Université de Montréal - MILA - Ville de Montréal

# Object Detection in Urban Context

by

**Jean-Sébastien Grondin, ing.**

Computer Science Department
Faculty of Arts and Sciences

Internship report for
Master's of Science (M.Sc.)
in Computer Science

Specialization: Machine Learning

November 20, 2020

## Abstract

The purpose of this project was to implement, train and evaluate two recent deep neural network solutions for real-time object detection in urban scenes. More specifically, the objective was to detect five different types of objects, namely vehicles, pedestrians, buses, cyclists and construction items (e.g. cones, fences). This project is a stepping stone towards a longer term goal of being able to perform automatic anomaly detection in Montreal city surveillance camera feeds and support CGMU (Centre de Gestion de la Mobilité Urbaine) operators with detecting incidents as soon as they arise.

At the beginning of this project, 10,000 labeled images from a previous project on semantic segmentation were available and used as a starting point. An internal annotation campaign was conducted to create the object detection annotations for the same images. Additional datasets with similar contexts were also obtained and added to the training set.

Three different neural architecture families were investigated: a Single-Shot Multibox Detector (SSD) which was chosen as a baseline, as well as YOLOv5 and EfficientDet, both released in 2020.

Unfortunately, EfficientDet was found to be extremely difficult to train. A viable training configuration that achieves a somewhat smooth and decreasing training loss could not be found. At that time, this model was still under active development and other users were raising similar issues. A decision was made to focus on improving the performance further for YOLOv5 models instead, and it was decided that the evaluation of EfficientDet would be put on hold.

In contrast, YOLOv5's performance exceeded expectations. The project delivered a YOLOv5 model solution that can be used to infer object detections in real time, and that is compatible with the type of cameras used by the CGMU. This solution was also shown to be robust to visual artefacts that are frequent in outdoor settings (e.g. sun glare, rain drops on lens, blurry images) and to different day and weather conditions (e.g. day, night, rain, snow, etc).

Hence, all project requirements were met. In addition, the project delivered value that extended outside of the project scope, with the release of a brand new urban scene object detection dataset and fully trained YOLOv5 model solutions to the public. In addition, a rigorous hyperparameter search pipeline that can easily be adapted to future machine learning projects was implemented.

## Contents

## List of tables

## List of figures

## List of acronyms

| | |
|---|---|
| ASHA | *Asynchronous Successive Halving Algorithm.* The hyperparameter search algorithm used for this project. |
| CGMU | *Centre de Gestion de la Mobilité Urbaine.* CGMU is the department in charge of smart transport at Ville de Montréal. They own the cameras used for this project. |
| CNN | *Convolutional Neural Network.* |
| CPU | *Central Processing Unit.* |
| CSPNet | *Cross Stage Partial Networks.* |
| CVAT | *Computer Vision Annotation Tool.* |
| FPN | *Feature Pyramid Network.* |
| FP | *False Positive.* |
| FN | *False Negative.* |
| GAN | *Generative Adversarial Network.* |
| GPU | *Graphical Processor Unit.* |

| | |
|---|---|
| mAP | *Mean Average Precision.* |
| MOT | *Multi-Object Tracking.* |
| NMS | *Non-Maximum Suppression.* |
| PTZ | *Pan-Tilt-Zoom.* The cameras that are used can be remotely controlled to pan left or right, tilt up or down and zoom to enlarge a section of the field of view. |
| ReLU | *Rectified Linear Unit.* |
| SGD | *Stochastic Gradient Descient.* |
| SSD | *Single-Shot multibox Detector.* The baseline model used for this study. |
| TP | *True Positive.* |
| TN | *True Negative.* |
| TTA | *Test Time Augmentation.* |
| YOLOv5 | *You-Only-Look-Once - 5th version.* One of two state-of-the-art models tested in this study. |

## Acknowledgements

First and foremost I would like to thank Marie-Odette St-Hilaire, the Data Scientist at Ville de Montréal leading this project, for her guidance and support throughout, but also for being a great mentor. Her enthusiasm and sense of humor added considerably to my internship experience.

I would also like to thank my supervisor Michel Charest for helping clearing obstacles, for providing supportive feedback, as well as for securing the resources necessary to undertake this project.

I want to also thank Vincent Michalski, my technical supervisor at MILA, whose insight and knowledge into the subject matter steered me through this project.

I want to give special thanks to Marie-Odette St-Hilaire, Carl Genest, Vincent Du and Basile Roth who have been an invaluable help during the annotation campaign.

Finally, I would also like to thank other colleagues in the Architecture and Data Science team, for providing a relaxed and enjoyable environment, despite the COVID-19 pandemic.

# Chapter 1

## Introduction

In Montreal, the *Centre de Gestion de la Mobilité Urbaine* (CGMU) is responsible for monitoring and managing the city road network in real time, as well as quickly reacting when an incident occurs. It is meant to be the heart and brain of smart mobility systems in Montreal. Its monitoring equipment consists in over 500 Pan-Tilt-Zoom (PTZ) cameras that are installed on various roads and intersections on the Montreal territory. These cameras can be rotated at various angles and zoomed in or out depending on the need of CGMU operators.

Currently, CGMU operators cannot simultaneously monitor all camera feeds in real time, which may cause some delays between the time of an incident and its detection, leading to aggravated congestion. With recent breakthroughs in computer vision and deep learning, there is now an opportunity for creating a system that is fueled by artificial intelligence and that can support CGMU operators with automatic detection of different types of anomalies on the network (e.g. accidents, vehicle breakdown, vehicle blocking bicycle path, etc).

A roadmap towards a deep learning based anomaly detection solution was proposed in [**8**], and is illustrated in Figure 1.1. The necessary technology was divided into three parts:

(1) **Location :** As a first step, the objective is understanding the context of the urban scene, in other words what is contained in the camera feed. This first module was implemented in 2019 with the development of a semantic segmentation solution [**6**] [**8**] . The latter enables generating masks that can be used to extract zones of interest (e.g. roads, sidewalks) and could also eventually be used to infer the orientation and pose of PTZ cameras.

(2) **Object :** Another important step towards achieving anomaly detection is being able to detect objects of various types. An object detection module is necessary to localize and classify moving objects like vehicles, pedestrians and cyclists and static objects like construction objects.

(3) **Movement :** The next logical step consists in detecting the trajectory of dynamic objects, in other words tracking objects over time and space.

This triplet of information, composed of location, objects and movement, can be used to detect anomalies. One example would be identifying one single car (object) that is static (movement) on a highway (location)

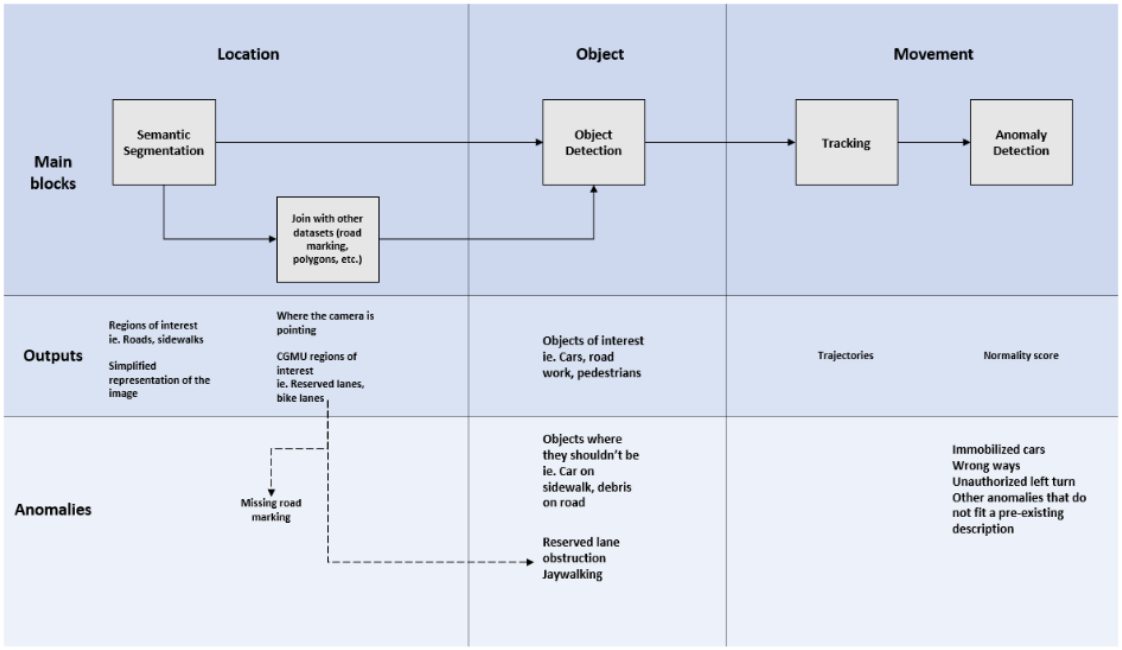



**Fig. 1.1.** Roadmap for anomaly detection at Ville de Montréal. This project and report focus on the object detection module. Source: [**8**]

when other cars are driving by (object-movement). This could be highlighted to CGMU operators as a potential car breakdown on the highway.

This report presents the outcome of a second project that took place in 2020, and which focused on the development of the object detection module. More specifically, the goal of this project was to propose, implement, train and evaluate two different model solutions for detecting five different types of objects in CGMU surveillance camera feeds, namely vehicles, pedestrians, buses, cyclists and construction objects. For this solution to be successful, it had to meet the following requirements:

(1) Run in real-time or near real-time on reasonable hardware, i.e. at at least 5 frames per second, or one frame every 200 ms.
(2) Be trainable with one GPU (Graphical Processor Unit) in reasonable time.
(3) Be trainable with existing labeled dataset consisting of 10,000 images.
(4) Be compatible with all existing camera makes and models, and adaptable to cameras that will be acquired in the future.
(5) Be compatible with PTZ cameras which are never guaranteed to be in same orientation or zoom setting.
(6) Be robust to images of different resolutions and to visual artifacts (e.g. out of focus, rain drop or dust on lens, sun glare, etc).
(7) Have good performance under various day and weather conditions (i.e. day and night, different cloud coverage, rain, snow, etc).

As an additional goal for this project, it was decided that all code, models and data would become open-source so that anybody can decide to reproduce results and make improvements to the proposed solution.

Therefore, this report aims at providing all the necessary details to enable reproducing any of the results discussed herein. Chapter 2 consists in a review of the literature and covers all of the essential concepts necessary for reading through and understanding the outcomes of this report. It is meant for readers who are not necessarily familiar with deep learning and object detection but who have a basic understanding of convolutional neural networks. It also presents the models that were selected and the rationale for their selection.

Then, Chapter 3 presents all the details and relevant statistics of the data used for training and evaluating models. This chapter also includes a complete description of the annotation campaign, which was necessary to obtain the right labels for this object detection task. Some additional datasets were also obtained and added to the CGMU dataset for training. All pre-processing steps for curating these datasets are explained.

Chapter 4 follows with a description of the methodology pertaining to each model that was investigated. It provides more details on all key aspects of the machine learning pipeline, including a description of models, hyperparameter search algorithms, the computing infrastructure used and a specification of dependencies.

Results are presented and discussed in Chapter 5, and possible next steps are discussed in Chapter 6.

# Chapter 2

# Literature Review

This chapter includes a description of the various components of object detection models and provide a list of options that were developed in recent years and considered for this project. It also presents some of the important techniques that are necessary to achieve good detection performance. The final selection of models and techniques to be used for this project, as well as the rationale, are also presented in this chapter.

The assumption is made that readers have a basic understanding of how convolutional neural networks (CNN) work. If that is not the case, they can use the following reference [**49**]. The survey articles [**32**][**43**] can also be good references for those who seek an understanding of how the field of object detection has evolved over recent years.

## 2.1. Anatomy of Object Detectors

Modern object detectors are generally composed of several parts, as shown in Figure 2.1: **a)** the *backbone* extracts relevant features from the input image and is usually trained on a large image dataset like ImageNet; **b)** the *head* generates predictions for object classes and bounding boxes; and **c)** [optionally] some models will include a *neck*, i.e. some layers in between that will often be used to collect and combine feature maps from different stages of the neural network. These three parts can be seen as building blocks that can be carefully selected and combined to meet specific applications requirements. Generally, backbones that yield higher accuracy tend to be more computationally hungry. Equivalently, head and neck architectures with larger numbers of trainable parameters tend to be less efficient and less amenable for real-time applications. Besides the trade-off between accuracy and efficiency, other considerations are also important when selecting these parts: for example how accurately they can detect small objects, or how sensitive they are to noise and artefacts of different kinds (e.g. occlusion, low light, etc.).

### 2.1.1. Backbone

The backbone network is also commonly referred to as the feature extractor, as it is there to convert input images into meaningful feature maps, or representations. A backbone network can be trained from scratch but it is very expensive to do so and datasets of sufficient sizes are required, which is why in practice this is rarely done. Usually, the backbone network used for object detection is a deep convolutional network that was trained on an image classification task on a very large dataset like ImageNet [**15**], OpenImages [**35**] or on object detection tasks like MS COCO [**42**], and from which we remove the last fully connected layer(s). From there we can either use it as a fixed feature extractor , or we can decide to continue fine-tuning the

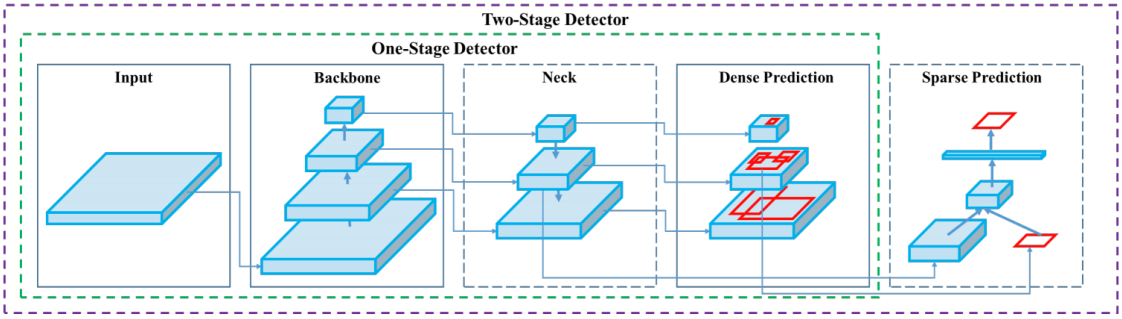



**Fig. 2.1.** General anatomy of object detection model. Two different types of heads are displayed: a) a dense prediction head and b) a sparse prediction head. Source: [**9**]

network by continuing backpropagation in subsequent training tasks. It is also possible to keep some layers fixed and only fine-tune some of the higher-level layers of the network. This is motivated by the fact that low-level layers tend to learn generic features (e.g. edges) while higher-level layers learn features that are progressively more specific to the type of images contained in the original dataset (e.g. wheels and chassis for a cars dataset) [**43**].

Some backbone networks like ResNet [**26**], ResNeXt [**65**] and Inception Resnet v2 [**56**] are deeper and/or more densely connected networks, which make them more computationally expensive. Other networks like MobileNet [**28**], ShuffleNet [**46**][**70**] and SqueezeNet [**30**] are more lightweight by design and can be used for inference in mobile applications or when GPUs cannot be used for inference. Networks like VGG-16 [**54**], Inception v2 and v3 [**58**] generally lie in between these two categories in terms of efficiency.

Recently, Cross Stage Partial Networks (CSPNet) [**61**], when integrated as backbones in object detection models, have enabled important inference computation reductions while significantly outperforming previous state of the art models on the MS COCO dataset. It became the backbone of choice for YOLOv4 [**9**] and YOLOv5 [**33**] object detection models, both released in 2020.

The same year, the Google Brain Team used neural architecture search to design EfficientNets [**59**], which achieved state of the art performance on ImageNet as well as 5 other transfer learning datasets while being an order of magnitude smaller and faster for inference than the next best existing convolutional network. The same authors, even proposed an object detection model that is using EfficientNet as its backbone. They named this object detection model EfficientDet [**60**].

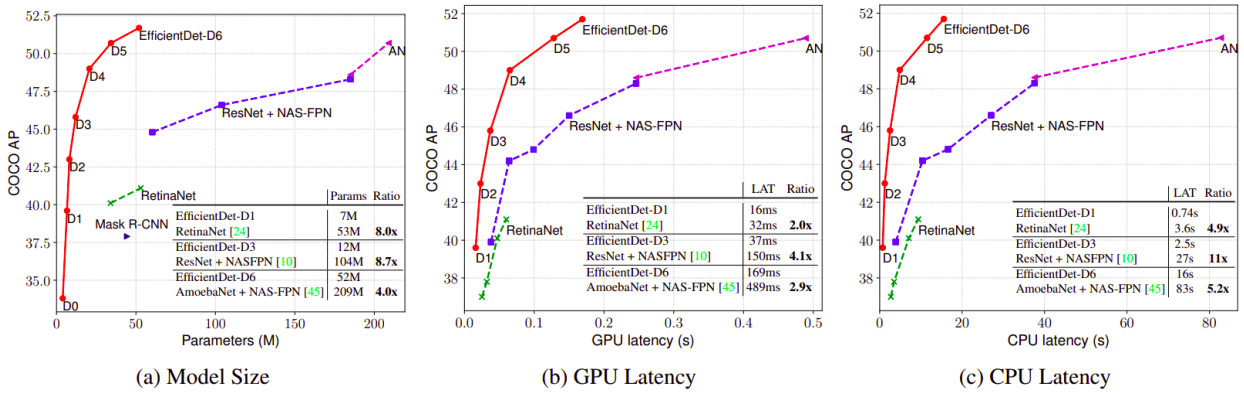
Given the project requirement of real-time inference, both the CSPNet and EfficientNet architectures seemed very promising.

### 2.1.2. Head

The head part is also commonly referred to as the meta-architecture and typically falls in two categories: two-stage detectors (i.e. sparse prediction) and one-stage detectors (i.e. dense prediction). The most representative two-stage detector is the region-based R-CNN [**23**] (2014) and its successors: Fast R-CNN (2015) [**22**], Faster R-CNN (2017) [**53**] and Mask R-CNN (2017) [**25**]. These models first predict region proposals and feed them to a convolutional network that subsequently classifies objects and predicts their

bounding boxes. For several years, two-stage detectors have been known to outperform one-stage detectors in accuracy, although recently single-stage detectors seems to have caught up in accuracy [**32**][**69**][**72**]. One key attribute that distinguishes two-stage from one-stage detectors is their greater computational complexity, which is why they are generally not well suited for real time applications. For example, the fastest two-stage detector runs at 7 frames per second (fps) or less on a GPU using a VGG-16 backbone while one-stage detectors can run as fast as 45 fps [**72**] with VGG-16. The earliest most representative one-stage detectors are YOLO (v1 in 2016) [**50**], Single-Shot Multibox Detector (SSD, 2016) [**44**] and RetinaNet (2017) [**41**]. One-stage detectors can classify objects and predict bounding boxes directly, without the need for a region proposal network. This were generally associated with smaller inference latency, however one-stage solutions have suffered from poorer accuracy for several years until recently.

In recent years, a plethora of one-stage detectors have been proposed in the literature to address some of their typical shortcomings. This can make a selection difficult. However, **YOLOv5** (2020) [**33**] and **EfficientDet** (2020) [**60**] were selected as they appeared to be good choices for several reasons. They seem to outperform most of the competing solutions both in accuracy and efficiency which is key to the success of this project. For example, Figure 2.2 show how for example EfficientDet outperforms other models on MS COCO for a given model size. Figure 2.3 shows how YOLOv5 (released slightly afterwards) outperforms EfficientDet on MS COCO. Both these models have also attracted a lot of interest from the Deep learning community and as a result there seems to be plenty of open-source implementations that are well reviewed and that could be used as starting points for this project. They also seem to have performances that are more robust to different scales and aspect ratios of objects, which is also important in this application (e.g. humans vs trucks, different zooms, etc). In contrast, the SSD is a simpler architecture and often used for baseline comparisons; also it is well known and documented in literature and thus easier to implement. As such, an **SSD** implementation was chosen as a baseline to determine if the YOLOv5 and EfficientDet truly offer improvements and if their added complexity is justified.




(a) Model Size (b) GPU Latency (c) CPU Latency

**Fig. 2.2.** Model size and inference latency comparison on MS COCO detection dataset - EfficientDet vs other models. Y axis: Average Precision on MS COCO. Source: [**60**]

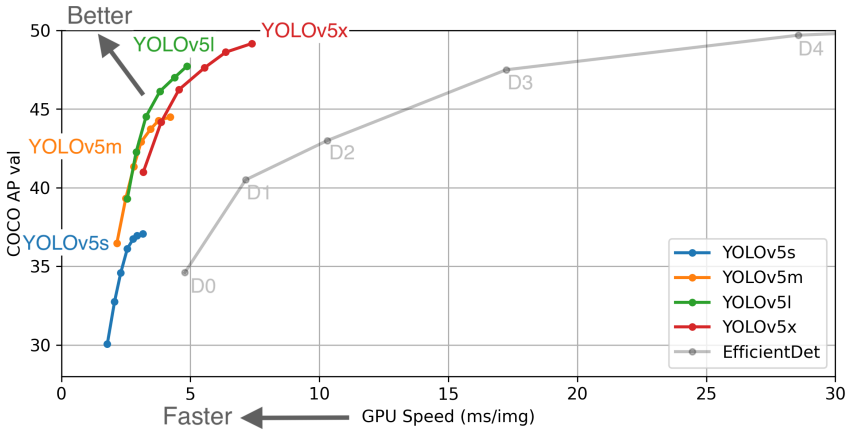



**Fig. 2.3.** Average Precision on MS COCO and inference speed - YOLOv5 vs EfficientDet. Source: [**33**]

## 2.2. Transfer Learning

In addition to using the CGMU labelled image dataset, there was a strong intuition that there may be supplemental datasets that could be used in a transfer learning scenario in order to improve the detector's performance. Recent studies [**37**] indicate that pre-training on large detection datasets can be very beneficial before fine-tuning on a small detection dataset. As discussed in subsection 2.1.1, it is common to use backbones that are pre-trained on a classification task using for example the ImageNet [**15**] or OpenImages [**35**] dataset. Since the task of interest is object detection, it may be beneficial to use a backbone that is pre-trained on a detection task like the ImageNet detection task or MS COCO [**42**] for example. ImageNet is composed of more than 14 million images, divided in 21,841 categories and with 1.5 objects per image on average. All images include object bounding boxes as well as their annotation label for their category. OpenImages include more than 9 million images, divided in over 6000 categories and with 8.3 objects per image on average, which makes it the largest generic image dataset at the moment. Annotations include image level labels, segmentation masks, visual relationships and object bounding boxes. MS COCO is composed of nearly 250K training images with object bounding boxes and labels from 100 different categories. It contains 3.5 categories and 7.7 objects per image on average.

In addition, another complementary transfer-learning strategy that was considered involved fine-tuning the network on a large detection task that is semantically close to the CGMU images. For example, the KITTI dataset [**20**] and the Caltech Pedestrian dataset [**18**] include 15k and 250k images respectively from car front view cameras. They are often used as benchmarks for car detection and/or pedestrian detection performance. However, due to the fact that these are images from a car front view perspective, they may not be the most adequate choice. Datasets that are more closely related to the CGMU dataset were found and are discussed in Chapter 3.

The specific transfer learning strategy used for each of the selected object detection models is explained in Section 4.


**Fig. 2.4.** More sophisticated types of augmentations. Source: [**9**]

## 2.3. Data Augmentation

Data augmentation strategies can be useful for increasing the effective number of examples seen during training and for making the model more robust to object scale, size and shapes as well as day and weather conditions since it effectively increases the variability of the input images. It also helps with reducing overfitting and improving generalization. For example, one method (suggested in [**44**]) consists in sampling training images randomly by either using the entire image or sampling a patch with varying size and aspect ratio. Other geometric distortion such as random scaling, flipping and rotating can also be effective [**9**].

Photometric distortions (e.g. brightness, contrast, hue, saturation or noise) can also be added. Another research has suggested combining multiple images together. MixUp [**68**] and CutMix [**67**], as shown in Figure 2.4 are two recent examples of this type of approach. For example, YOLOv4's implementation [**9**] uses CutMix built-in into its pipeline. Other recent research has also proposed learning data augmentation policies (e.g. AutoAugment) rather than using implementations that are manually designed [**13**][**75**]. These policies can then be randomly chosen and combined for each image. In [**13**], AutoAugment achieves state of the art accuracy on CIFAR-10, CIFAR-100, ImageNet. In [**75**], augmentation learning obtains state of the art performance on MS COCO and Pascal VOC. Both papers claim that learned policies can transfer well to other tasks and improve performance. Finally, style transfer was also suggested as another approach to data augmentation and has been shown to help reduce the bias of CNNs towards texture [**21**]. Data augmentations used for each model are discussed in Chapter 4 - Methodology.

## 2.4. Post-Processing Techniques

Object detectors will generate multiple overlapping bounding boxes for the same object. Post-processing techniques such as Non-Maximum Suppression (NMS) are required to only retain the best candidate bounding box. NMS is a method that is commonly used in deep learning based object detection and several variants were proposed over the years. The baseline technique that is often used is a Greedy NMS [**23**] which consists in keeping the most confident bounding box prediction among all boxes that are overlapping above a certain Intersection-Over-Union (IoU) threshold (see definition below) and that are of the same class. Amongst more recent alternatives, Soft NMS [**10**], Learning NMS [**27**] and DIoU-NMS [**73**] are popular. YOLOv4 and YOLOv5 use the latter in their implementation.

## 2.5. Evaluation Metrics

The most commonly used evaluation metric for object detection is the mean average precision (mAP) [**32**][**43**], which is a high level metric that is derived from a combination of the Intersection over Union (IoU), Precision and Recall. Let's first define these. Precision is the ratio of accurate positive predictions (TP) out of the total predicted positives (TP + FP).

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

Recall is the ratio of accurate positive predictions (TP) out of the total of positives (TP + FN).

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

The difference between a positive prediction (i.e. detecting an object) and a negative prediction (i.e. no detection) will depend on the model confidence and on whether it is above or below a certain confidence threshold. Varying this threshold will enable trading-off between these two metrics. For example, using a very small threshold of say 0.05 will result in many False Positives (FP) and very few False Negatives. Precision will be low and Recall will be high. A very high threshold of say 0.95 will result in the opposite, few FP, many FN, a high Precision and a low Recall. Thus, by varying the threshold between 0 and 1, we can obtain different pair values for Precision and Recall. The Average Precision is the area under the curve generated by these Precision - Recall pairs. The Average Precision is also always relative to an IoU criterion or threshold. As shown in Figure 2.5, the IoU is the ratio (between 0 and 1) of overlapping areas between the predicted and ground truth bounding box.

$$IoU(A,B) = \frac{A \cap B}{A \cup B}$$

Standard IoU thresholds used are usually 0.5 or 0.75. A predicted detection will be regarded as a True Positive if the IoU is greater than the predefined threshold (e.g. 0.5 or 0.75). The AP is computed for each

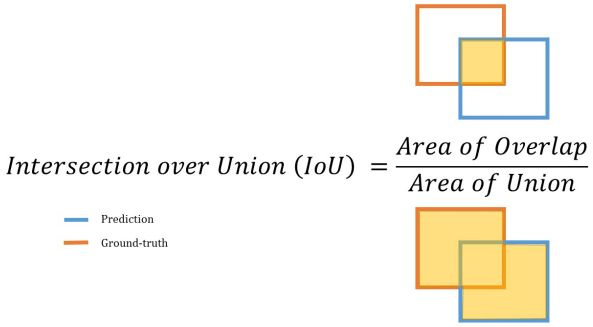



**Fig. 2.5.** Intersection over Union. Reference: *kaggle.com*

object class, but the number of class is usually high, thus it is common to report the mean average precision (mAP) of all $N$ classes.

$$mAP = \frac{1}{N} \sum_{i}^{N} AP_i$$

In summary, the mAP @ 0.5 IoU indicates the mean average precision using an IoU threshold of 0.5. In contrast, the mAP @ 0.05:0.95 IoU is the mAP averaged across 0.05 increments of IoU thresholds between 0.05 and 0.95. Finally, some research papers will also include the mAP for small, medium and large objects so as to show how the performance is affected by the object size.

## 2.6. Challenges of object detection in urban context

**Poor image quality** is a common problem with city camera footage and can be caused by low lighting, low quality devices, or simply by weather (e.g. dust, rain, etc). In literature, there is still limited work to address this problem. The research of Dimou et al. [**17**] looked into using CCTV footage for tracking applications and has faced the problem of blurred images, which was tackled by augmenting the training data with blurred examples to make the object detector more robust to poor image quality.

**Object occlusion** is another issue that is common in an urban context (e.g. construction items partially hidden behind cars and trucks, or vegetation blocking a camera field of view). Occluded pedestrians are notoriously hard to detect. And this problem is still far from being solved [**43**]. One way to approach it is by simulating object occlusions and including more examples in the training set via data augmentation. Methods like random erase [**74**] and CutOut [**16**], which effectively consist in randomly selecting a path in the image and filling it with a random value or with zeros have shown positive results. Hide-and-seek [**55**] and GridMask [**12**] do similar manipulations but on multiple rectangle regions in an image. Other studies have attempted to use adversarial networks (GANs) to generate examples with occlusions [**63**]. Another research [**62**] was able to significantly improve performance with people occlusion by using an attention mechanism across CNN channels to represent various occlusions patterns as combinations of body parts.

# Chapter 3

# Data

## 3.1. CGMU Dataset

At the beginning of the project, a semantic segmentation dataset including 10,000 images was available [**5**]. This dataset is described in subsection 3.1.1. Object detection annotations were then obtained for the same 10,000 images. The procedure used for doing so is detailed in subsection 3.1.2

### 3.1.1. Semantic Segmentation

Semantically segmenting an image consists in allocating a class to every pixel in an image, among a predefined list of classes. Semantic segmentation annotations had already been obtained via a third party service provider in 2019 in preparation for the first project which focused on the Location technology module of Figure 1.1). The following is a list of all classes that are included in this dataset.

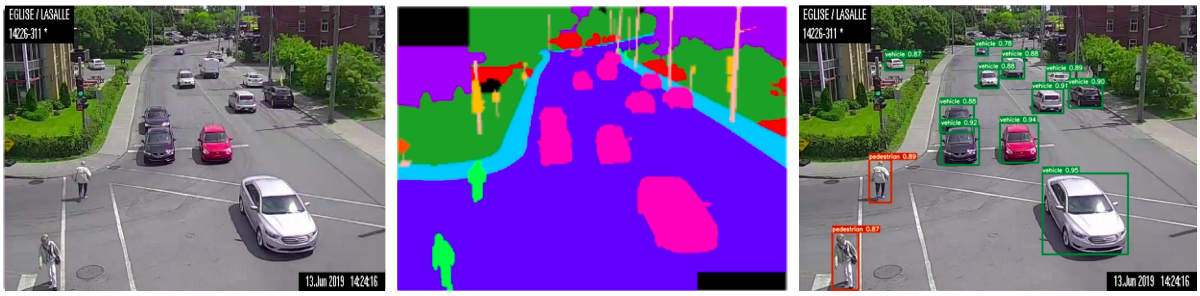
- Void
- Pole
- Traffic Sign
- Vehicle
- Vegetation
- Median Strip
- Building
- Private
- Sidewalk
- Road
- Pedestrian
- Structure
- Construction

More details on the 2019 annotation campaign can be obtained in the previous project report [**8**], but here is a brief summary. A total of 74 cameras were selected based on how well they represented the total population of CGMU's surveillance cameras. As a result, two different camera resolutions were sampled. The amount of images from each resolution is as follows:

- **Large resolutions:** 4326 images with size 352x288 pixels and 352x240 pixels.
- **Small resolution:** 5672 images with size 704x480 pixels.

A total of 20 days were selected randomly between May 24th and July 23rd and for each day, ten images were sampled randomly. Then a manual vetting of the image was performed to remove outlier images (e.g. completely obstructed images, night time images that were too dark, etc). One typical example of a CGMU image with semantic segmentation labels is shown in Figure 3.1 a) and b).

It should be noted that all labeled images available for this project had been previously anonymized



(a) Original image (b) Semantic Segmentation (c) Object Detection

**Fig. 3.1.** CGMU dataset: from semantic segmentation to object detection

[**5**], therefore no additional protection regarding privacy was necessary.

The next subsection indicates how object detection annotations were obtained from this semantic segmentation dataset.

### 3.1.2. Object Detection

The goal of this project was to perform object detection, which is equivalent to localizing and classifying objects. Training a model to perform this task requires a different type of annotations. Each training image requires bounding box coordinates for all relevant objects contained in it, along with their class label. An example image with such annotation is shown in Figure 3.1 c).

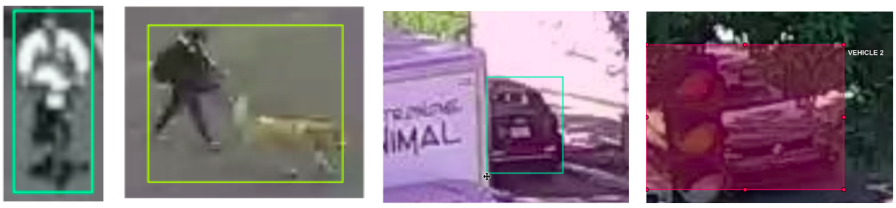
Hence, an annotation campaign was organised internally using the annotation tool *CVAT* [**2**]. To save time, an automatic pre-annotation of images was performed by extracting the coordinates of bounding boxes from the contours of contiguous pixels from the same classes, as depicted in Figure 3.2. However, since the segmentation does not make a distinction between different object instances, this technique was not able to distinguish between overlapping objects of the same class. Ultimately, the pre-annotation tool was used but the large majority of bounding boxes had to be created by hand using *CVAT*.

(a) Semantic Segmentation (b) Automatic Pre-Annotation

**Fig. 3.2.** Automatic conversion of semantic segmentation into object detection

The annotation work was split between five annotators. To ensure consistency, the following annotation policy was used for these special cases:

- For cyclists, a single bounding box was annotated around the bicycle including the person riding it, and the label "cyclist" was used. (See Figure 3.3a)
- For pedestrians walking next to their bicycle, a single bounding box was annotated around the pedestrian including the bicycle, and the label "pedestrian" was used. Equivalently for a pedestrian walking a dog, or carrying an object, the bounding box was drawn around the person including the pet/object. (See Figure 3.3b)
- For people riding motorcycles or scooters, the label "vehicle" was used.
- For people riding kick scooters, unicycles or electric bicycles, the label "cyclist" was used.
- For occluded objects (i.e. not entirely visible), a bounding box was drawn around their visible parts (as opposed to extrapolated). See Figure 3.3c).
- For objects that were split into several visible parts due to occlusion, a single bounding box was drawn around all visible parts. See Figure 3.3d).




(a) Cyclist (b) Pedestrian (c) Occluded Car (d) Occluded Car

**Fig. 3.3.** Examples of annotation policy used for labeling ground truth bounding boxes and classes.

Then a manual verification of all 10,000 images was performed by the lead annotator to assess annotation consistency and quality and rectify as needed. Once the annotation campaign was completed, the whole dataset was then split into four parts:

- 70% of images for training,
- 10% for validation,
- 10% for in-domain test and
- 10% for out-domain test.

The purpose of the training set was for learning model weights and biases. The validation set was used at training time to monitor model performance and determine when the model started overfitting the training data. It was also used for comparing hyperparameter configurations and finding ones that achieve better performance. Finally, test splits were used only at the end of the project, once a model configuration had been selected, to obtain an unbiased measure of model performance.

Training, validation and in-domain test splits all use the same sub-population of cameras (i.e. intersections) while the out-domain test split uses a reserved population of cameras that are not seen during training nor validation. The rationale for using this particular split was to be able to discern at test time

| | Average Number of Objects / Image | | | | | | % Images | |
|---|---|---|---|---|---|---|---|---|
| | vehicles | pedestrian | construction | cyclist | bus | total | Low Res. | High Res. |
| **Training** | 8.78 | 1.51 | 1.23 | 0.12 | 0.14 | 11.77 | 43 | 57 |
| **Validation** | 8.76 | 1.57 | 1.21 | 0.14 | 0.18 | 11.87 | 43 | 43 |
| **Test (in-domain)** | 8.79 | 1.64 | 1.17 | 0.13 | 0.15 | 11.88 | 43 | 43 |
| **Test (out-domain)** | 12.82 | 1.97 | 1.8 | 0.08 | 0.14 | 16.81 | 49 | 51 |

**Table 3.1.** Number of objects per image in each data split - CGMU dataset

| | Images | Number of Objects Seen in Dataset | | | | | |
|---|---|---|---|---|---|---|---|
| | Total | vehicles | pedestrian | construction | cyclists | bus | total |
| **Training** | 7007 | 61521 | 10581 | 8619 | 841 | 981 | 82472 |
| **Validation** | 1000 | 8760 | 1570 | 1210 | 140 | 180 | 11870 |
| **Test (in-domain)** | 1000 | 8790 | 1640 | 1170 | 130 | 150 | 11880 |
| **Test (out-domain)** | 991 | 12705 | 1952 | 1784 | 79 | 139 | 16659 |

**Table 3.2.** Number of images and objects from different classes in each data split - CGMU dataset

the model performance on images from intersections that were never seen before as opposed to intersections that were seen during training. Another benefit of using this split was to have a more similar distribution of classes in the training and validation sets.

Table 3.1 shows the average number of objects per image for each data split. By design, the training, validation and in-domain test splits were created by randomly selecting images from the same pool of cameras (i.e. intersections), which is why their distribution of object types and image resolutions are very similar. The cameras which were most closely aligned with these distributions were selected for the out-domain test split, but some larger differences can be noted in the distributions due to the fact that it is a smaller sample of images.

Table 3.2 provide more details on the number of images and total number of objects from each type in each data split. It can be noted that the distribution of classes is unbalanced, with a far greater number of vehicles than cyclists and buses for example. The fact that fewer than 1000 examples of cyclists and buses are seen at training time could lead to poor detection performance for these classes during inference. Therefore, additional datasets were obtained and are described in the following section.

## 3.2. Additional Datasets

There is not a vast amount of publicly available object detection datasets with images from city surveillance cameras. There are even fewer when project specific needs are considered. This made finding additional datasets quite difficult. For example, one requirement was to have a similar angle of view, which led to the decision to eliminate datasets with images coming from car front view cameras. Another requirement was not to have to do additional manual annotation. This meant that that more or less the same classes and same annotation policy had to be used. One last requirement was that images had similar resolutions. To add to the challenge, a lot of previously publicly available city surveillance camera datasets were terminated in recent years due to privacy concerns [**24**].

It was possible nonetheless to identify three additional datasets: PETS, MIO-TCD and MOT which are described in subsections 3.2.1 to 3.2.3.

### 3.2.1. PETS

The PETS 2009 Benchmard Dataset [**19**] consists in various sequences of crowd activities. It was initially released as part of a crowd analysis challenge for the PETS 2009 workshop, where tasks included counting crowds, tracking individuals and detecting separate flows and crowd events. The video sequences are copyright of University of Reading, and a permission to use images is granted for the purposes of academic and industrial research. Sample images are shown in Figure 3.4.

The following pre-processing steps were performed:

- Images were cropped in order to only include labeled pedestrians, as some persons were unlabeled in images. Manually annotating these would have been too time consuming.
- Two unlabeled cars were visible in all images although always in the same location, thus bounding boxes for these vehicles were added to all annotations.
- Annotations were converted in a Pascal VOC format to match the CGMU format.

**Fig. 3.4.** Sample images from PETS dataset

### 3.2.2. MIO-TCD

The MIOvision Traffic Camera Dataset (MIO-TCD) [**45**] is a large dataset comprising 137,743 annotated images of motorized traffic and includes 11 different traffic objects, including cars, trucks, buses, motorcycles, cyclists and pedestrians. It was originally released as part of an object detection challenge under a *Attribution-NonCommercial-ShareAlike 4.0* license that enables non-commercial use. Sample images are shown in Figure 3.5. One can note that the angle of view is very much similar to that of CGMU cameras. In addition, the image resolutions are also very similar to that of the CGMU dataset, with two different types: 342x228 and 720x480 pixels. This dataset also includes images from colder months and include images with snow. The only drawback of this dataset is that it does not include annotations for construction items.

In order to add this dataset to the training set, the following pre-processing steps were made:

- All images that didn't contain at least one of the under-represented classes (i.e. pedestrians, cyclists, buses) were excluded. The vehicle class was already over-represented.

- All images which included unlabeled construction items (e.g. cones, fences) were manually removed. Manually annotating these would have been too time consuming.
- All motorized vehicle classes except buses (i.e. articulated truck, car, motorcycle, motorized vehicle, pickup truck, single unit truck, work van) were reclassified as "vehicle", per the CGMU definition.
- The "bicycle" class was renamed "cyclist", to match the CGMU class label. Bounding boxes for the "bicycle" class already included the person riding the bicycle, which was fortunate, otherwise it would not have been possible to include these images.
- Annotations were converted in a Pascal VOC format.

Overall, with the MIO-TCD dataset, it was possible to add 2260 cyclists and 8319 buses examples to the training set.

**Fig. 3.5.** Sample images from MIO-TCD dataset

### 3.2.3. MOT Benchmark

The Multi-Object Tracking (MOT) centralized Benchmark was created for evaluating and comparing different multi-target tracking methods for pedestrian detection. Datasets released in 2016 [**36**] [**48**] and 2020 [**14**] were the most amenable to this project and are under a *Attribution-NonCommercial-ShareAlike 3.0* license that enables non-commercial use. Sample images are shown in Figure 3.6.

In order to be able to add this dataset to the training set, the following was performed:

- The "Pedestrian" and "Static person" classes were combined and renamed to "Pedestrian" to match the CGMU dataset.

- Images were cropped in order to only include labeled pedestrians, and exclude unlabeled construction cones. Manually annotating these would have been too time consuming.
- Since MOT datasets are originally designed for tracking, images corresponds to frames from three different video sequences, and all images are very similar. Thus, smaller dataset samples were generated using a sampling strategy of 1% (1 every 100 frames) and 10% of all images.
- Annotations were converted in a Pascal VOC format.

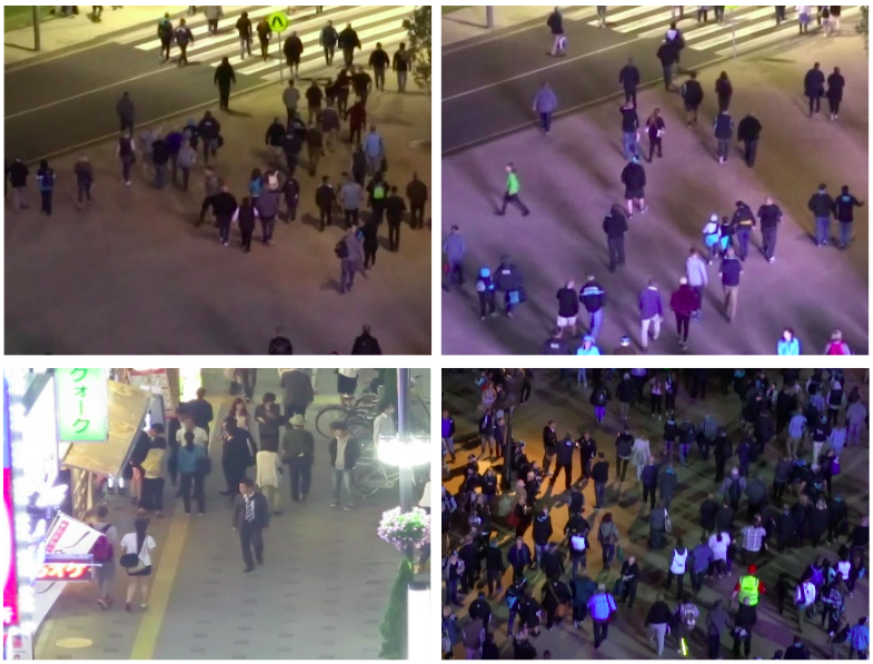


**Fig. 3.6.** Sample images from MOT datasets

### 3.2.4. Combined Datasets

Table 3.3 details the number of objects from each class in the newly created datasets. One should note that only the MIO-TCD dataset includes cyclists and buses. Figure 3.7 shows the class distribution when the PETS, MIO-TCD and MOT are added to the CGMU dataset. There is still an important unbalance between the classes, but it is less severe. Various combinations of these additional datasets with the CGMU dataset were tested and results are presented in Section 5 - Results and Discussion.

| Datasets extra | vehicle | pedestrian | construction | cyclist | bus | Total images |
|---|---|---|---|---|---|---|
| PETS | 0 | 4623 | 0 | 0 | 0 | 795 |
| MIO-TCD (all) | 40067 | 7084 | 0 | 2260 | 8319 | 11877 |
| MIO-TCD (no vehicle) | 0 | 595 | 0 | 768 | 2424 | 2856 |
| MOT20-05 (10%) | 0 | 44534 | 0 | 0 | 0 | 328 |
| MOT20-05 (1%) | 0 | 4401 | 0 | 0 | 0 | 33 |
| MOT20-03 (10%) | 0 | 19446 | 0 | 0 | 0 | 237 |
| MOT20-03 (1%) | 0 | 2026 | 0 | 0 | 0 | 24 |
| MOT16-04 (10%) | 0 | 2808 | 0 | 0 | 0 | 102 |
| MOT16-04 (1%) | 0 | 268 | 0 | 0 | 0 | 10 |

**Table 3.3.** Number of images and objects in datasets.

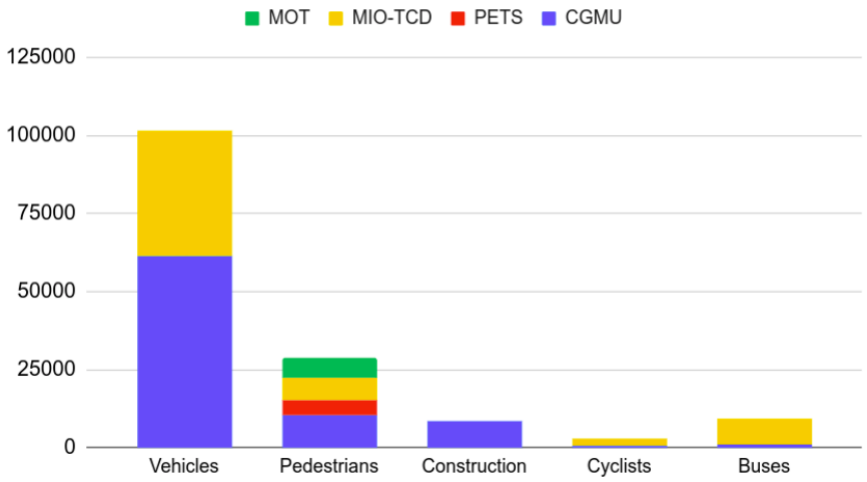



**Fig. 3.7.** Class distribution with and without additional datasets: MIO-TCD (all), MOT20-05(10%), MOT20-03(10%) and MOT16-04(10%)

# Chapter 4

# Methodology

The original intent was for three different model solutions to be implemented and tested for this project: **1)** a Single-Shot Multibox Detector (SSD), which was chosen as a baseline, **2)** YOLOv5 and **3)** EfficientDet. Unfortunately, it was not possible to train an EfficientDet model on the CGMU dataset. Many attempts were made to find a successful training configuration but none achieved a somewhat smooth and decreasing training loss. At the time, the model was still under active development and several other users were experiencing and reporting similar issues on the Github repository [**4**] and it was decided that this model evaluation would be put on hold for until this issue was resolved by the authors. After obtaining promising results on YOLOv5, it was decided that time was better invested pursuing further performance improvements with this model rather than working on troubleshooting and resolving the issues experienced with EfficientDet. For this reason, the methodology pertaining to EfficientDet will only be discussed minimally in the following sections. A greater focus is given to the baseline model and YOLOv5 to compensate.

## 4.1. Single-Shot Multibox Detector

The Single Shot Multibox Detector (SSD) [**44**] was one of the earliest one-stage detectors and has been used extensively since its creation in 2015. At that time, it was one of the fastest model available, making it very well-suited for real world and real-time applications. Today, it is no longer a state-of-the-art model but its simplicity makes it a good baseline model. Its architecture is illustrated in Figure 4.1.

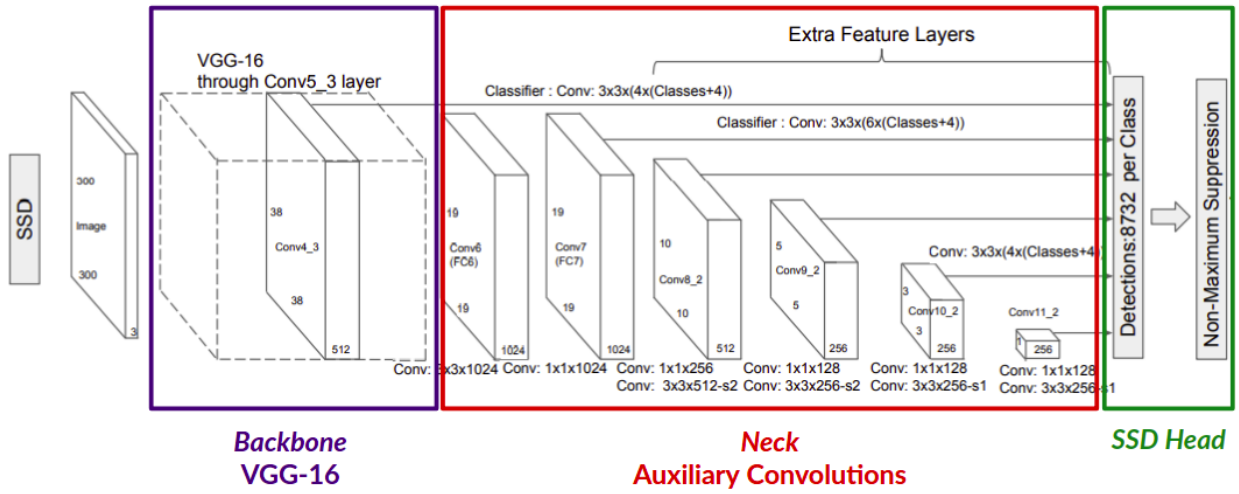



**Fig. 4.1.** Architecture of the SSD network. Adapted from [**44**]

### 4.1.1. Pre-Processing

The implementation chosen for this project [**1**] requires that images and labels are in the Pascal VOC format, which is conveniently the format that was used for the annotation campaign. Thus no further pre-processing step was required for the baseline model.

### 4.1.2. Data Augmentations

It was shown that an SSD model is far more robust if data augmentation is used during training [**44**]. Authors have suggested randomly sampling augmentations and amplitudes every time a training image propagated through the network. From epoch to epoch, the same image is assigned a new combination of random augmentations. The pool of augmentations used is shown in Figure 4.2.

### 4.1.3. Model

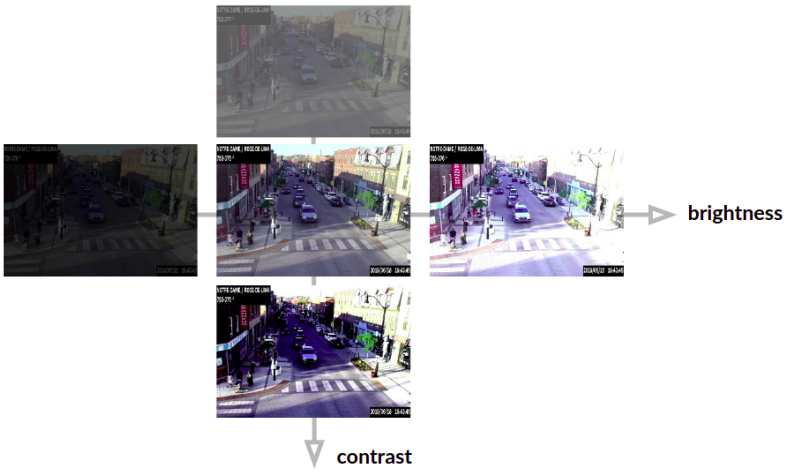
A Pytorch implementation of an SSD300 [**1**] found on Github was used as a starting point for this baseline model. The original SSD model was released in two variants, SSD300 and SSD512, for images of input size 300x300 and 512x512 respectively. The former was selected given its faster inference speed. Note that the resizing of input images is done via the dataloader, thus resizing images prior to training is not necessary. Original image sizes can be used as input to the model.

**Backbone:** The VGG-16 architecture was used as the base network and pre-trained weights from an ImageNet classification task were used.

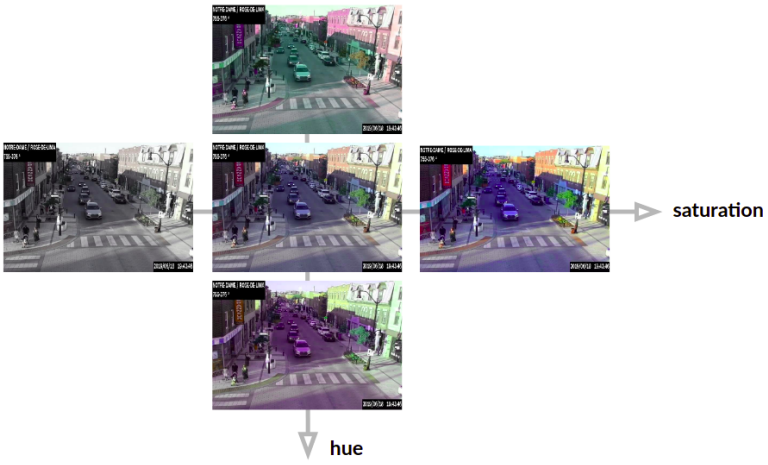
**Neck:** The SSD adds auxiliary convolutions on top of the VGG-16 backbone to extract feature maps at different scales and receptive fields. The outputs from convolutions $4_3$, 7, $8_2$, $9_2$, $10_2$ and $11_2$ are each progressively smaller than the last. Every positions in these feature maps is assigned multiple priors (or anchors). These are predefined shapes of bounding box that are superimposed on all cells of specific feature maps and used as approximate bounding box candidates. They are transmitted to the head for the regression of bounding boxes and classification. A total of 8732 priors are being extracted in this fashion. The default SSD priors scale and aspects ratios were used.

**Head:** Each prior (or anchor) is assigned to the head and used as an approximate starting point. The regression tasks consists in predicting deviations from the prior in the form of offsets (e.g. translation, contraction, expansion). Hence, a total of 8732 predicted boxes and 8732 sets of class scores are generated by the model.

**Loss Function:** The SSD loss function is derived from the MultiBox loss [**57**], which is a weighted sum of two components: a confidence loss corresponding to the sum of cross-entropy losses for positive and negative examples, and a localization loss corresponding to average Smooth L1 distance between the ground truth boxes and the positively matched offsets. Since the the ratio of background to objects in CGMU images is far greater than one, this introduces an imbalance between negative examples (i.e. no object) and




(a) Photometric Augmentation




(b) Photometric Augmentation (cont'd)


(c) Geometric Augmentation

**Fig. 4.2.** SSD augmentations.

positive examples. To address this, hard negative mining is used to sort and select negative examples with the highest confidence loss (i.e. hard examples). This helps reducing the ratio of negative examples to at most 3:1.

#### 4.1.4. Post-Processing

Non-Maximum Suppression (NMS) is used to filter out the number of boxes generated to make sure that the model detects each object only once.

### 4.2. YOLOv5

YOLOv5 was first published in May 2020 as a Pytorch implementation on Github [**33**] and is still under active development with improvements being made on a weekly basis. For this reason, a fixed commit (**bb8872e**) was forked at the latest possible stage of the project, and then used for project specific development. Using a newer commit may have beneficial effects in regards to performance but commit **bb8872e** should be used if one requires a stable version.

At the moment of writing this report, a scientific paper from YOLOv5 authors is still pending, and as a result some elements of the technology may be unintentionally missed in this report. The key aspects of this model are described in the following subsections.

#### 4.2.1. Pre-Processing

YOLOv5 requires ground truth data to be converted to a specific annotation format. Thus, a python script was created to convert the annotation format from PASCAL VOC to this YOLO specific file format.

All versions of the YOLO family models utilize anchor boxes to make bounding box predictions. Thousands of candidate anchor boxes around the image are being evaluated by the model, and their shapes and aspect ratios are based on priors knowledge, i.e. a pre-defined setting. It is beneficial to use anchor boxes that are well representing the dataset being used. By default, YOLOv5 is configured with anchor boxes that are optimal for the COCO detection dataset. However, a notable feature of YOLOv5 is its auto-anchor pre-processing step, which uses a k-means algorithm to learn anchor boxes (i.e. shape only) based on the ground truth bounding box distribution in the custom dataset. This step is particularly important for the CGMU dataset, which has construction cones and pedestrians that are tall and skinny, and cars and buses that are wide and flat.

#### 4.2.2. Data Augmentations

YOLOv5 applies a combination of photometric (e.g. brightness, hue, saturation) and geometric distortions (e.g. translation, scaling, cropping, rotating, flipping) to each training batch prior to forward propagation. At every epoch, the mix of distortions used and their amplitudes are randomly selected. In other words, images are slightly different at every epochs. This has the effect of ariticially generating a larger training dataset and improving robustness and generalization of the trained model.

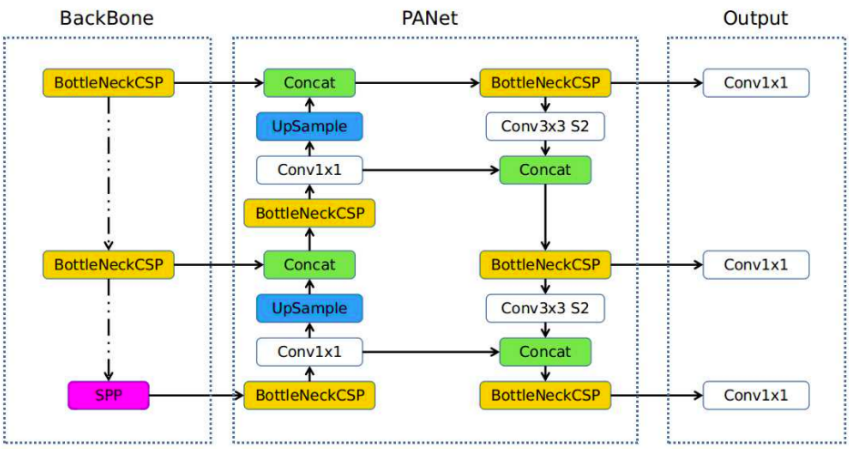



**Fig. 4.3.** General Architecture of YOLOv5. Source: [**33**] **- issue 280**

YOLOv5 also uses a certain level of MixUp (see Figure 2.4), in combination with other augmentation techniques. This method is equivalent to multiplying and superimposing the pixel values from two different images, and adjusting the labels accordingly with the coefficients used.

Another notable feature of YOLOv5 is the integration of mosaic augmentations (see Figure 2.4), which consists in combining four images into a new image using random size ratios. Hence, this enables simulating detection of objects outside of their normal context, and at smaller scales than normal. On the COCO object detection benchmark, YOLOv5 made large performance improvements especially on small objects thanks to this augmentation technique. Given that the mosaic augmentation combines multiple images together, another benefit is to reduce the need for using large mini-batch sizes.

Augmentation techniques are only applied at training time and thus do not affect inference speed.

### 4.2.3. Model

YOLOv5 is in fact a suite of four different model architectures: Small (S), Medium (M), Large (L) and Extra Large (X). The smallest has 7.5 million parameters and 140 layers, while the largest has 89 million parameters and 284 layers. In this study, all four architectures were tested. With YOLOv5, one has the option to train from scratch or use pre-trained checkpoints. This project adopted the transfer learning approach, and trained from models that were pre-trained on the MS COCO detection dataset [**42**]. The general YOLOv5 model architecture is illustrated in Figure 4.3. This general architecture is shared among all 4 model variants, only the width and depth are changing.

**Backbone:** YOLOv5 uses a Cross-Stage Partial Network (CPSNet) [**61**] as its backbone to generate feature maps. Figure 4.4 shows illustrations of a CSPNet layer and its predecessor, the DenseNet [**29**] from which it is derived. The DenseNet had been a very successful way to deal with vanishing gradients in very deep convolutional neural networks. The inventors of the CSPNet discovered that by separating the feature map from previous DenseNet stages into two parts, and feeding one through a series of dense and transition

blocks and combining the other part with the feature map transmitted to the next stage, it is possible to remove computational bottlenecks and reduce memory costs while improving the network learning capability. This new backbone is one key contributor to giving YOLOv5 state-of-the-art performance at higher inference speed and smaller model size.

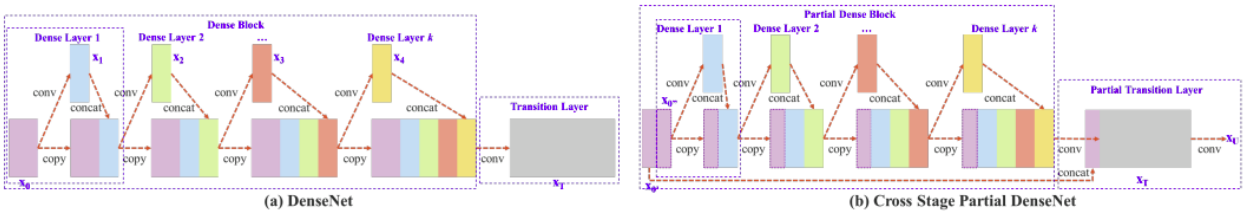



**Fig. 4.4.** Comparison between a) DenseNet and b) Cross-Stage Partial Net. Source: [**42**]

**Neck:** Figure 4.5 shows several neck architectures that were developed in recent years and that have been associated with good performance gains. All four variants were tested as part of the development of the YOLOv4 variant and it is the PA-Net architectures that were found to give the best results [**9**]. This research inquiry led to the adoption of PA-Net blocks for parameter aggregation in YOLOv5. Compared to Feature Pyramid Networks (FPN) which were used in YOLOv3, PA-Net introduces a top-down pathway to fuse feature layers from various scales.

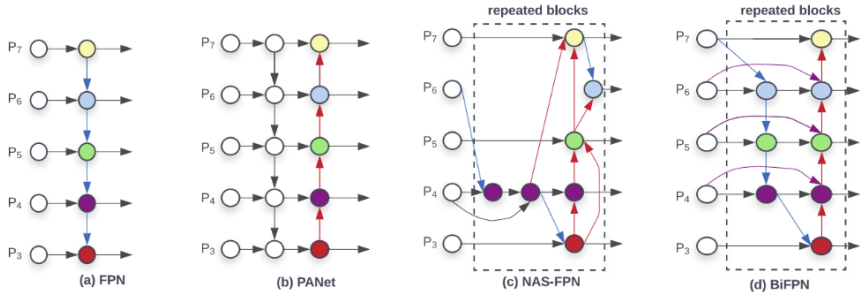



**Fig. 4.5.** Comparison of various neck designs Net. Source: [**60**]

**Head:** YOLOv5's detection head is the same anchor based head as YOLO v2's [**51**], v3's [**52**] and v4's [**9**]. Figure 4.6 illustrates an example of a bounding box, with the dimension priors (dotted line) for one anchor and the predicted location detection (blue rectangle). For each bounding box, the head generates 5 output values: $t_x$, $t_y$, $t_w$, $t_h$ and $t_o$. The coordinates for the center of the bounding box are obtained with:

$$b_x = \sigma(t_x) + c_x$$

$$b_y = \sigma(t_y) + c_y$$

where $(c_x, c_y)$ is the coordinate of the anchor cell center. The bounding box width and height is predicted as an offset from the bounding box prior $p_w$, $p_h$, as follows:

$$b_w = p_w e^{t_w}$$

$$b_h = p_h e^{t_h}$$

Finally, a measure of how much the predicted box matches the ground truth box is obtained with:

$$Pr(\text{object} * IoU(\text{b}, \text{object}) = \sigma(t_o)$$

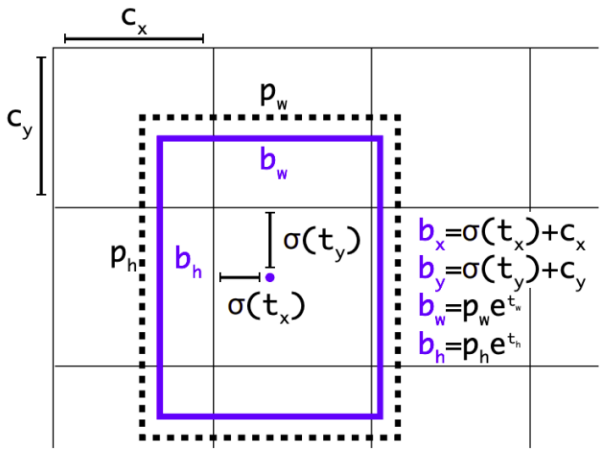



**Fig. 4.6.** Example of bounding box in YOLO's frame of work, with dimension priors shown as the dotted line and the location detection shown as the blue rectangle. Source: [**51**]

**Activation functions:** YOLOv5 uses Leaky ReLU activations [**47**] after each CSP block and Hard Swish activations [**28**] after each convolution. Leaky ReLU activations enable some amount of information to flow even when the input signal is negative. Hard Swish activations have the accuracy benefit of Swish activations, but are much faster to compute since the sigmoid function is replaced by a piece-wise linear analog.

$$LeakyReLU(x) = \begin{cases} x, & \text{if } x \geq 0 \\ 0.01x, & \text{otherwise} \end{cases}$$

$$HardSwish(x) = \begin{cases} 0, & \text{if } x \leq -3 \\ x, & \text{if } x \geq 3 \\ \frac{x(x+3)}{6}, & \text{otherwise} \end{cases}$$

**Loss Function:** YOLOv5 computes a total loss which combines three different loss components - 1) a regression loss for the bounding box coordinates, 2) an objectness loss and 3) a classification loss. The regression loss used is the Complete-IoU loss [**73**], which is based on the definition of IoU but includes

penalty terms that allows to simultaneously consider the overlap area, the central point distance and the aspect ratio. This type of loss performs better in non-overlapping cases and was shown to achieve more accurate box detection compared to alternative losses like Generalized-IoU and Distance-DIoU [**73**]. The objectness loss is there to punish the network if it predicts an object in a cell where there is no object and the classification loss is self-explanatory. Both objectness and classification losses are based on binary cross-entropy with logits. The classification loss also integrates class label smoothing for regularization [**71**]. Note that as an alternative, it is possible to use a focal loss [**40**] for objectness and classification, which is what is being used in YOLOv4, however this option was not tested in this project.

**Optimization Function:** Both Adam and Stochastic Gradient Descent (SGD) are available optimizer options in YOLOv5's off-the-shelf implementation. Both types of optimizers were tested.

### 4.2.4. Post-Processing

To address multiple detections overlapping for one same object, YOLOv5 uses Distance-IoU Non-Maximum Suppression (NMS).

## 4.3. EfficientDet

The EfficientDet neural newtork is an object detection model that was created by the Google Brain team [**60**] [**4**]. It uses the EfficientNet classification architecture [**59**] as its backbone. To aggregate feature representations of the input image at different scales, a bidirectional feature pyramid network is used (see Figure 4.5). These are fed to a class prediction head and a box prediction head. Like YOLOv5, EfficientDet consists in a family of architecture sizes, with nine networks ranging from 3.9 to 77.0 million parameters. A compound scaling method is used to uniformly scale the networks resolution, depth and width and this feature enables an easier selection in accordance with the hardware being used for training or inference. The EfficientDet architecture is shown in Figure 4.7.

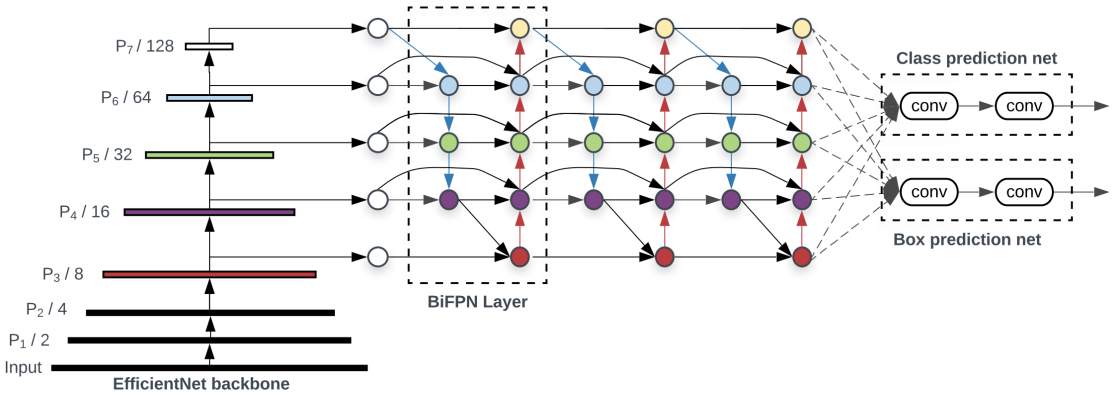



**Fig. 4.7.** EfficientDet neural network architecture. Source: [**51**]

## 4.4. Hyperparameter Search

To achieve the best possible object detection mAP for a fixed model architecture, a careful tuning of hyperparameters pertaining to the model is often required. Training the heaviest YOLOv5 model takes around 22 minutes per epoch, which means fully training a model for 300 epochs takes just under five days. A grid search or random search strategy would be extremely costly in computing resources and/or would simply take too long. With the project constraint of having only one GPU to train models, exploiting parallelism was not an option, even though it is becoming the new paradigm for hyperparameter optimization [**38**]. This prompted a search for a quick and efficient solution.

The Orion framework [**11**] was used to automate the search for a good hyperparameter configuration. Orion is an open-source asynchronous framework for optimization that enables treating the training script as a black box, which means only minimal alteration to the model training pipeline is necessary. This made it a flexible and convenient optimization framework for this project and would be easily adaptable to future machine learning projects.

Within Orion, the algorithm chosen for exploring hyperparameter configurations and optimizing detection performance is the Asynchronous Successive Halving Algorithm (ASHA) [**38**]. ASHA is the asynchronous version of Hyperband [**39**]. The main idea of these two algorithms consist in performing early stopping of poor candidate trials based on the available information of the trials executed so far. For this, a strategy similar to Successive Halving [**31**] is employed.

To illustrate successive halving, let's consider an example where a total of 32 configurations must be explored. And let's assume the algorithm uses an elimination rate $\eta = 2$ and a minimum resource budget of 1 epoch. The successive steps are shown in Table 4.1. All trials are first trained for one epoch. Then, all candidates are evaluated and only the best $1/\eta$ trials are promoted to the next step (called rung), where each configuration now receives $\eta$ times the computing budget. Thus, the best 16 trials are promoted and trained for 2 epochs. Then, the best 8 trials are promoted to the next rung of 4 epochs and so on. In the end, only the best trial is trained for the full 32 epochs. Instead of fully training 32 configurations for 32 epochs (total: 1024 epochs), the total computing budget is only 192 epochs.

| | **Configurations Remaining** | **Epochs per Configuration** |
|---|---|---|
| Rung 1 | 32 | 1 |
| Rung 2 | 16 | 2 |
| Rung 3 | 8 | 4 |
| Rung 4 | 4 | 8 |
| Rung 5 | 2 | 16 |
| Rung 6 | 1 | 32 |

**Table 4.1.** Example of Successive Halving with 32 configurations, $\eta = 2$ and a minimum resource of 1 epoch in the first rung

The Hyperband algorithm waits for all trials of a given rung to be finished before proceeding with the next rung (i.e. synchronous promotion). ASHA uses a modified promotion rule that leverages partial information from a rung to sequentially promote trials to the next run. This enables a better workers utilisation when multiple workers are available. It was also found to be superior to Hyperband in single-worker settings [**38**]. In summary, ASHA enables a more efficient evaluation of hyperparameter configurations for a given computing resource budget.

The ASHA settings used for this project are as follows:

- A minimum budget of 2 epochs per configuration,
- A target budget of 256 epochs for the best configuration,
- An elimination rate, $\eta = 2$,
- A Bracket of 2, which means the algorithm is repeated with a minimum budget of 4 epochs per configuration.

Experimenting with SSD, EfficientDet and YOLOv5 revealed that the latter was the most promising model for this project. Thus YOLOv5 was the only model for which an hyperparameter search was conducted. Table 4.2 shows the list of parameters that were explored, as well as the range and sampling distribution used during the ASHA exploration. More details on this procedure are given in Chapter 5.

| **Hyperparameter** | **Range** | **Distribution** |
|---|---|---|
| lr0 | 0.001 - 0.1 | log uniform |
| lrf | 0.05 - 0.5 | uniform |
| momentum | 0.75-0.98 | uniform |
| weight decay | 0.0002-0.0007 | uniform |
| giou | 0.02-0.2 | uniform |
| cls | 0.2-2 | uniform |
| cls_pw | 0.5-2 | uniform |
| obj | 0.2-2 | uniform |
| obj_pw | 0.7-2 | uniform |
| iou_t | 0.1-0.3 | uniform |

| **Hyperparameter** | **Range** | **Distribution** |
|---|---|---|
| anchor_t | 2-6 | uniform |
| hsv_h | 0-0.1 | uniform |
| hsv_s | 0-0.9 | uniform |
| hsv_v | 0-0.6 | uniform |
| degrees | 0-5 | uniform |
| translate | 0-0.5 | uniform |
| scale | 0-1 | uniform |
| shear | 0-1 | uniform |
| mixup | 0-1 | uniform |

**Table 4.2.** Hyperparameter exploration configuration in Orion

## 4.5. Computational Resources

All experiments and training were conducted using notebooks on Google Colab Pro, with the exception of hyperparameter search which was performed using a GPU instance on a Google Cloud Platform.

For 10$ USD per month, Google Colab Pro has a time limitation of 24 hours and idle timeouts that are relatively lenient compared to the free version. It also does not limit the amount of experiments that can be ran on any given week, which was an important limitation for this project. The Pro version has a hard drive limit of 150 GB, 26 GB of ram, and a GPU memory of 16 GB. The GPU resource that is allocated is either a Tesla P100 or V100 from NVIDIA. Since some of the heavier models take more than 24 hours to

train, the key was to ensure that experiments are resumable and most importantly, that they are reproducible.

Once the best model configuration was selected, a rigorous hyperparameter search was conducted for a period of 30 days, which made Google Colab Pro less convenient due to the 24 hour time limit. Therefore, an instance on the Google Cloud Platform was created, with a permanent SSD hard disk of 1 TB, 26 GB of ram, and a Tesla P100 NVDIA GPU card.

## 4.6. Specification of Dependencies

The following is a list of packages used for this project, along with their versions at the time of conducting experiments:

- Python - 3.6.9 (Colab) or 3.8.5 (Google Cloud Platform)
- Pytorch - 1.6.0
- torchvision - 0.7.0
- Matplotlib - 3.3.2
- Numpy - 1.19.1
- Orion - 0.1.8
- opencv - 4.4.0
- pillow - 7.2.0
- PyYAML - 5.3.1

## 4.7. Reproducibility

To make experiments perfectly reproducible during the hyperparameter search, a new flag was created, `--orion`, and added to YOLOv5's training script. This achieves the following:

- Setting `torch.manual_seed(`*`seed`*`)`
- Setting `torch.cuda.manual_seed(`*`seed`*`)`
- Setting `torch.cuda.manual_seed_all(`*`seed`*`)`
- Setting `numpy.random.seed(`*`seed`*`)`
- Setting `os.environ['PYTHONHASHSEED'] = str(`*`seed`*`)`
- Setting `cudnn.deterministic = True`
- Setting `cudnn.benchmark = False`
- Update scheduler function for fixed training duration of 300 epochs
- Disable auto-anchor
- Loading/saving scheduler state dictionary
- Loading/saving optimizer state dictionary
- Loading/saving model state dictionary
- Loading/saving torch random number generator state
- Loading/saving numpy random number generator state
- Loading/saving python random number generator state

In addition to the above, the flag `--noautoanchor` should be used, otherwise new anchors may be generated in between rungs, which will affect reproducibility.

# Chapter 5

## Results and Discussion

Table 5.1 shows how the general size of the neural network affects performance when trained on only the CGMU dataset for a fixed input image size of 320x320 and while using default YOLOv5 hyperparameters (i.e. optimal on COCO). The baseline SSD model, with its 26.7 M parameters, achieves an mAP of 0.466 while the 'M' model achieves an mAP of 0.663 with roughly the same model size. The YOLOv5 'M' model is not only more accurate, its inference speed is much faster and its training time is much shorter. The largest YOLOv5 model, the X with 89.0 M parameters, achieves an mAP of 0.696. Model size also affects training time. Not only does the training time per epoch increase with size, but larger models are found to generally require more epochs to achieve their best performance. Larger models also have higher latency at inference time. For example, the smallest YOLOv5 architecture runs in 86.1 ms on a CPU and 10.4 ms on a GPU, with a batch size of 1. The largest architecture runs in 629.9 ms and 29.2 ms on CPU and GPU respectively. Note that latency tests were achieved on a Colab Pro instance, which was running with an NVIDIA Tesla P100 GPU card. To improve latency further, a better strategy might be to collect images from multiple cameras and treat them at once in batches. For example, the latency on the small YOLOv5 architecture would go from 86.1 ms to 55.5 ms on CPU, and from 10.4 ms to 2.4 ms on GPU, when increasing the batch size from 1 to 32. The benefit is larger for GPU inference.

In the early stages of experiments, both Adam and Stochastic Gradient Descent (SGD) optimizers were tested but SGD was found to be far superior with the default hyperparameter settings. Thus, SGD was used for all subsequent experiments.

| | **Parameters** (M) | **mAP** @ 0.5 IoU | **Latency** batch size 1 CPU \| GPU | **Latency** batch size 32 CPU \| GPU | **Train Time** (minutes per epoch) |
|---|---|---|---|---|---|
| Baseline-SSD | 26.3 | 0.466 | *n/a* \| *n/a* | *n/a* \| 21.7 ms | 3.31 |
| YOLOv5-S | 7.5 | 0.617 | 86.1 \| 10.4 ms | 55.5 \| 2.4 ms | 0.46 |
| YOLOv5-M | 21.8 | 0.663 | 187.5 \| 14.1 ms | 128.9 \| 3.7 ms | 0.56 |
| YOLOv5-L | 47.8 | 0.685 | 351.3 \| 22.0 ms | 259.1 \| 5.7 ms | 0.76 |
| YOLOv5-X | 89.0 | 0.696 | 629.9 \| 29.2 ms | 489.4 \| 9.8 ms | 1.13 |

**Table 5.1.** Influence of model size on training time, performance and latency. SSD latency measured with a Titan X GPU card and a batch of 8. Input image size: 320x320

Another important configuration setting that was found to influence the performance and latency trade-off is the input image size. For example, the baseline SSD model uses images of size 300x300 as inputs. For

both the SSD and YOLOv5, resizing of the input images is done via the data loader, thus pre-processing the images in the dataset is not necessary. Another aspect to be aware of is the fact that input image size does not change the architecture width or depth due to the convolutional nature of the network, but it increases the size of feature maps and number of computations. Table 5.2 shows that large incremental gains in performance can be achieved by increasing the input image size from 320 to 704. CGMU large image resolutions are 704x480, thus with an input image size of 704x704, no pixel information is lost. This is especially important for small objects, like pedestrians and construction cones, which are very narrow. Another interesting finding is that while the latency increases with input image size on a GPU, latency is found to slightly decrease with CPU inference.

| | **mAP** @ 0.5 IoU | **Latency** CPU \| GPU | **Training Time** (min/epoch) |
|---|---|---|---|
| Baseline-SSD | 0.466 | *n/a* \| 21.7 | 3.31 |
| YOLOv5-X-320 | 0.696 | 629.9 \| 9.8 ms | 1.13 |
| YOLOv5-X-512 | 0.804 | 584.3 \| 17.1 ms | 2.24 |
| YOLOv5-X-704 | 0.845 | 577.1 \| 31.0 ms | 13.89 |

**Table 5.2.** Influence of input image size on training time, performance and latency. Training using CGMU dataset only. For YOLOv5, batch sizes of 1 and 32 are used for CPU and GPU inference respectively. SSD latency measured with Titan X GPU card and a batch size of 8.

As shown in Table 5.3, including the extra datasets during training was found to help achieve better overall performance on the YOLOv5 model in all permutations of dataset combinations that were tested. Specifically, an improvement is seen for all classes except "construction" objects, which was expected as none were added via these new datasets. The fact that the performance decreases for this class may not be too much of a concern however considering that construction items are always static in video frames, thus it may not be as important to detect these objects on every frames. The biggest performance improvement is seen on the "bus" class. This also makes sense since this is the class that saw the biggest relative increase of objects seen during training, with roughly a tenfold increase. Adding extra datasets comes at the expense of additional training time. Just adding the MIO-TCD dataset which is larger than the CGMU dataset, results in training times that are nearly 4 times as long. A surprising discovery was the fact that adding the MIO-TCD dataset for training the SSD model was not found to improve overall performance. It is still unclear at this point why this was the case. But a reasonable explanation may be that the SSD model complexity is not sufficient to adequately generalize the learning contribution of non-CGMU types of objects to the CGMU domain.

Readers with a keen eye will notice that the mAP of model YOLOv5-X-320 in Table 5.2 differs from that in Table 5.3. This is because results in Table 5.3 are associated to an earlier YOLOv5 repository commit version. Some important changes were made to the repository on July 22nd and August 13th and they led to significant performance improvement. Commit **bb8872e** is the one which was forked and corresponds to a snapshot that was taken on September 8th 2020. Table 5.2 and all subsequent Tables with the exception of Table 5.3 were generated with this specific commit version. Results from Table 5.3 are still valuable in the sense that they demonstrate how adding additional datasets have a positive effect on the YOLOv5

| | **Extra Datas** | **mAP** 0.5 IoU | **vehicle** mAP | **pedestrian** mAP | **construction** mAP | **cyclist** mAP | **bus** mAP | **Training** min/epoch |
|---|---|---|---|---|---|---|---|---|
| SSD | - | 0.466 | 0.601 | 0.340 | 0.290 | 0.471 | 0.621 | 3.31 |
| SSD | MIO-TCD | 0.433 | 0.604 | 0.353 | 0.276 | 0.457 | 0.631 | 9.62 |
| YOLOv5 X-320 | - | 0.638 | 0.793 | 0.606 | 0.576 | 0.629 | 0.588 | 1.13 |
| YOLOv5 X-320 | MIO-TCD | 0.664 | 0.815 | 0.62 | 0.502 | 0.643 | 0.739 | 4.16 |
| YOLOv5 X-320 | MIO-TCD + PETS + MOT(1%) | 0.666 | 0.815 | 0.629 | 0.511 | 0.648 | 0.728 | 4.37 |
| YOLOv5 X-320 | MIO-TCD + PETS + MOT(10%) | 0.665 | 0.812 | 0.611 | 0.492 | 0.661 | 0.751 | 4.49 |
| YOLOv5 X-320 | MIO-TCD + MOT(1%) | 0.665 | 0.813 | 0.617 | 0.517 | 0.638 | 0.741 | 4.18 |

**Table 5.3.** Influence of adding extra datasets on performance and training time

model performance.

The constraint of achieving real time inference is very much different whether a GPU is available for inference or not. For a first prototype, it may not be possible to use a GPU, but for the subsequent stages it is likely that the solution will be deployed over multiple cameras, thus using GPU will very quickly become necessary. Hence, it was decided that two different models would be selected for the hyperparameter optimization: one for real-time inference on GPU and another for inference on CPU. For inference on GPU, it was decided that an Extra Large 'X' architecture with an input image size of 512x512 was a good compromise for achieving good performance on both small and large image resolutions and would be trainable in reasonable time. While an input image size of 704x704 would have increased model performance, it would also increase training times drastically, which would significantly reduce the number of configurations that can be explored with a fixed computation budget. As an example fully training the 'X' architecture for 300 epochs with an input image size of 704x704 takes nearly 3 days on the CGMU dataset and nearly 11 days on the combined CGMU and MIO-TCD datasets. For CPU inference, the Small 'S' YOLOv5 architecture was chosen, also with an input image size of 512x512. The latency for such model is close to 167 ms on CPU, which is equivalent to $\sim$ 5 FPS. For both models, the training dataset was supplemented with the MIO-TCD dataset. The MOT and PETS datasets were not included because the incremental gain in performance was not sufficient to justify the increase in training time, which would cascade into longer hyperparameter search times.

ASHA was used for conducting the hyperparameter search for both models. Hyperparameters chosen for the exploration as well as the range of exploration and sampling distributions are listed in section 4.4. While conducting the exploration on the 'X' model, it became apparent that achieving near perfect reproducibility was essential to the success of ASHA as model training is interrupted and resumed after each rung. Initially, reproducibility was not guaranteed and this led to dips in the model training and validation performance every time model training was resumed. This behavior only became visible after 32-64 epochs, when the performance usually starts slowly plateauing. At this moment, the exploration was already half way done, and 10 days of computing power had already been consumed. A decision had to made on whether to interrupt and restart the experiment or to let it continue. Initially, the belief was that

this behaviour would not prevent finding an optimal configuration, as all configurations would be affected in a similar manner and only the best ones would be promoted to the end. In retrospect, some configurations were likely affected more than others. However, having to stop, then fix and test reproducibility and restart the exploration from the beginning might have compromised the exploration on the smaller model due to lack of time or insufficient computing resource budget and for this reason, it was probably the right decision to make. To help mitigate the risk of not finding an improved configuration, the exploration was stopped when 8 trials had been promoted to 64 epochs. Then these 8 trials were fully trained to 300 epochs from zero, and the best configuration was selected based on its mAP on the validation set.

While the exploration on the 'X' model was progressing, a large effort to make YOLOv5 models training reproducible was undertaken. More details specific to this effort are provided in Section 4.7. As discussed above, these changes were not incorporated in the ASHA exploration of the 'X' model as it would have meant having to restart the exploration from scratch. However, they were incorporated in time for the exploration of the 'S' model.

Model performance on the validation set prior and after the hyperparameter search is shown in Table 5.4. For the 'S' variant, the default hyperparameter configuration from YOLOv5 is found to be superior to the best trials configurations found with ASHA. For the 'X' variant, the best trial configuration from the ASHA exploration is slightly better compared to the default configuration. Overall, the hyperparameter search was not successful with finding configurations that significantly outperform the default one. The default YOLOv5 hyperparameter configuration is one that was optimized by the YOLOv5 authors for the MS COCO detection challenge using several hundreds generations of hyperparameter evolution using a genetic algorithm. This configuration is already extremely effective for training a detection model on MS COCO, thus it is not too surprising if it performs very well on the CGMU detection task. It is also a useful finding since in the future, it may be more straightforward and cost effective to simply use the off-the-shelf training hyperparameter configuration from YOLOv5 with good confidence that it will yield nearly optimal results. In retrospective, the inferior exploration results may also be linked to the fact that the exploration space was too big for the number of trials. A total of 19 different hyperparameters were explored and only 277 and 206 trials were tested for the 'S' and 'X' architecture sizes respectively. The number of trials that were tested was constrained by the computational resource budget that was available. Here are ways to make better use of the exploration budget in the future:

- YOLOv5 uses the same default hyperparameter configuration for all model variants (i.e. S, M, L and X) and yet achieves very good performance for all of them. Instead of conducting the exploration twice on two different model sizes (S and X), it would probably make more sense to conduct only one exploration using an in-between architecture size like 'M' or 'L' and use the same configuration for both models. The same ASHA exploration configuration took around 5 days on the 'S' and nearly 30 days on the 'X'. Thus, important time savings could be achieved by using an in between architecture size. This time saving could be reinvested into exploring more trials, and hence better covering the hyperparameter space.

- Training on the CGMU dataset alone is 3-4x faster than training a model on the combined CGMU + MIO-TCD datasets. Thus, it may be advisable to conduct the hyperparameter search using only the CGMU dataset, and then fully train a model on both datasets using the best trial configuration. These time savings could be reinvested into exploring more trials.
- It may be preferable to reduce the number of hyperparameters to explore, or reduce the range of exploration. It may also be preferable to use a normal sampling distribution centered around the default hyperparameter setting instead of using a uniform sampling distribution.
- Finally, ASHA will naturally reward configurations that achieve high validation mAP in fewer epochs. However, a configuration that is a straggler compared to other configurations may still achieve a higher maximum validation mAP if it is allowed to train for long enough. It is in fact what was observed when the winning trial from ASHA and the experiment running with the default hyperparameter settings were compared. To address this problem, a higher minimum budget should be used (e.g. 4 instead of 2) and/or a higher number of brackets (i.e. 3 instead of 2).

| | **mAP** | **vehicle** | **pedestrian** | **construction** | **cyclist** | **bus** | **Latency CPU \| GPU** |
|---|---|---|---|---|---|---|---|
| | 0.5 IoU | mAP | mAP | mAP | mAP | mAP | (ms) |
| SSD-300 | 0.466 | 0.604 | 0.353 | 0.276 | 0.457 | 0.631 | *n/a* \| 21.7 |
| YOLOv5-S-512 (before) | **0.738** | **0.879** | **0.700** | 0.594 | **0.719** | **0.798** | 164.2 \| 4.3 ms |
| YOLOv5-S-512 (after) | 0.732 | **0.879** | 0.695 | **0.611** | 0.689 | 0.786 | 167.4 \| 4.1 ms |
| YOLOv5-X-512 (before) | 0.807 | 0.917 | **0.782** | 0.703 | **0.779** | **0.854** | 1398.0 \| 16.6 ms |
| YOLOv5-X-512 (after) | **0.809** | **0.919** | 0.780 | **0.725** | 0.773 | 0.846 | 1268.0 \| 16.6 ms |

**Table 5.4.** Best model performance on validation set before and after hyperparameter search. SSD: Titan X GPU and batch size of 8 used for measuring latency. YOLOv5: NVIDIA Tesla P100 GPU used for measuring latency, and batch sizes of 1 and 32 respectively for CPU and GPU inference.

In the end, three different model variants are proposed for inference. The first variant is a YOLOv5-S model with an input image size of 512x512, which is ideal for a prototype running on a CPU, with an inference of 5 FPS. The second variant is a YOLOv5-X model with an input image size of 512x512 which has good inference latency on GPU and could handle up to 12 cameras at 5 FPS. Finally, a third variant is a YOLOv5-X with an input image size of 704x704, which offers increased performance at the cost of speed. It can handle up to 6 cameras at 5 FPS on a GPU. The first two variants were fully trained on CGMU and MIO-TCD datasets and the third variant was trained on CGMU images only.

Their performance on the test sets is displayed in Table 5.5. As a reminder, the in-domain test set contains images that were never seen before, but at intersections that were included in the training set. One can note that the in-domain test performance is similar to that of the validation set. In contrast, the out-domain test set contains not only images that were never seen before, but also intersections that were

| | **mAP** 0.5 IoU | **mAP** 0.05-0.95 IoU | **vehicle** mAP | **pedestrian** mAP | **construction** mAP | **cyclist** mAP | **bus** mAP | **Latency CPU \| GPU** (ms) |
|---|---|---|---|---|---|---|---|---|
| YOLOv5-S-512 (in-domain) | 0.753 | 0.452 | 0.875 | 0.701 | 0.589 | 0.78 | 0.82 | 167.1 \| 4.2 ms |
| YOLOv5-S-512 (out-domain) | 0.645 | 0.399 | 0.753 | 0.723 | 0.391 | 0.572 | 0.783 | 166.6 \| 4.2 ms |
| YOLOv5-X-512 (in-domain) | 0.820 | 0.534 | 0.915 | 0.788 | 0.719 | 0.829 | 0.849 | 1398.0 \| 16.6 ms |
| YOLOv5-X-512 (out-domain) | 0.692 | 0.456 | 0.835 | 0.779 | 0.419 | 0.596 | 0.829 | 1398.0 \| 17.1 ms |
| YOLOv5-X-704 (in-domain) | 0.861 | 0.574 | 0.939 | 0.828 | 0.809 | 0.838 | 0.892 | *n/a* \| 31.0 ms |
| YOLOv5-X-704 (out-domain) | 0.735 | 0.493 | 0.864 | 0.808 | 0.45 | 0.697 | 0.853 | *n/a* \| 31.0 ms |

**Table 5.5.** Best model performance on in-domain and out-domain test sets.

not included in the training set. The out-domain mAP is lower, which is expected, and is a good measure of the performance that can be expected for all cameras that were not included in the first annotation campaign, or new cameras that will be added on the territory in the future.

Some image samples are displayed in Appendix A for all three best model variants. They show the incremental gain in performance obtained by increasing the model size or input image size. They also demonstrate how each model performs in different day light or night settings, on different resolutions, seasons, or deal with artefacts such as glare, blurriness, occlusion, water droplets and fog.

# Chapter 6

## Way Forward

Here are some ideas (not in order of importance) that would be interesting to investigate and that could help further improving the results:

(1) If 2-3x increase in latency can be afforded, Test-Time Augmentation (TTA) was shown to improve performance significantly on MS COCO [**33**]. This option can be activated via the `--augment` flag in YOLOv5's test script.
(2) Performance gains could also be achieved at the cost of latency by implementing ensembling methods.
(3) The best models already seem to be performing well in winter months. However this may not be well quantified due to the lack of labeled winter images. An additional 1000 CGMU images from winter months could be annotated and added to the training and validation sets. The best model proposed in this report could be used as a pre-annotation tool and help reduce the time needed for the labeling effort.
(4) The 1000 images from the validation set could be combined to the training set for a final training of the best models.
(5) In several months, once important changes are made to the original YOLOv5 repository [**33**], a full retraining of the best model(s) configurations may yield an increase in accuracy and/or a latency reduction. Some updates to the folder hierarchy or training notebooks may be required.
(6) Additional datasets with similar urban scenes and more cyclists, construction objects and buses would also help.
(7) Adjusting and optimizing the YOLOv5 architecture (i.e. the width and depth of specific layers) is not trivial but could be achieved and may also help improve performance.
(8) Self-Training (or Pseudo Labelling) was found to be an effective way to extend available training data [**34**] [**66**]. Using this technique would essentially involve using one of the proposed fully trained model to infer new detections on unlabeled CGMU images, and combine these self-annotated images to the existing training images. The initial training of the model could then be resumed with the extended dataset.
(9) YOLOv5 includes an hyperparameter evolution method that is based on a Genetic Algorithm, and that can be used for optimizing hyperparameters [**33**]. This method may be superior to the one used for this project. This option can be activated via the `--evolve` flag in YOLOv5's train script.
(10) No augmentation technique used in this project specifically tackled blurry images. A solution more robust to blurriness may be obtained by using additional augmentation techniques that simulate this phenomenon in CGMU images during training.

(11) In several weeks or months, when the EfficientDet model is more mature, it may be worth investigating whether it is now possible to train a model on the CGMU dataset.

After having completed this project, the next logical step would now be to start the third and last phase of the roadmap towards anomaly detection (see Figure 1.1) and implement a multi-object tracking solution. A review of the literature would help identify models and techniques that are best adapted for this task. One important requirement for tackling this next project would first to obtain labeled multi-object tracking data from the CGMU. Using the best proposed model (i.e. YOLOv5-X-704) as a pre-annotation tool would be an effective way for reducing the burden on annotators. It would probably be beneficial to also search for additional multi-object tracking datasets. The MOT dataset evaluated in this project [**36**] would be one example of a suitable addition even though it only contains pedestrians.

A very elegant model solution would be one that reuses the fully trained YOLOv5 that was proposed in this project and combine it with a deep network architecture for tracking. One promising solution would be to combine YOLOv5 with a Deep-SORT model [**64**]. There already seem to be some implementations that have shown some success with combining these two networks together [**3**] [**7**]. There is an important learning synergy between object detection and object tracking, therefore attempting to reuse the proposed object detection solution as much as possible would be strongly recommended. The options of backpropagating through the pre-trained YOLOv5 architecture or freezing these layers and only training the tracking architecture should be investigated.

# Chapter 7

# Conclusion

This project aimed at developing an object detection solution for detecting five types of objects in CGMU surveillance camera feeds: vehicles, pedestrians, buses, cyclists and construction items. Three different types of models were investigated. An SSD model was used as a baseline. YOLOv5 models of various architectural sizes were evaluated. EfficientDet models were also tested but could not lead to interesting results due to difficulty with finding an adequate training configuration. Additional datasets were also researched and evaluated and the MIO-TCD dataset was found to enable significant model performance improvements.

YOLOv5 models were found to outperform the baseline in detection performance and latency. One small (S) and one extra large (X) architectures were selected for conducting hyperparameter search using ASHA. A total of 277 configurations were evaluated for the S model, and 206 for the X model. The hyperparameter exploration enabled finding an improved training configuration for the 'X' architecture, but not for the 'S'. For the latter, the default configuration from YOLOv5's repository performed better. Default hyperparameters were optimized by YOLOv5 authors for achieving the best results on the MS COCO detection task, thus it is not too surprising if it performs very well on the CGMU dataset as well. The best YOLOv5-S model was trained on the CGMU and MIO-TCD datasets and achieves an mAP of 0.738, 0.753 and 0.645 on the validation, in-domain and out-domain test sets respectively. With an inference latency of 167 ms on CPU ($\sim$5 FPS), it is adequate for real-time detection. The best YOLOv5-X model was trained on the CGMU dataset only and achieves an mAP of 0.845, 0.861 and 0.735 on validation, in-domain test and out-domain test sets respectively. It has an inference of 31 ms on GPU, which means it can process images from 6 cameras at $\sim$5 FPS.

With the future goal of anomaly detection in mind, a natural next step to follow would now be to incorporate one of these pre-trained YOLOv5 architectures into a multi-object tracking neural network.

# References


[1] A pytorch tutorial to object detection, 2018. https://github.com/sgrvinod/a-PyTorch-Tutorial-to-Object-Detection.

[2] Computer vision annotation tool (cvat), 2020. https://github.com/openvinotoolkit/cvat.

[3] Deep sort with pytorch, 2020. https://github.com/ZQPei/deep_sort_pytorch.

[4] Efficientdet, 2020. https://github.com/google/automl/tree/master/efficientdet.

[5] Images annotées extraites du flux vidéo des caméras de circulation, 2020. https://donnees.ville.montreal.qc.ca/dataset/images-annotees-cameras-circulation.

[6] Urban segmentation, 2020. https://github.com/VilledeMontreal/urban-segmentation.

[7] Yolov5 + deep sort with pytorch, 2020. https://github.com/mikel-brostrom/Yolov5_DeepSort_Pytorch.

[8] Saber Benchalel : Semantic segmentation of the urban context using ptz cameras, 2020. https://github.com/VilledeMontreal/urban-segmentation/blob/master/Internship_Report_Saber_Benchalel_.pdf.

[9] Alexey Bochkovskiy, Chien-Yao Wang et Hong-Yuan Mark Liao : Yolov4: Optimal speed and accuracy of object detection, 2020.

[10] Navaneeth Bodla, Bharat Singh, Rama Chellappa et Larry S. Davis : Improving object detection with one line of code. *CoRR*, abs/1704.04503, 2017.

[11] Xavier Bouthillier, Christos Tsirigotis, François Corneau-Tremblay, Pierre Delaunay, Reyhane Askari, Dendi Suhubdy, Michael Noukhovitch, Dmitriy Serdyuk, Arnaud Bergeron, Peter Henderson, Pascal Lamblin, Mirko Bronzi et Christopher Beckham : Oríon - asynchronous distributed hyperparameter optimization, octobre 2019.

[12] Pengguang Chen, Shu Liu, Hengshuang Zhao et Jiaya Jia : Gridmask data augmentation, 2020.

[13] Ekin Dogus Cubuk, Barret Zoph, Dandelion Mané, Vijay Vasudevan et Quoc V. Le : Autoaugment: Learning augmentation policies from data. *CoRR*, abs/1805.09501, 2018.

[14] Patrick Dendorfer, Hamid Rezatofighi, Anton Milan, Javen Shi, Daniel Cremers, Ian Reid, Stefan Roth, Konrad Schindler et Laura Leal-Taixé : Mot20: A benchmark for multi object tracking in crowded scenes, 2020.

[15] J. Deng, W. Dong, R. Socher, L. Li, Kai Li et Li Fei-Fei : Imagenet: A large-scale hierarchical image database. *In 2009 IEEE Conference on Computer Vision and Pattern Recognition*, pages 248–255, 2009.

[16] Terrance Devries et Graham W. Taylor : Improved regularization of convolutional neural networks with cutout. *CoRR*, abs/1708.04552, 2017.

[17] A. Dimou, P. Medentzidou, F. Á. García et P. Daras : Multi-target detection in cctv footage for tracking applications using deep learning techniques. *In 2016 IEEE International Conference on Image Processing (ICIP)*, pages 928–932, 2016.

[18] P. Dollar, C. Wojek, B. Schiele et P. Perona : Pedestrian detection: An evaluation of the state of the art. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 34(4):743–761, 2012.

[19] J. Ferryman et A. Shahrokni : Pets2009: Dataset and challenge. *In 2009 Twelfth IEEE International Workshop on Performance Evaluation of Tracking and Surveillance*, pages 1–6, 2009.

[20] Andreas Geiger, Philip Lenz et Raquel Urtasun : Are we ready for autonomous driving? the kitti vision benchmark suite. pages 3354–3361, 05 2012.

[21] Robert Geirhos, Patricia Rubisch, Claudio Michaelis, Matthias Bethge, Felix A. Wichmann et Wieland Brendel : Imagenet-trained cnns are biased towards texture; increasing shape bias improves accuracy and robustness, 2019.

[22] Ross B. Girshick : Fast R-CNN. *CoRR*, abs/1504.08083, 2015.

[23] Ross B. Girshick, Jeff Donahue, Trevor Darrell et Jitendra Malik : Rich feature hierarchies for accurate object detection and semantic segmentation. *CoRR*, abs/1311.2524, 2013.

[24] Jules. Harvey, Adam. LaPlace : Megapixels: Origins and endpoints of biometric datasets "in the wild", 2019.

[25] Kaiming He, Georgia Gkioxari, Piotr Dollár et Ross B. Girshick : Mask R-CNN. *CoRR*, abs/1703.06870, 2017.

[26] Kaiming He, Xiangyu Zhang, Shaoqing Ren et Jian Sun : Deep residual learning for image recognition. *CoRR*, abs/1512.03385, 2015.

[27] Jan Hendrik Hosang, Rodrigo Benenson et Bernt Schiele : Learning non-maximum suppression. *CoRR*, abs/1705.02950, 2017.

[28] Andrew Howard, Mark Sandler, Grace Chu, Liang-Chieh Chen, Bo Chen, Mingxing Tan, Weijun Wang, Yukun Zhu, Ruoming Pang, Vijay Vasudevan, Quoc V. Le et Hartwig Adam : Searching for mobilenetv3. *CoRR*, abs/1905.02244, 2019.

[29] Gao Huang, Zhuang Liu, Laurens van der Maaten et Kilian Q. Weinberger : Densely connected convolutional networks, 2018.

[30] Forrest N. Iandola, Matthew W. Moskewicz, Khalid Ashraf, Song Han, William J. Dally et Kurt Keutzer : Squeezenet: Alexnet-level accuracy with 50x fewer parameters and <1mb model size. *CoRR*, abs/1602.07360, 2016.

[31] Kevin G. Jamieson et Ameet Talwalkar : Non-stochastic best arm identification and hyperparameter optimization. *CoRR*, abs/1502.07943, 2015.

[32] Licheng Jiao, Fan Zhang, Fang Liu, Shuyuan Yang, Lingling Li, Zhixi Feng et Rong Qu : A survey of deep learning-based object detection. *CoRR*, abs/1907.09408, 2019.

[33] Glenn Jocher, Alex Stoken, Jirka Borovec, NanoCode012, ChristopherSTAN, Liu Changyu, Laughing, tkianai, Adam Hogan, lorenzomammana, yxNONG, AlexWang1900, Laurentiu Diaconu, Marc, wanghaoyang0106, ml5ah, Doug, Francisco Ingham, Frederik, Guilhen, Hatovix, Jake Poznanski, Jiacong Fang, Lijun Yu , changyu98, Mingyu Wang, Naman Gupta, Osama Akhtar, PetrDvoracek et Prashant Rai : ultralytics/yolov5: v3.1 - Bug Fixes and Performance Improvements, octobre 2020.

[34] Sven Koitka et Christoph Friedrich : Optimized convolutional neural network ensembles for medical subfigure classification. 09 2017.

[35] Alina Kuznetsova, Hassan Rom, Neil Alldrin, Jasper R. R. Uijlings, Ivan Krasin, Jordi Pont-Tuset, Shahab Kamali, Stefan Popov, Matteo Malloci, Tom Duerig et Vittorio Ferrari : The open images dataset V4: unified image classification, object detection, and visual relationship detection at scale. *CoRR*, abs/1811.00982, 2018.

[36] Laura Leal-Taixé, Anton Milan, Ian Reid, Stefan Roth et Konrad Schindler : Motchallenge 2015: Towards a benchmark for multi-target tracking, 2015.

[37] Hengduo Li, Bharat Singh, Mahyar Najibi, Zuxuan Wu et Larry S. Davis : An analysis of pre-training on object detection. *CoRR*, abs/1904.05871, 2019.

[38] Liam Li, Kevin G. Jamieson, Afshin Rostamizadeh, Ekaterina Gonina, Moritz Hardt, Benjamin Recht et Ameet Talwalkar : Massively parallel hyperparameter tuning. *CoRR*, abs/1810.05934, 2018.

[39] Lisha Li, Kevin Jamieson, Giulia DeSalvo, Afshin Rostamizadeh et Ameet Talwalkar : Hyperband: A novel bandit-based approach to hyperparameter optimization, 2018.

[40] Tsung-Yi Lin, Priya Goyal, Ross Girshick, Kaiming He et Piotr Dollár : Focal loss for dense object detection, 2018.

[41] Tsung-Yi Lin, Priya Goyal, Ross B. Girshick, Kaiming He et Piotr Dollár : Focal loss for dense object detection. *CoRR*, abs/1708.02002, 2017.

[42] Tsung-Yi Lin, Michael Maire, Serge J. Belongie, Lubomir D. Bourdev, Ross B. Girshick, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár et C. Lawrence Zitnick : Microsoft COCO: common objects in context. *CoRR*, abs/1405.0312, 2014.

[43] Li Liu, Wanli Ouyang, Xiaogang Wang, Paul W. Fieguth, Jie Chen, Xinwang Liu et Matti Pietikäinen : Deep learning for generic object detection: A survey. *CoRR*, abs/1809.02165, 2018.

[44] Wei Liu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott E. Reed, Cheng-Yang Fu et Alexander C. Berg : SSD: single shot multibox detector. *CoRR*, abs/1512.02325, 2015.

[45] Zhiming Luo, Frederic B.-Charron, Carl Lemaire, Janusz Konrad, Shaozi Li, Akshaya Mishra, Andrew Achkar, Justin Eichel et Pierre-Marc Jodoin : Mio-tcd: A new benchmark dataset for vehicle classification and localization. *IEEE Transactions on Image Processing*, PP:1–1, 06 2018.

[46] Ningning Ma, Xiangyu Zhang, Hai-Tao Zheng et Jian Sun : Shufflenet V2: practical guidelines for efficient CNN architecture design. *CoRR*, abs/1807.11164, 2018.

[47] Andrew L. Maas, Awni Y. Hannun et Andrew Y. Ng : Rectifier nonlinearities improve neural network acoustic models. *In in ICML Workshop on Deep Learning for Audio, Speech and Language Processing*, 2013.

[48] A. Milan, L. Leal-Taixé, I. Reid, S. Roth et K. Schindler : Mot16: A benchmark for multi-object tracking. *ArXiv*, abs/1603.00831, 2016.

[49] Waseem Rawat et Zenghui Wang : Deep convolutional neural networks for image classification: A comprehensive review. *Neural Computation*, 29:1–98, 06 2017.

[50] Joseph Redmon, Santosh Kumar Divvala, Ross B. Girshick et Ali Farhadi : You only look once: Unified, real-time object detection. *CoRR*, abs/1506.02640, 2015.

[51] Joseph Redmon et Ali Farhadi : YOLO9000: better, faster, stronger. *CoRR*, abs/1612.08242, 2016.

[52] Joseph Redmon et Ali Farhadi : Yolov3: An incremental improvement. *CoRR*, abs/1804.02767, 2018.

[53] Shaoqing Ren, Kaiming He, Ross B. Girshick et Jian Sun : Faster R-CNN: towards real-time object detection with region proposal networks. *CoRR*, abs/1506.01497, 2015.

[54] Karen Simonyan et Andrew Zisserman : Very deep convolutional networks for large-scale image recognition. *arXiv 1409.1556*, 09 2014.

[55] Krishna Kumar Singh, Hao Yu, Aron Sarmasi, Gautam Pradeep et Yong Jae Lee : Hide-and-seek: A data augmentation technique for weakly-supervised localization and beyond. *CoRR*, abs/1811.02545, 2018.

[56] Christian Szegedy, Sergey Ioffe et Vincent Vanhoucke : Inception-v4, inception-resnet and the impact of residual connections on learning. *CoRR*, abs/1602.07261, 2016.

[57] Christian Szegedy, Scott Reed, Dumitru Erhan, Dragomir Anguelov et Sergey Ioffe : Scalable, high-quality object detection, 2015.

[58] Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jonathon Shlens et Zbigniew Wojna : Rethinking the inception architecture for computer vision. *CoRR*, abs/1512.00567, 2015.

[59] Mingxing Tan et Quoc V. Le : Efficientnet: Rethinking model scaling for convolutional neural networks, 2020.

[60] Mingxing Tan, Ruoming Pang et Quoc V. Le : Efficientdet: Scalable and efficient object detection, 2020.

[61] Chien-Yao Wang, Hong-Yuan Mark Liao, I-Hau Yeh, Yueh-Hua Wu, Ping-Yang Chen et Jun-Wei Hsieh : Cspnet: A new backbone that can enhance learning capability of cnn, 2019.

[62] Chuang Wang, Ruimin Hu, Min Hu, Jiang Liu, Ting Ren, Shan He, Ming Jiang et Jing Miao : Lossless attention in convolutional networks for facial expression recognition in the wild, 2020.

[63] Xiaolong Wang, Abhinav Shrivastava et Abhinav Gupta : A-fast-rcnn: Hard positive generation via adversary for object detection. *CoRR*, abs/1704.03414, 2017.

[64] Nicolai Wojke, Alex Bewley et Dietrich Paulus : Simple online and realtime tracking with a deep association metric, 2017.

[65] Saining Xie, Ross B. Girshick, Piotr Dollár, Zhuowen Tu et Kaiming He : Aggregated residual transformations for deep neural networks. *CoRR*, abs/1611.05431, 2016.

[66] Moi Hoon Yap, Ryo Hachiuma, Azadeh Alavi, Raphael Brungel, Manu Goyal, Hongtao Zhu, Bill Cassidy, Johannes Ruckert, Moshe Olshansky, Xiao Huang, Hideo Saito, Saeed Hassanpour, Christoph M. Friedrich, David Ascher, Anping Song, Hiroki Kajita, David Gillespie, Neil D. Reeves, Joseph Pappachan, Claire O'Shea et Eibe Frank : Deep learning in diabetic foot ulcers detection: A comprehensive evaluation, 2020.

[67] Sangdoo Yun, Dongyoon Han, Seong Joon Oh, Sanghyuk Chun, Junsuk Choe et Youngjoon Yoo : Cutmix: Regularization strategy to train strong classifiers with localizable features. *CoRR*, abs/1905.04899, 2019.

[68] Hongyi Zhang, Moustapha Cissé, Yann N. Dauphin et David Lopez-Paz : mixup: Beyond empirical risk minimization. *CoRR*, abs/1710.09412, 2017.

[69] Shifeng Zhang, Longyin Wen, Xiao Bian, Zhen Lei et Stan Z. Li : Single-shot refinement neural network for object detection. *CoRR*, abs/1711.06897, 2017.

[70] Xiangyu Zhang, Xinyu Zhou, Mengxiao Lin et Jian Sun : Shufflenet: An extremely efficient convolutional neural network for mobile devices. *CoRR*, abs/1707.01083, 2017.

[71] Zhi Zhang, Tong He, Hang Zhang, Zhongyue Zhang, Junyuan Xie et Mu Li : Bag of freebies for training object detection neural networks, 2019.

[72] Qijie Zhao, Tao Sheng, Yongtao Wang, Zhi Tang, Ying Chen, Ling Cai et Haibin Ling : M2det: A single-shot object detector based on multi-level feature pyramid network. *CoRR*, abs/1811.04533, 2018.

[73] Zhaohui Zheng, Ping Wang, Wei Liu, Jinze Li, Rongguang Ye et Dongwei Ren : Distance-iou loss: Faster and better learning for bounding box regression, 2019.

[74] Zhun Zhong, Liang Zheng, Guoliang Kang, Shaozi Li et Yi Yang : Random erasing data augmentation. *CoRR*, abs/1708.04896, 2017.

[75] Barret Zoph, Ekin D. Cubuk, Golnaz Ghiasi, Tsung-Yi Lin, Jonathon Shlens et Quoc V. Le : Learning data augmentation strategies for object detection. *CoRR*, abs/1906.11172, 2019.

# Appendix A

## Sample inferences from best models on test set images


(a) Original image

(b) YOLOv5-S-512

(c) YOLOv5-X-512

(d) YOLOv5-X-704

**Fig. A.1.** Sunny day, large resolution


(a) Original image

(b) YOLOv5-S-512

(c) YOLOv5-X-512

(d) YOLOv5-X-704

**Fig. A.2.** Sunny day, small resolution

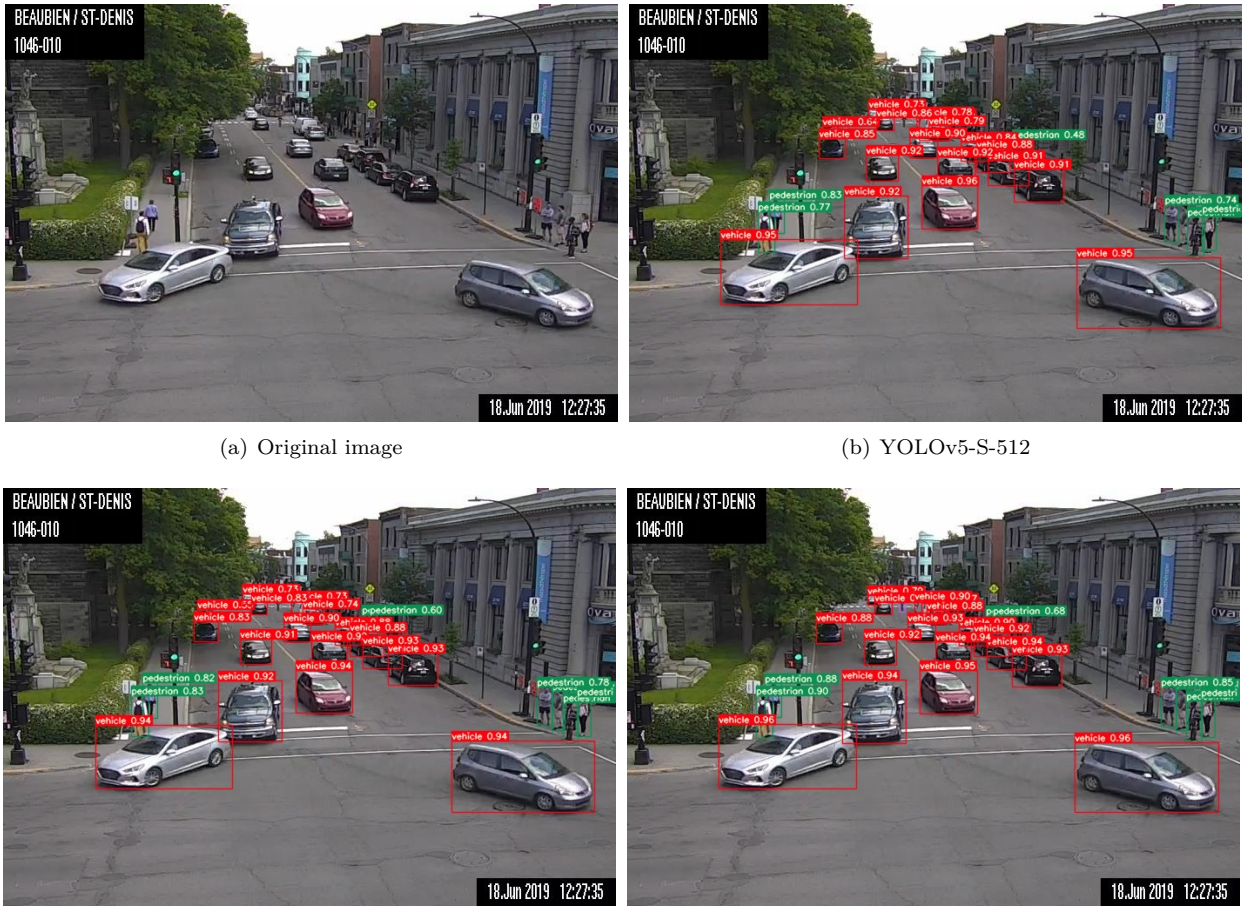

(a) Original image

(b) YOLOv5-S-512

(c) YOLOv5-X-512

(d) YOLOv5-X-704

**Fig. A.3.** Sunny day with glare, small resolution



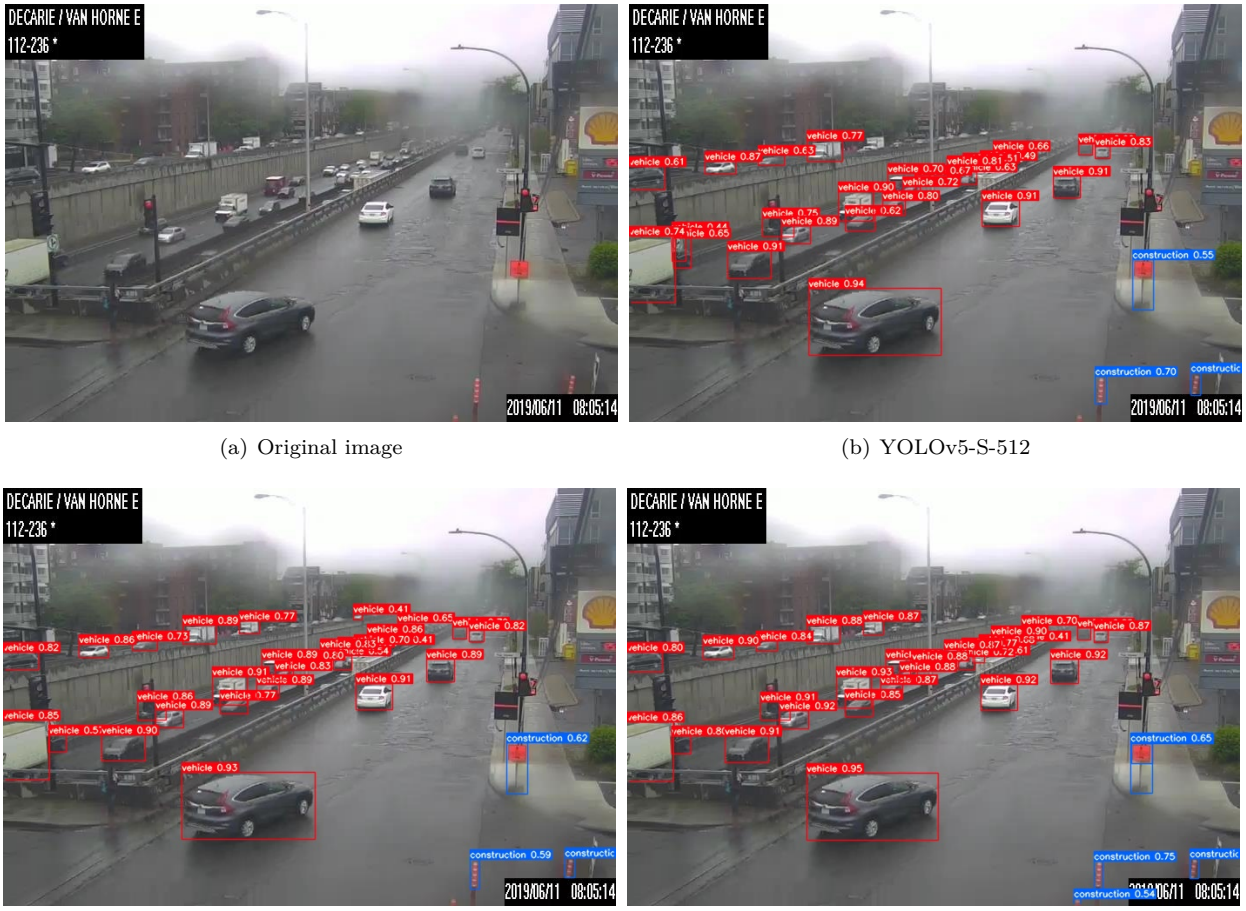

(a) Original image

(b) YOLOv5-S-512

(c) YOLOv5-X-512

(d) YOLOv5-X-704

**Fig. A.4.** Sunny day, large resolution



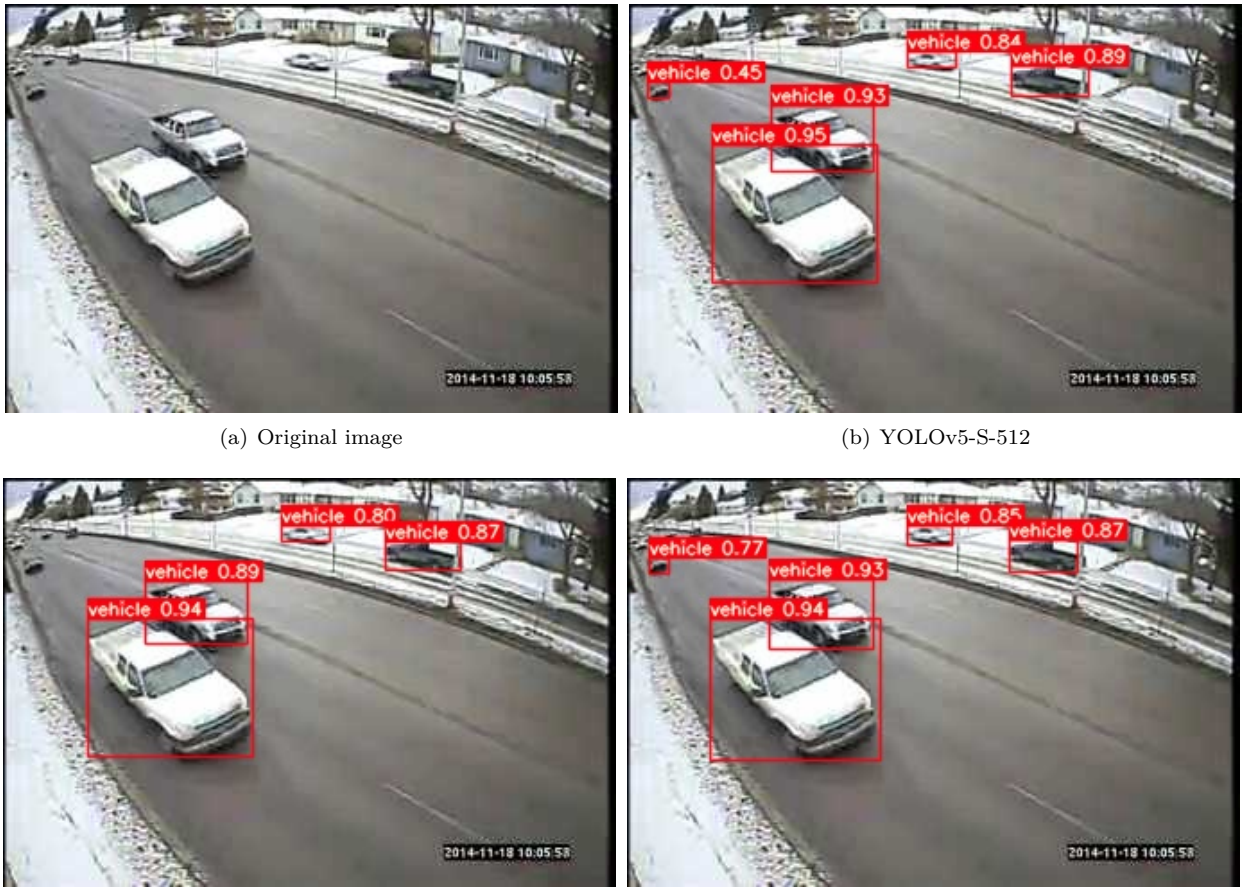

(a) Original image

(b) YOLOv5-S-512

(c) YOLOv5-X-512

(d) YOLOv5-X-704

**Fig. A.5.** Rainy day with droplets on lens



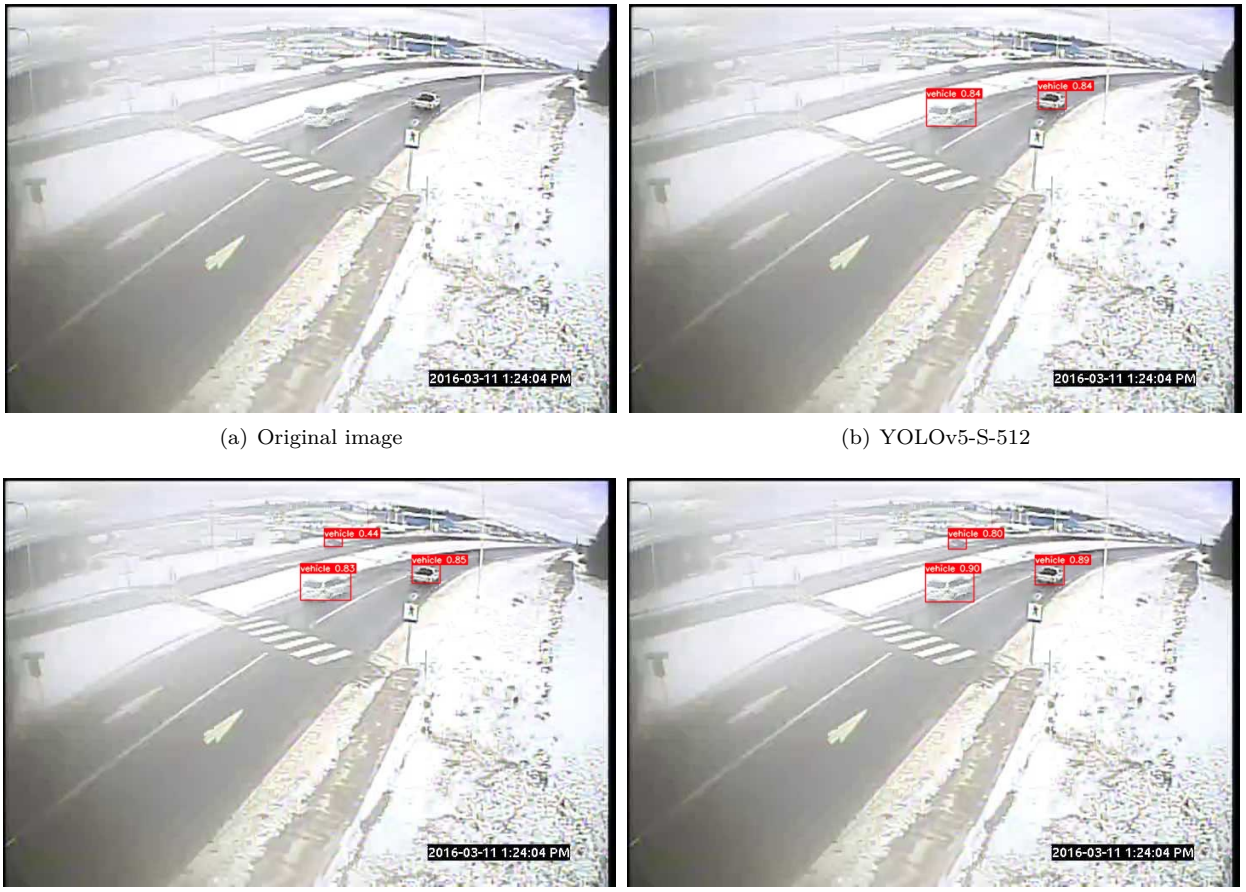

(a) Original image

(b) YOLOv5-S-512

(c) YOLOv5-X-512

(d) YOLOv5-X-704

**Fig. A.6.** MIO-TCD image in winter time, small resolution



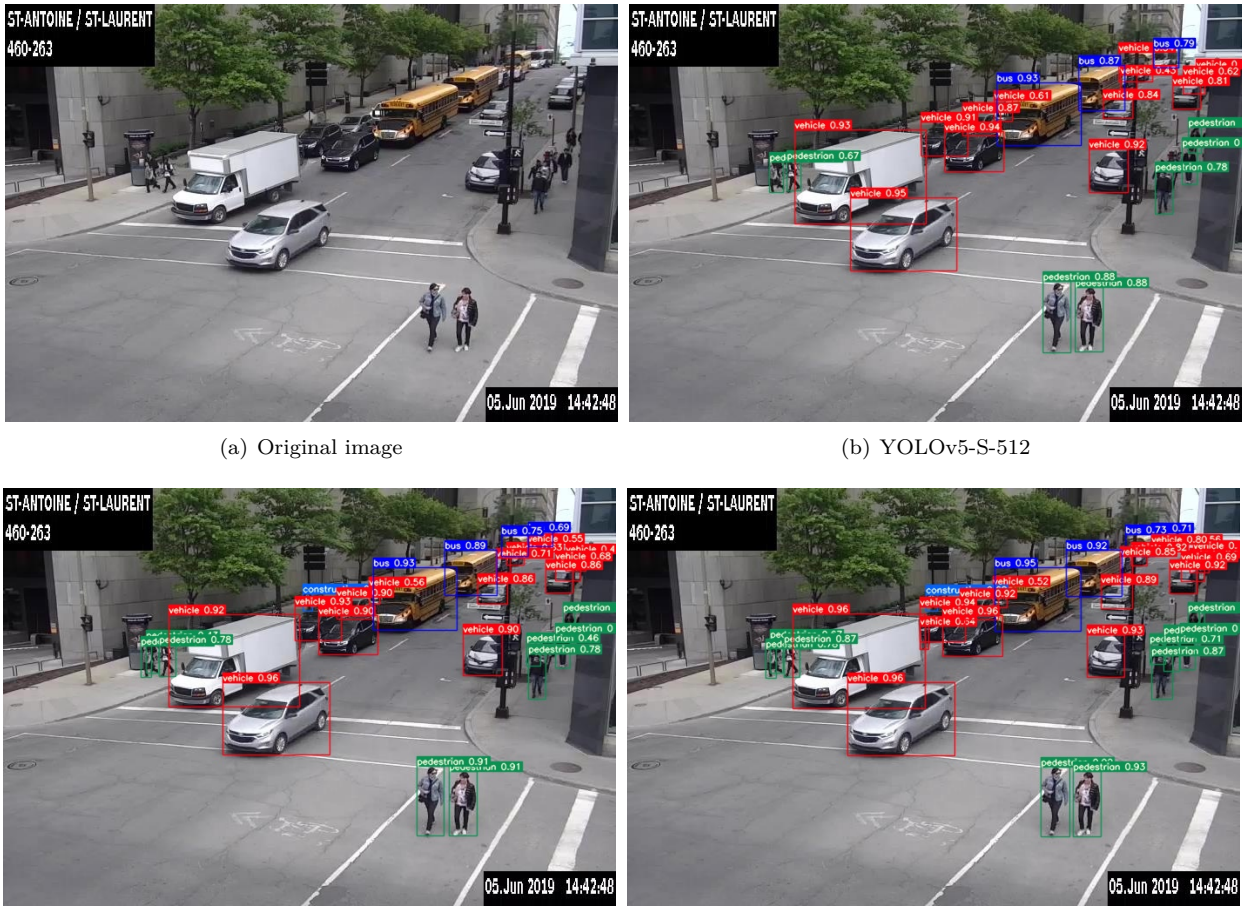

(a) Original image

(b) YOLOv5-S-512

(c) YOLOv5-X-512

(d) YOLOv5-X-704

**Fig. A.7.** MIO-TCD image in winter time with fog, large resolution



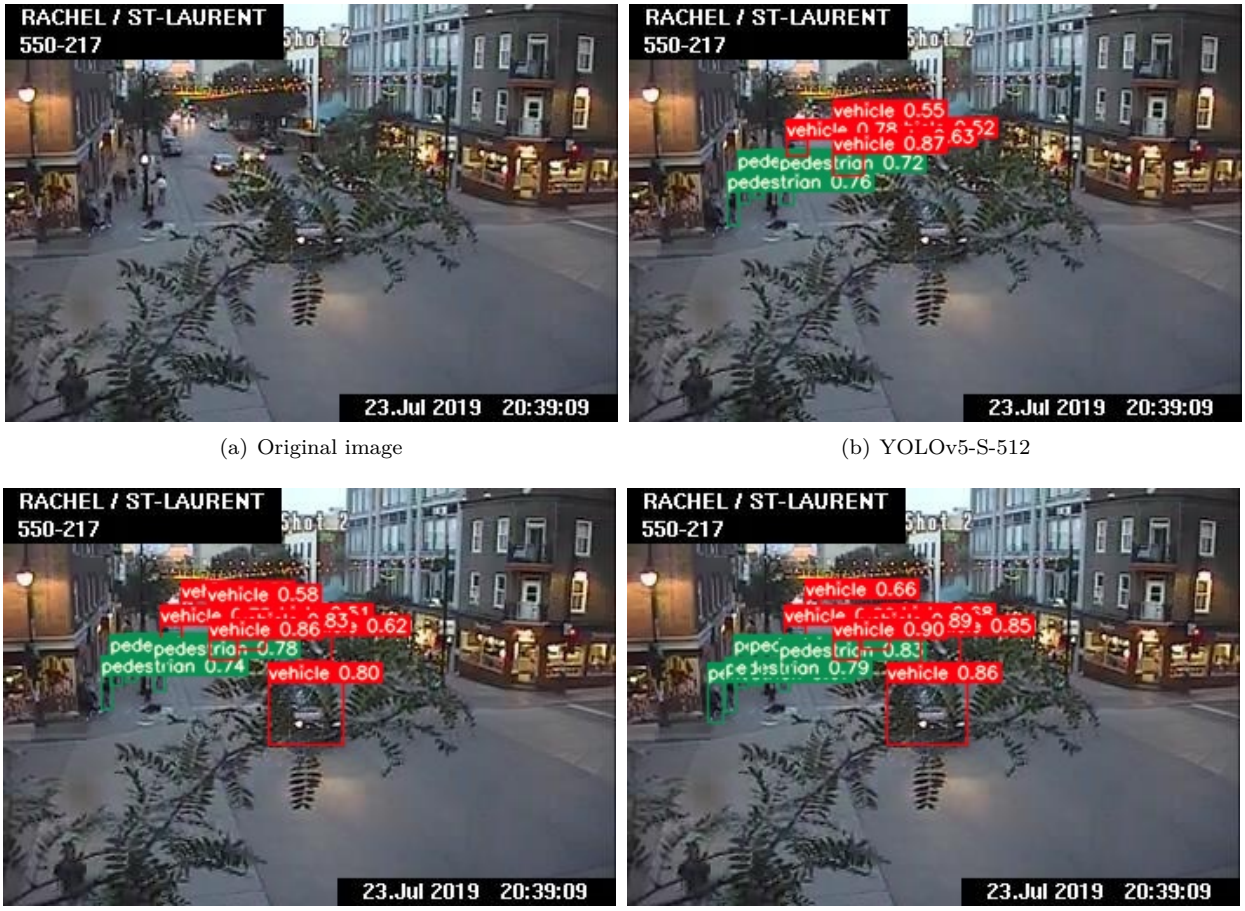

(a) Original image

(b) YOLOv5-S-512

(c) YOLOv5-X-512

(d) YOLOv5-X-704

**Fig. A.8.** Sunny day, large resolution




(a) Original image

(b) YOLOv5-S-512

(c) YOLOv5-X-512

(d) YOLOv5-X-704

**Fig. A.9.** Sunny day with occlusion from tree branch, small resolution

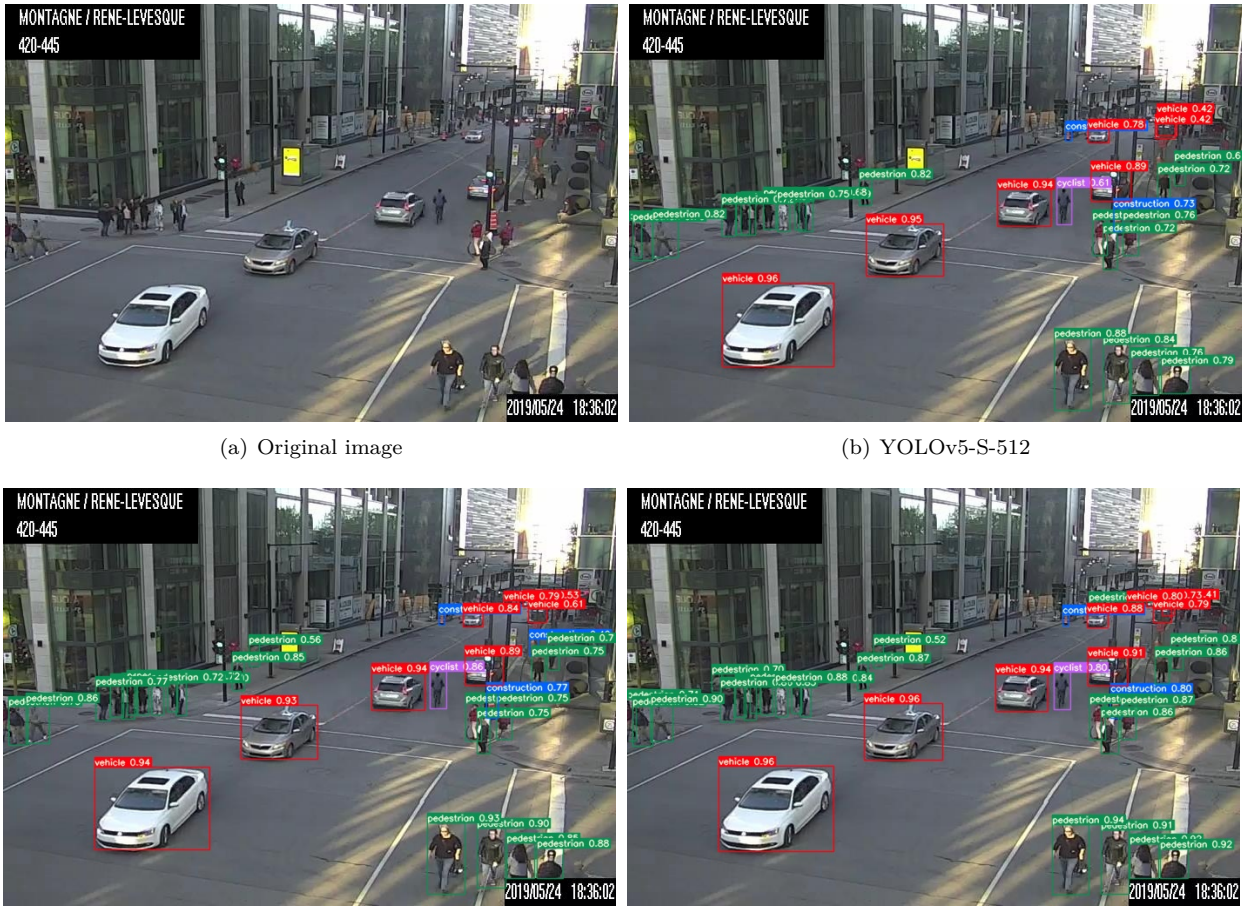

(a) Original image

(b) YOLOv5-S-512

(c) YOLOv5-X-512

(d) YOLOv5-X-704

**Fig. A.10.** Night time, out of focus with glare, small resolution



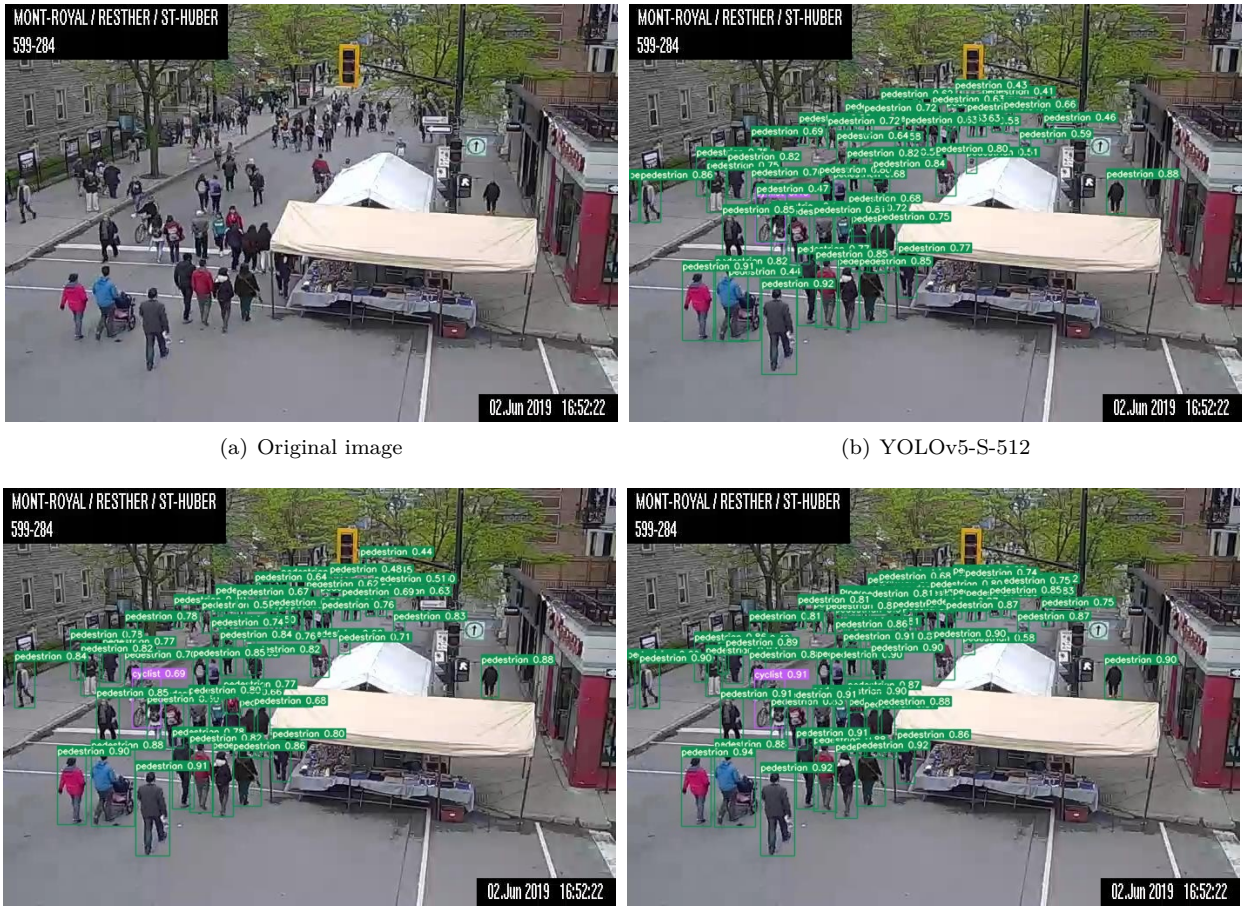

(a) Original image

(b) YOLOv5-S-512

(c) YOLOv5-X-512

(d) YOLOv5-X-704

**Fig. A.11.** Sunny day, large resolution




(a) Original image

(b) YOLOv5-S-512

(c) YOLOv5-X-512

(d) YOLOv5-X-704

**Fig. A.12.** Sunny day with crowded space, large resolution