1 Labeling format

Since we'll be training models using the Darknet framework [15] and as the documentation suggests, the labelling format needs to follow specific rules. Every training image in the data set needs to have accompanied label .txt file with the same name and every line in that file represents class number and bounding box coordinates per each present object in a following format:

1. `<object-class> <x-center> <y-center> <width> <height>`, where:
   a) `<object-class>` - integer class number from 0 to (# of classes-1)
   b) `<x-center> <y-center> <width> <height>` - float values relative to width and height of image in range: `(0.0 to 1.0]`
   c) `<x-center> <y-center>` - center coordinates of a bounding box
   d) `<width> <height>` - width and height of a bounding box

4.4.2 Labeling tool

The labeling process was done using the CVAT video annotating tool, available at *cvat*. As the documentation [17] states: "CVAT is free, online, interactive video and image annotation tool for computer vision. It is being used by our team to annotate million of objects with different properties."

4.5 Train/test/valid split

Since the images came from consequent video footage, splitting them into the training, testing and validation set randomly would not be a good idea. It could split very similar images into both training and testing sets. This means

that the model will be evaluated on the images it has already seen before, and the results will be skewed. Therefore we decided to cut out two different 15 min parts of the video containing multiple scenes and proclaimed them as testing and validation set. The rest of the labeled footage was used for the training set.

4.6 Image pre-processing

4.6.1 Frame filtering

Since the footage from both sources is mostly still, and crucial scenes happen erratically, we decided to filter out the frames in a specific way so that the data set doesn't contain multiple similar images. The filtering algorithm can be described in a few steps:

1. Manually find where the scene doesn't change for a long period of time (more than 3 min.) and cut this footage completely

2. Manually find where crucial aircraft scenes happen and keep only every 10th frame of this footage

3. Manually find where crucial vehicle scenes happen and keep only every 30th frame of this footage

4. Keep only every 50th frame from the rest of the training video

5. Keep only every 10th frame of the testing/validation video

This process radically shrank the data set's size, just for example: from previously annotated 320 421 images, we ended up with 8103 images.

4.6.2 Image augmentation

Considering the previous point, numerous hours of labeled video footage was needed and since it's a very time-consuming task, We decided to apply data augmentation techniques. We performed 5 different image augmentation techniques only on the training data set which in the end quintupled the size of it. Used techniques:

1. Vertical flip

Figure 4.13: Original image. Source: [13]

Figure 4.14: Vertical flip performed

2. Horizontal flip

Figure 4.15: Original image. Source: [13]

Figure 4.16: Horizontal flip performed

3. Horizontal and vertical flip at the same time

Figure 4.17: Original image. Source: [13]

Figure 4.18: horizontal and vertical flip performed at the same time

4. Blurring of the image

Figure 4.19: Original image. Source: [13]

Figure 4.20: Blurring performed

5. Raising hue value

Figure 4.21: Original image. Source: [13]

Figure 4.22: Raising hue value performed

Chapter 5

Realisation

5.1 Model training

5.1.1 Training environment

For the training of the models, we used the Darknet framework. Darknet [27] is an open-source neural network framework written in C and CUDA. It is fast, easy to install, and supports CPU and GPU computation. More specifically we used Alexey Bochkovskiy's fork [15] of Darknet that includes many improvements and options to train more recent models. Both proposed models were trained on GPU Tesla V100 using the Research Center for Informatics [37] cluster which provides CTU students with access to a very expensive hardware.

5.1.2 Model adjustment

In order t train the models in the Darknet environment, model needs to be defined in a numerous separate files:

1. `model.names` - defining category labels, one label per line
2. `train.txt` - containing paths to all training data, one image path per line
3. `test.txt` - same, but for testing data
4. `valid.txt` - same, but for validation data
5. `model.data` - defining the number of classes and paths to all other model files
6. `model.cfg` - defining the architecture of the model

We based the architecture model.cfg file on the original YoloV4.cfg file [28]. However, as the documentation [15] suggests: for training a custom object detector with a custom number of classes some changes need to be made:

1. set param $batch = 64$
2. set param $subdivisions = 16$
3. set param $max\_batches = num\_of\_classes * 2000$
4. change network size set param $width = 416$ set param $height = 416$
5. set param $steps = 80\%, 90\%$ of param max_batches
6. set param $classes = num\_of\_classes$ in each Yolo layer
7. set param $filters = (num\_of\_classes + 5) * 3$ in each convolutional layer before the Yolo layer

The same process was repeated for YoloV4-tiny with minor changes. All configuration files are accompanied in the repository. Darknet has the ability to start the training process on already pre-trained weights, thus enabling transfer learning. For both YoloV4 and YoloV4-tiny we downloaded the pre-trained weights for the convolutional layers available at [15].

5.1.3 Model training

The models in Darknet are trained using the command: `./darknet detector train <path_to_data_file> <path_to_cfg_file> <path_to_weights> -map`. YoloV4 and YoloV4-tiny were trained on 10 000 and 4000 iterations respectively.

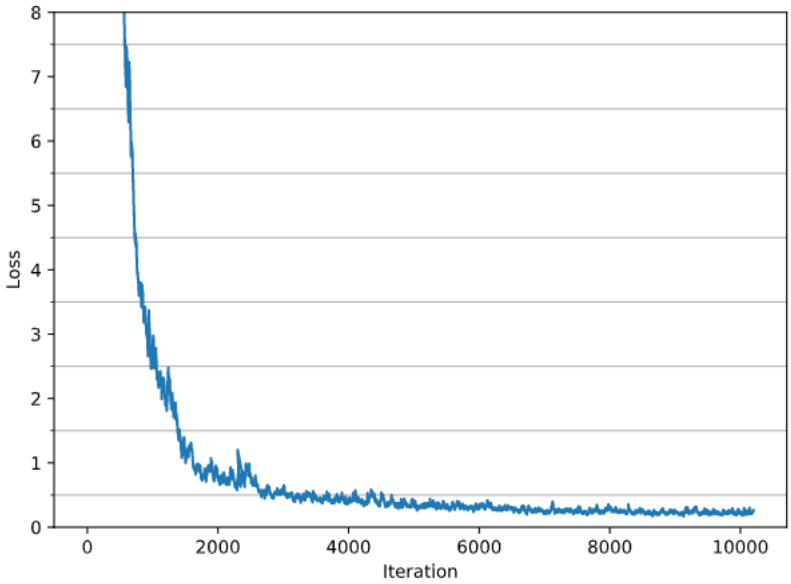


Figure 5.1: Custom YoloV4 train loss across epochs chart

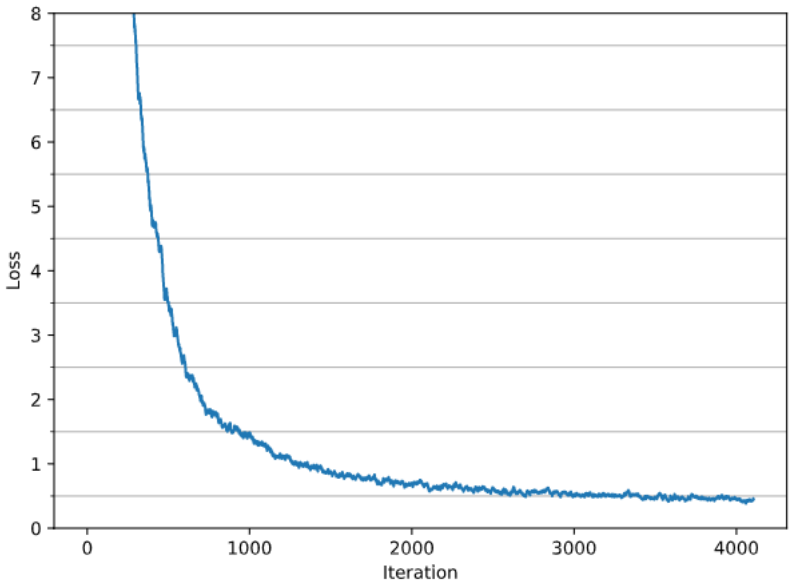


Figure 5.2: Custom YoloV4-tiny train loss across epochs chart

Using the flag -map in the training command, Darknet will compute the mAP on validation data set for each 4 epochs, where `epoch = number of training images/batch size` and automatically chooses the weights where the mAP was the highest and saves them.

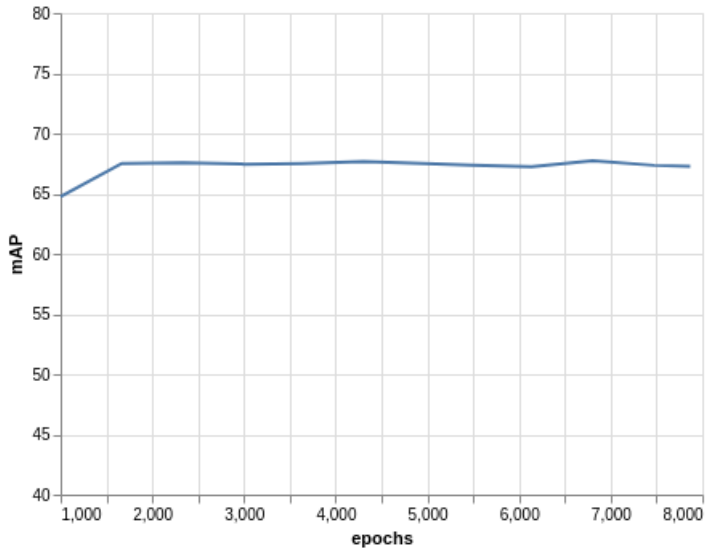


Figure 5.3: Custom YoloV4 mAP (validation data) across epochs chart

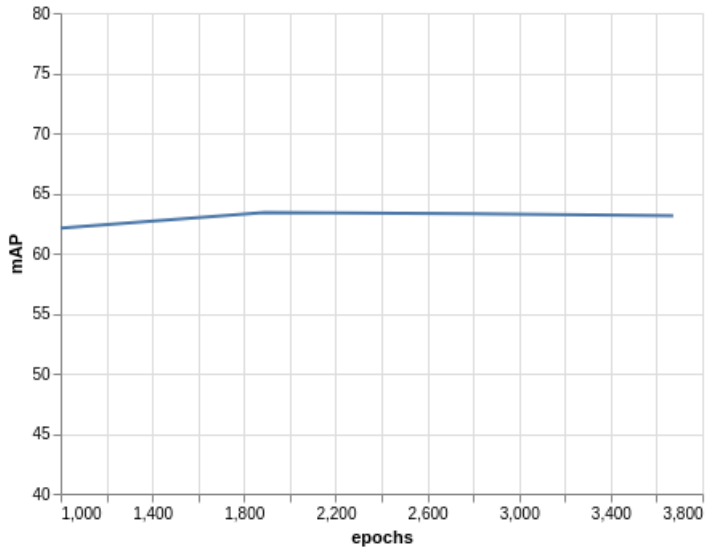


Figure 5.4: Custom YoloV4-tiny mAP (validation data) across epochs chart

It seems like the models learned the characteristics of the data in the first $\approx$ 2000 iterations and after that reached the performance ceiling.

5.1.4 Model evaluation

Models where evaluted using the mAP described in section 2.6.5 on testing data.

Metric	YoloV4	YoloV4-tiny
mAP_test	67.72%	64.25%
FPS(Tesla V100)	90	369

YoloV4-tiny has a slightly lower mAP, but the increase in the FPS performance is significant (4x). Therefore YoloV4-tiny seems like a better candidate to be deployed into the application. However, both models are included in the implementation and user can choose which one to use.

5.2 Implementation

This section describes the implementation of proposed framework in Section 3.3. The back-end side of the application is written in Pyhton with the dependence on other libraries, such as OpenCV [22], Altair.io [23] and Clodsa (library for data augmentation) [30]. The main functionality is divided into 5 modules:

5.2.1 Module constants

The module contains definitions of constants used in the application, including paths to model configuration files and various parameters for used algorithms.

5.2.2 Module create_dataset

The module handles the creation of the data set described in section Chapter 4. In more detail, it:

1. extracts the frames from the video files
2. extracts annotations from the annotation tool
3. binds the frames with annotation files
4. tests if the annotations match the correct frames and the correction of labelling format
5. handles filtering described in (4.6.1)
6. handles data augmentation described in (4.6.2)
7. splits the data into training/testing/validation sets as proposed in (4.5)

5.2.3 Module model

It handles model building, image inference, drawing the detections into the image and comparing the detections against grounding truths for testing purposes. The model includes:

1. `class BBox` - represents the bounding box.

2. `class Detection` - represents only one detection in image.

3. `class Inference` - represents one inference made by the model, containing multiple detections. It implements the non-maximum suppression described in (2.7.2) and handles the drawing of detections into the image.

4. `class Model` - represents the trained model, handles model re-construction from configuration files and creates the Inference instance.

5. `class Model` - represents the trained model, handles model re-construction from configuration files and creates the Inference instance.

6. `class GroundingTruth` - represents the grounding truths labels loaded from annotation files for testing purposes.

7. `class Tester` - tests and compares the detections made by the model against the grounding truths.

5.2.4 Module object_tracking

The module contains two classes:

1. `class VisDataPoint` - representing one tracked object in the scene

2. `class Tracker` - implements the centroid tracking algorithm described in (2.5.1)

Also the `test_tracker` function for testing of the `Tracker` logic is included.

5.2.5 Module app

Includes one `class App` that handles the overall running of the application, collects the arguments inputted by user, loads the chosen model, sets the parameters, loads the video file and generates the annotated video and timeline chart.

5.2.6 Input arguments

The application can be evoked by running `python app.py -m <model_type> -i <path_to_input_video> -s <frequency>`. Meaning of this arguments:

1. `-m, --model` - type of the model to be loaded, either ”tiny” for YoloV4-tiny or ”yolov4” for YoloV4, default=”yolov4”
2. `-i, --input` - path to the input video file, required
3. `-f, --freq` - integer number that sets how often will the inference be performed, it will be run only on every n-th frame for speed improvement, default=1 (every frame).

5.2.7 Output of the application

After the the whole video file is processed, the application generates the annotated video with detected object accompanied with the timeline chart.

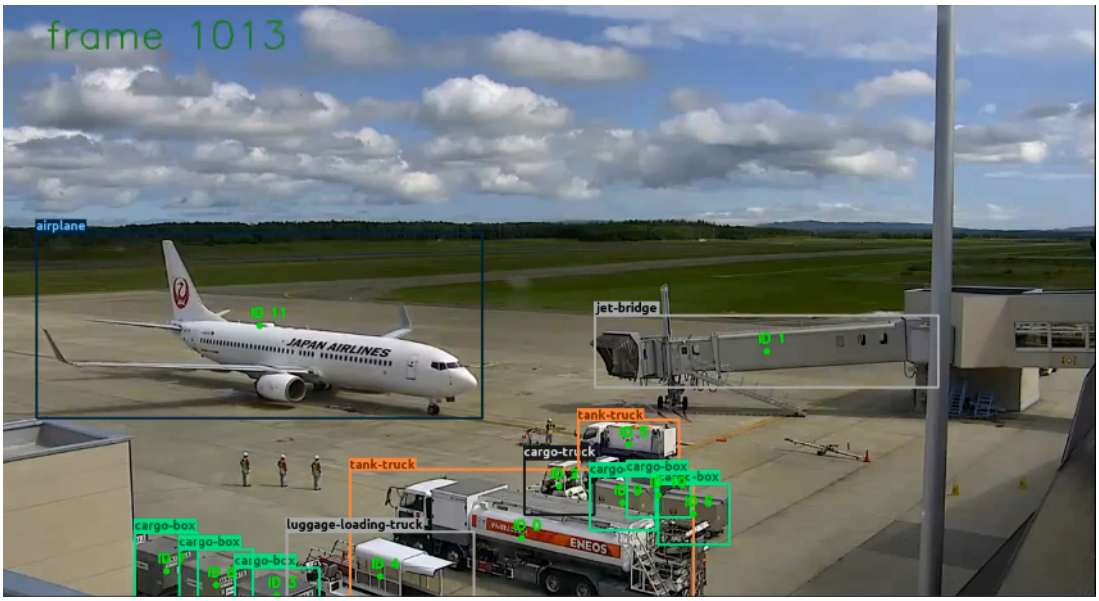


Figure 5.5: Outputted annotated video with detected and tracked objects (using video that the model has not seen before)

5.2.8 Visualization

The front-end part of the application serves as a visualization tool for rendering the results generated by the Python app in a web browser. The main page consists of two tabs: Detection tab - rendering the annotated video and Dashboard tab rendering the timeline chart. It is written in ReactJS [24], and it was build using the create-react-app tool [31] which according to the

documentation [32] is a comfortable environment for learning React, and is the best way to start building a new single-page application in React. After the generation of the annotated video and the timeline chart by the Python application, the results can be shown in the browser by command: `npm start` in the front-end/my-app directory of the repository.

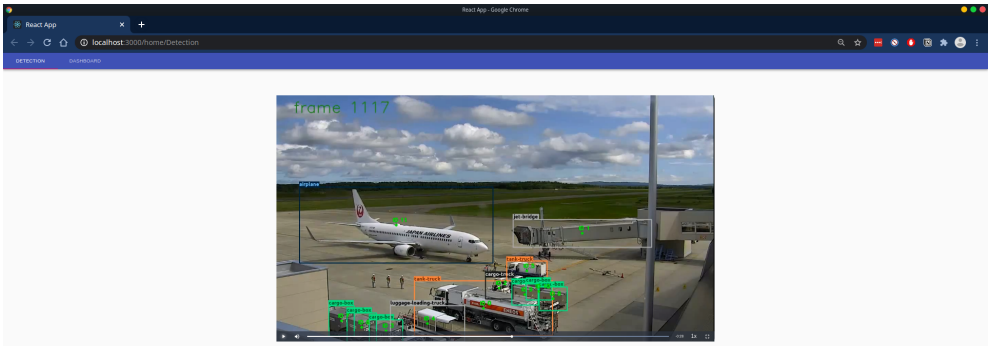


Figure 5.6: Detection tab, visualizing the annotated video in the browser

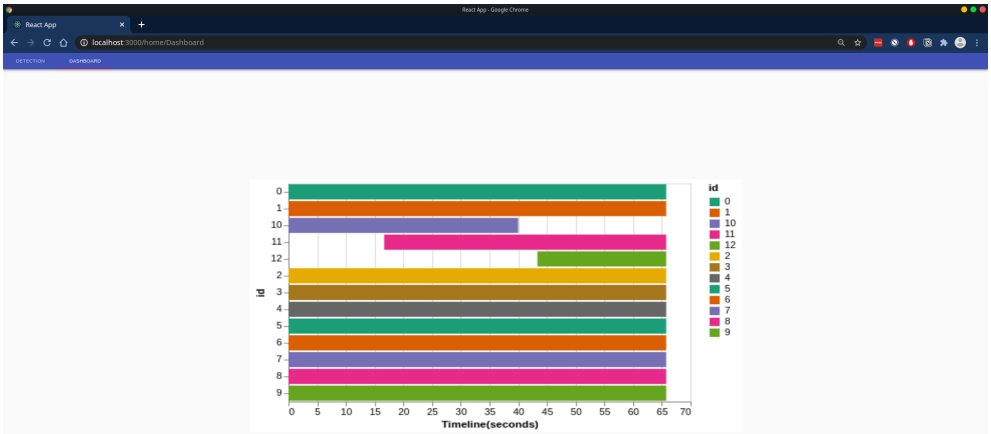


Figure 5.7: Dashboard tab, visualizing the timeline of detected objects in the browser

Conclusion

During the course of this thesis, we described the main goals and researched the current state-of-the-art methods and models that could be used for reaching these goals. We created the custom airport apron data set and proposed two different models: YoloV4 and YoloV4-tiny. We trained them on this newly created data set and evaluated the results. The models were then deployed into the application that was designed, implemented, and tested. Resulting in the final application that can process APR recordings and automatically detect the present key objects. In addition, the application generates a timeline chart with detected objects and renders the results in the web browser.

Discussion

The proposed models seem like a good fit for the airport environment since they can accurately detect the key APR objects in the scenes that the models have never seen before. However, it will be interesting to see how the models will perform on a significantly different APR recordings.

Proposed Improvements

A real-time camera system could be implemented since the FPS on GPU match the real-time requirement. Besides, used object detectors could be replaced with different models or even with Action Recognition models [38] to figure out if they are applicable for the airport environment.

Bibliography

[1] CHOLLET, François. Deep Learning with Python. Manning Publications Company, 2018. ISBN 9781617294433

[2] JOHNSON, Justin, Fei-Fei LI a Serena YEUNG. CS231 Lecture 11 - Detection and Segmentation, figure on slide 94 [online]. In: . [cit. 2021-01-28]. Avalaible at: http://cs231n.stanford.edu/slides/2017/cs231n_2017_lecture11.pdf

[3] JOHNSON, Justin. Stanford University School of Engineering - Lecture 11: Detection and Segmentation [online]. 2017 [cit. 2020-11-02]. Avalaible at: http://cs231n.stanford.edu/slides/2017/cs231n_2017_lecture11.pdf. Lecture material. Stanford University School of Engineering.

[4] GIRSHICK, Ross a Jeff DONAHUE. Rich feature hierarchies for accurate object detection and semantic segmentation Tech report (v5) [online]. 2014 [cit. 2020-11-17]. Avalaible at: https://arxiv.org/pdf/1311.2524.pdf. UC Berkeley.

[5] GIRSHICK, Ross. Fast R-CNN [online]. 2015 [cit. 2020-11-17]. Avalaible at: https://arxiv.org/abs/1504.08083. UC Berkeley.

[6] REN, Shaoqing, Kaiming HE, Ross GIRSHICK a Jian SUN. Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks [online]. 2016 [cit. 2020-11-18]. Avalaible at: https://arxiv.org/pdf/1506.01497.pdf

[7] REDMON, Joseph, Santosh DIVVALA, Ross GIRSHICK a Ali FARHADI. You Only Look Once: Unified, Real-Time Object Detection [online]. 2016 [cit. 2020-11-19]. Avalaible at: https://arxiv.org/pdf/1506.02640.pdf University of Washington, Allen Institute for AI, Facebook AI Research.

[8] REDMON, Joseph, Farhadi, Ali. YOLOv3: An Incremental Improvement [online]. 2016 [cit. 2020-11-20]. Avalaible at: https://arxiv.org/pdf/1804.02767.pdf University of Washington

[9] GANDHI, Rohith. R-CNN, Fast R-CNN, Faster R-CNN, YOLO — Object Detection Algorithms [online]. 2018 [cit. 2020-11-19]. Avalaible at: https://towardsdatascience.com/r-cnn-fast-r-cnn-faster-r-cnn-yolo-object-detection-algorithms-36d53571365e

[10] Papers with code object detection categorisation [online]. [cit. 2020-11-5]. Avalaible at: https://paperswithcode.com/task/object-detection/codeless

[11] GANDHI, Rohith. Support Vector Machine — Introduction to Machine Learning Algorithms [online]. [cit. 2020-11-22]. Avalaible at: https://towardsdatascience.com/support-vector-machine-introduction-to-machine-learning-algorithms-934a444fca47

[12] U.S DEPARTMENT OF TRANSPORTATION. Advisory circular, Airport apron definition [online]. [cit. 2021-01-15]. Avalaible at: https://www.faa.gov/documentLibrary/media/Advisory_Circular/AC%20120-57A.pdf

[13] Hong Kong Airport with ATC Live Stream by Full Flights and Airports HD [online]. [cit. 2021-02-09]. Avalaible at: https://www.youtube.com/watch?v=z26XibRtqCk&ab_channel=FullFlightsandAirportsHD

[14] Japan airport video footage provided by Profinit. Not publicly available.

[15] AlexeyAB. Darknet repository. [software]. [cit. 2021-02-09]. Avalaible at: https://github.com/AlexeyAB/darknet

[16] AlexeyAB. Darknet repository documentation. [online]. [cit. 2021-02-09]. Avalaible at: https://github.com/AlexeyAB/darknet/blob/master/README.md

[17] Computer Vision Annotation Tool (CVAT), Readme.md [online]. [cit. 2021-02-09]. Avalaible at: https://github.com/openvinotoolkit/cvat/blob/develop/README.md

[18] BOCHKOVSKIY, Alexey, Chien-Yao WANG a Hong-Yuan Mark LIAO. YOLOv4: Optimal Speed and Accuracy of Object Detection [online]. [cit. 2021-02-09]. Avalaible at: https://arxiv.org/pdf/2004.10934.pdf

[19] LIN, Tsung-Yi a Michael MAIRE. Microsoft COCO: Common Objects in Context [online]. [cit. 2021-01-12]. Avalaible at: https://arxiv.org/pdf/1405.0312.pdf

[20] BOCHKOVSKIY, Alexey. YOLOv4-tiny released: 40.2% AP50, 371 FPS (GTX 1080 Ti), 1770 FPS tkDNN/TensorRT [online]. [cit. 2021-02-10]. Avalaible at: https://github.com/AlexeyAB/darknet/issues/6067

[21] SOLAWETZ, Jacob. Train YOLOv4-tiny on Custom Data - Lightning Fast Object Detection [online]. [cit. 2021-02-10]. Avalaible at: https://blog.roboflow.com/train-yolov4-tiny-on-custom-data-lighting-fast-detection/

[22] Open Source Computer Vision Library [software]. [cit. 2021-02-8]. Avalaible at: https://github.com/opencv/opencv

[23] Altair, statistical visualization library for Python. [software]. [cit. 2021-02-8]. Avalaible at: https://github.com/altair-viz/altair

[24] Facebook and community. React A JavaScript library for building user interfaces. [software]. [cit. 2021-02-8]. Avalaible at: https://www.npmjs.com/package/react

[25] PAPERS WITH CODE. Object Tracking definition [online]. [cit. 2021-02-10]. Avalaible at: https://paperswithcode.com/task/object-tracking

[26] PAPERS WITH CODE. Non Maximum Suppression definition [online]. [cit. 2021-02-10]. Avalaible at: https://paperswithcode.com/method/non-maximum-suppression

[27] Redmon, Joseph. Darknet repository. [software]. [cit. 2021-02-09]. Avalaible at: https://github.com/pjreddie/darknet

[28] AlexeyAB. Darknet repository. yolov4-custom configuration file. [software]. [cit. 2021-02-09]. Avalaible at: https://github.com/AlexeyAB/darknet/blob/master/cfg/yolov4-custom.cfg

[29] Solawetz, Jacob. What is Mean Average Precision (mAP) in Object Detection? [online]. [cit. 2021-02-10]. Avalaible at: https://blog.roboflow.com/mean-average-precision/

[30] Heras, Jónathan. CLoDSA - an open-source image augmentation library [software]. [cit. 2021-02-10]. Avalaible at: https://github.com/joheras/CLoDSA

[31] Facebook. create-react-app. [software]. [cit. 2021-02-10]. Avalaible at: https://github.com/facebook/create-react-app

[32] Facebook. create-react-app documentation. [online]. [cit. 2021-02-10]. https://reactjs.org/docs/create-a-new-react-app.html

[33] Brownlee, Jason. How to Use ROC Curves and Precision-Recall Curves for Classification in Python. [online]. [cit. 2021-02-10]. Avalaible at: https://machinelearningmastery.com/roc-curves-and-precision-recall-curves-for-classification-in-python/

[34] Srivastava, Aakash. Label Smoothing: An ingredient of higher model accuracy. [online]. [cit. 2021-02-10]. Avalaible at:https://towardsdatascience.com/label-smoothing-making-model-robust-to-incorrect-labels-2fae037ffbd0

[35] Diganta, Misra. Mish: A Self Regularized Non-Monotonic Activation Function. [online]. [cit. 2021-02-10]. Avalaible at:https://arxiv.org/pdf/1908.08681.pdf

[36] Wikipedia. Euclidean distance. [online]. [cit. 2021-02-10]. Avalaible at:https://en.wikipedia.org/wiki/Euclidean_distance

[37] Research center for informatics. [online]. [cit. 2021-02-10]. Avalaible at:http://rci.cvut.cz/

[38] Papers with code. Action Recognition benchmarks. [online]. [cit. 2021-02-10]. Avalaible at:https://paperswithcode.com/task/action-recognition-in-videos

[39] Wikipedia. Andrew Ng. [online]. [cit. 2021-02-10] https://en.wikipedia.org/wiki/Andrew_Ng

[40] He, Kaiming. Mask R-CNN. [online]. [cit. 2021-02-10]. Avalaible at:https://arxiv.org/pdf/1703.06870.pdf

[41] Cai, Zhaowei. Cascade R-CNN: Delving into High Quality Object Detection. [online]. [cit. 2021-02-10]. Avalaible at:https://arxiv.org/pdf/1712.00726.pdf

[42] Liu, Wei. SSD: Single Shot MultiBox Detector. [online]. [cit. 2021-02-10]. Avalaible at:https://arxiv.org/pdf/1512.02325.pdf

Appendix A

Acronyms

AI Artificial intelligence

DL Deep learning

CV Computer vision

IC Image classification

SEM Semantic segmentation

OD Object detection

IS Instance segmentation

APR Airport apron

OT Object tracking

mAP Mean average precision

AP Average precision

FP False positive

FN False negative

TP True positive

TN True negative

NMS Non-maximum suppression

CNN Convolutional neural network

Appendix B

Contents of enclosed SD

- `readme.txt` ..... the file with SD contents description
- `models` ..... the directory with saved models
  - `yoloV4` ..... the directory with saved yoloV4 model
    - `yoloV4.cfg` ..... .cfg file containg the configuration of the model
    - `yoloV4.weights` ..... the file containing trained weights
  - `yoloV4-tiny` ..... the directory with saved yoloV4-tiny model
    - `yoloV4-tiny.cfg` . .cfg file containg the configuration of the model
    - `yoloV4-tiny.weights` ..... the file containing trained weights
- `letisni-stojanka` .. the github repository containing the source codes of the application
- `thesis.pdf` ..... the thesis text in PDF format