**Hannu-Tapani Turtiainen**

# State-of-the-art object detection model for detecting CCTV and video surveillance cameras from images and videos

Master's Thesis in Information Technology

May 22, 2020

University of Jyväskylä

Faculty of Information Technology

**Author:** Hannu-Tapani Turtiainen

**Contact information:** hannu.ht.turtiainen@student.jyu.fi

**Supervisor:** Andrei Costin

**Title:** State-of-the-art object detection model for detecting CCTV and video surveillance cameras from images and videos

**Työn nimi:** Viimeisintä tekniikkaa sisältävä objektintunnistusmalli havaitsemaan turvakameroita kuvista ja videoista

**Project:** Master's Thesis

**Study line:** Cyber security

**Page count:** 94+1

**Abstract:** As the current GDPR law in the EU prohibits unnecessary use of CCTV cameras in public places, and privacy concerns of smart CCTV cameras have been raised, CCTV cameras cannot be regarded as just easy tools to help secure your assets. Smart technologies such as facial recognition have reached CCTV cameras and the amount of data gathered from them is expanding rapidly. This thesis explores how to use state-of-the-art object detection architectures and frameworks to create models to detect CCTV cameras from images (e.g., street view) and video. A literature review was performed to establish a fundamental understanding of object detector algorithms. The metrics from Microsoft Common Objects in Context were used to evaluate the detectors as state-of-the-art since most recent architectures have been tested with it. The selection process also took into account the framework around the detector as the tools in use are essential for the future refinement and adoption of the model. Three detectors were identified as prime candidates; CenterMask, ATSS, and TridentNet. This thesis developed a set of state-of-the-art detectors for 'CCTV-camera' objects achieving over 90% mAP@0.5 and with further possibilities, improvements, and testing datasets being disclosed. This thesis is a part of a three-way project collaboration with two other M.Sc. theses being written by Tuomo Lahtinen and Lauri Sintonen. The intention is to create a toolset to improve the detection model further and to use it for mapping CCTV cameras from street view images and create safety-centric routing and navigational suggestions.

**Keywords:** CCTV cameras, computer vision, object detection, state-of-the-art, convolutional neural networks

**Suomenkielinen tiivistelmä:** CCTV kameroita ei voida nykyisin pitää helppoina työkaluina omaisuuden turvaamiseen, sillä huoli ihmisten yksityisyydestä on noussut esiin kameroista puhuttaessa. EU:n tietosuoja-asetus GDPR kieltääkin tarpeettomien CCTV kameroiden käytön julkisilla paikoilla. Kasvojentunnistus ja muut älyteknologiat ovat soluttautuneet turvakameroihin ja niistä kerätyn datan määrä on kasvanut nopeasti. Tässä tutkielmassa luodaan viimeisintä tekniikkaa sisältävä objektintunnistusmalli havaitsevaan CCTV kameroita kuvista (esim. katutasokuvat) ja videoista. Tutkielman kirjallisuuskatsauksessa luodaan perustason ymmärrystä objektintunnistusalgoritmeista. Tutkielmassa käytetyt algoritmit tunnistettiin olevan viimeisintä tekniikkaa edustavia Microsoft Common Objects in Context - datakokoelman mittareiden avulla, sillä useimmat algoritmit testataan niiden avulla. Käytettävien algoritmien valintaan vaikutti myös niiden käytössä tarvittava viitekehys, sillä sen toiminta vaikuttaa mallin käyttöönottoon ja kehitykseen. Kolme tunnistusalgoritmia valittiin ehdokkaiksi; CenterMask, ATSS ja TridentNet. Tutkielmassa kehitettiin viimeisintä tekniikkaa edustavat objektintunnistusmallit havaitsemaan CCTV kameraobjekteja ja tuloksena saavutettiin yli 90% mAP@0.5 ja tulevaisuuden mahdollisuudet, kehittämisideat ja uudet testausdatakokoelmat tuotiin esille. Tämä tutkielma on osa kolmen maisteritason tutkielman yhteistyötä, joista kaksi muuta toteuttaa Tuomo Lahtinen ja Lauri Sintonen. Tarkoituksena on luoda työkalut tunnistusmallin päivittämiseen ja käyttää sitä havaitsemaan CCTV kameroita katutasokuvista ja siten luomaan yksityisyyttä parantavia navigaatioreittiehdotelmia.

**Avainsanat:** CCTV kamerat, tietokonenäkö, objektintunnistus, konvolutiiviset neuroverkot

# Abbreviations

| | |
|---|---|
| API | Application Programming Interface |
| AP | Average Precision |
| AR | Average Recall |
| BN | Batch Normalization |
| CPU | Central Processing Unit |
| CCTV | Closed-Circuit Television |
| CUDA | Compute Unified Device Architecture |
| CNN, ConvNet | Convolutional Neural Network |
| eSE | Effective Squeeze-Excitationt |
| EDPB | European Data Protection Board |
| EVGP | Exploding/Vanishing Gradient Problem |
| FAIR | Facebook AI Research lab |
| FRVT | Facial Recognition Vendor Test |
| FPN | Feature Pyramid Networks |
| FGCI | Finnish Grid and Cloud Infrastructure |
| FPS | Frames-per-Second |
| GDPR | General Data Protection Regulation |
| GPGPU | General-Purpose Graphics Processing Units |
| GPU | Graphics Processing Unit |
| HOG | Histograms of Oriented Gradients |
| ILSVRC | ImageNet Large Scale Visual Recognition Challenge |
| IoT | Internet of Things |
| IoU | Intersection-over-Union |
| LBP | Local Binary Pattern |
| mAP | Mean Average precision |
| MSE | Mean Square Error |
| MS COCO | Microsoft Common Objects in Context |
| NIST | National Institute of Standard and Technology |
| NMS | Non-maximum Suppression |

| | |
|---|---|
| OSA | One-shot Aggregation |
| OpenCV | Open Source Computer Vision Library |
| Pascal VOC | Pattern Analysis, Statistical Modelling and Computational Lerning Visual Object Classes |
| ReLU | Rectified Linear Unit |
| R-CNN | Region-Convolutional Neural Network |
| RoI | Region of Interest |
| RPN | Region Proposal Network |
| SIFT | Scale Invariant Fea-ture Transform |
| SNIP | Scale Normalization for Image Pyramids |
| SAG-Mask | Spatial Attention-Guided Mask |
| SAM | Spatial Attention module |
| SPM | Spatial Pyramid Matching |
| SPP | Spatial Pyramid Pooling |
| SE | Squeeze-and-Excitation Networks |
| SGD | Stochastic Gradient Descent |
| SVM | Support Vector Machine |
| JYU | University of Jyväskylä |
| VRAM | Video Random Access Memory |

# List of Figures

## List of Tables

# Contents

# 1 Introduction

CCTV cameras and surveillance systems have always been controversial topics. Some may view CCTV cameras as a necessity for securing peace and their assets, and others view them as an infringement of their privacy - a way for 'big brother' to watch over their every move. An estimate entails that there are over one billion CCTV cameras around the world in 2021. That constitutes a rise of over a quarter from the estimates of today's 770 million cameras of which over half is located in China (Philippou 2019). Still, estimates vary with the methods used and the availability of data. Many analysts, such as Comparitech (Bischoff 2019), show equal growth numbers for CCTV camera installations. However, the core challenge of accurately counting the CCTV cameras in the real-world persists, and this thesis is one step forward towards solving that challenge.

Datasets of the locations of publicly placed cameras have been gathered (see f.ex. *THE CCTV MAP*). Still, they lack the ability for easy automatic updates, and the availability of them in many regions is negligible. These problems are due to the manual nature of the data gathering method. In this thesis, an alternative method to expose CCTV camera locations is proposed. Proposed model uses state-of-the-art computer vision algorithms to generate an object detection model to detect and map CCTV cameras from geotagged images from sources such as Google StreetView (*Street View Photos Come from Two Sources, Google and Our Contributors.*), OpenStreetCam (*OpenStreetCam*), and Flickr (*Flickr*). At the time of writing and to the best of our knowledge, there is no comparable object detection model or an image dataset publicly available for training computer vision models and detecting CCTV cameras. In this thesis, we also introduce and motivate several real-world practical uses of such a tool.

The conducted literature review explores the object detection field and the underlying technology behind the object detectors. As the field of object detection is evolving quickly and new architectures pop up quite frequently, the sources for this thesis must be up to date. Therefore, the review is concluded by studying explanatory eBooks of convolutional neural networks, review papers, and blogs, architecture explanations as well as the papers of the respective object detectors. Framework websites and documentation are needed to understand the underlying mechanics.

Three state-of-the-art object detection architectures were selected for evaluation; CenterMask (Lee and Park 2019), ATSS (S. Zhang et al. 2019), and TridentNet (Li et al. 2019) as they were top 10 performers in the MS COCO (T. Lin et al. 2014) test-dev dataset with the code publicly available (*Papers With Code*). They all are implemented on the Facebook Artificial Intelligence Research's (*Facebook AI Research*) Detectron/maskrcnn-benchmark frameworks (Wu et al. 2020), which provide good future-proofing and ready-made tools. This thesis aims to answer the following main research question one and supportive research questions:

- RQ1: "How can we accurately and efficiently detect 'CCTV camera' objects from images and videos?"
- RQ2: "How do the various trained models fare against each other?"
- RQ3: "How can we improve on the models?"
- RQ4: "What are the main challenges and applicative use-cases of such technology?"

A dataset of CCTV cameras was gathered and further split into training and evaluation datasets. A separate testing dataset was also gathered to establish results from images that correspond to the images provided by the intended use case of the model. Two types of results are presented; metrics from the validation and testing datasets in the form of MS COCO (T. Lin et al. 2014) evaluation format, and visual results.

Our research involves creating object detection models and evaluating them for our use case; thus, the research is very applied and practically oriented. The performance of the models is evaluated with numerical data. Although the performance is also evaluated by visually inspecting the images, they are still presenting numerical information. The visual aspect is not relevant in any other means except for demonstration purposes. As there are three models created and evaluated in the thesis, the chosen research method could have been a case study. Each model might have acted as a case, they could have been individually evaluated, and generalizations could have been determined from the results. The issue with this approach is that, although the chosen detectors have been justified as prime candidates, there is no way with reasonable resources, to be sure that the detectors are the most suitable or best performing of the available options for our specific dataset and needs. Therefore, the main focus of the research is not the comparison of the chosen detectors, but the realization of the model and the determination that the intended use case for it can be achieved. Thus this thesis follows Design Science Research Methodology (DSRM) for information systems (Peffers et al. 2006).

The focus of DSRM research is the development of an applicable artifact. The purpose of the theory is to propose the arguments that the artifact can be constructed and that it could work. The evaluation and demonstration of the model bring out the practicality of the research, and therefore, it is very outcome-oriented. A proper DSRM research proposes sound evaluation, produces a generalizable artifact, and is properly motivated. (Peffers et al. 2006). The steps of DSRM in this thesis are presented in sequential order, although they were not conceptualized in that order. The DSRM process is flexible; it can begin from any step and then process forward (Peffers et al. 2006).

This thesis is organized in 5 chapters. Chapter 2 introduces the issues regarding CCTV cameras and one's privacy. The chapter adduces the motivation behind model creation. Chapter 3 examines the object detection study space and presents the architectures of the three object detectors. Chapter 4 presents the methodology used in the creation of the models as well as the environments and resources employed. In chapter 5, we declare the results of the model evaluation. Finally, in chapter 6, we reflect on the models, and the results achieved, as well as discuss the possible future work. Any releasable code and data related to this thesis will be available at https://github.com/Fuziih.

# 2 OVERVIEW OF ISSUES RELATED TO CCTV CAMERAS

Privacy and use case issues concerning CCTV cameras are relevant now more than ever as they are being implemented with smart features (see Alshammari and Rawat 2019; Akbar and Azhar 2018). The privacy issues are in danger of being neglected over the benefits gained from these systems, and the users need to be aware of the laws such as GDPR in order to use the cameras properly. This chapter promotes a few of these issues and explores how the proposed model could be used in the matter.

Peffers et al. (2006) describe the first activity of Design science research to be problem identification and motivation. The aim is to advocate the value of the research and its solution. The justification will provide the motivation to seek the solution. The next step, according to Peffers et al. (2006), is defining the objectives of the solution. The objectives should be addressed in a rational manner and the way the artifact should aid in the solving of the issues presented. This chapter concludes the problem identification, motivation and proposes ways the artifact can help in regards to the issues, in turn, answering the research question four; RQ4: "What are the main challenges and applicative use-cases of such technology?".

## 2.1 Privacy and GDPR

Within the European Union, the General Data Protection Regulation (GDPR) was implemented on 25.5.2018 (European Commission). The law is vast; thus, misinterpretations and vagueness can be an issue. Therefore, the European Data Protection Board (EDPB) issued guidelines for public review on the processing of personal data gathered through video devices on 10.7.2019, and these guidelines were adopted on 29.1.2020. In terms of video surveillance, the regulation raises concerns that video and audio recording technologies may omit the possibilities for anonymous movements for people. While there is certainly a necessity for CCTV cameras in some scenarios, they should be considered as a final option. While they state that individuals can be unconcerned with the existing surveillance, the concerns of misuse of the recorded material are prevailing. The use of smart technologies in conjunction with CCTV cameras make the recording intrusive to one's privacy as they combine biometrics to the data. (European Data Protection Board 2020).

The board maintains that video surveillance should be adopted as a necessity, not as a convenience and that the installation of the cameras, which monitor public areas, should be regulated. People who are recorded have the right to request information about the data stored about them and can request the removal of it after a reasonable time. After 72 hours of data storage, the data owner must prove the necessity for data storing. (European Data Protection Board 2020).

The EDPB expects organizations to consider having policies in place to manage their surveillance system. Responsibilities should be appointed, and the system should be managed accordingly. The scope of the surveillance, the necessity, and purpose should be documented for a further inquest. Transparency of the conducted surveillance is crucial as well as the appropriate use of the data. The member states of the EU can propose their specific legislation to comply and adapt to the GDPR (European Data Protection Board 2020). According to the CMS GDPR enforcement tracker, several fines for unauthorized use of CCTV cameras have been issued around the EU already (*GDPR Enforcement Tracker - List of GDPR Fines*).

With the regulation in place, member states need to enforce it and maintain the necessity requirement for surveillance systems as well as protect the privacy of those recorded by these systems. Therefore, it is fundamental to keep track of CCTV camera installations to limit unlawful or otherwise malicious surveillance systems. The proposed model helps to identify camera installations in public. With the proposed model, one can quickly go through vast amounts of geotagged image datasets or video recordings to look for CCTV cameras. The model could be in the future implemented to a framework that could map the positive findings to help associate the surveillance systems to their owners through an address database. The mapping could prove helpful in automating the enforcement of the GDPR in regards to surveillance systems.

## 2.2 Placement of CCTV cameras

There is still a necessity for many CCTV cameras. The cameras need to be placed properly to minimize cost, increase system performance (Murray et al. 2007), and limit the recording to an appropriate area. Prior research for optimum camera placement has been conducted (see Erdem and Sclaroff 2004; Kenichi Yabuta and Hitoshi Kitazawa 2008; Murray et al. 2007). These papers propose solutions to optimizing the coverage of an area with the least number of cameras placed. To truly optimize the system, the problem must be handled with visibility analysis in three dimensions. With tracking, detection, recognition, and other smart technologies included in these cameras, proper placement becomes even more essential (Murray et al. 2007). Some tools already exist for use by the CCTV industry, such as 'IPVM Camera Calculator' (*IPVM Camera Calculator V3*).

In future work, the proposed detection model could be used in conjunction with such placement algorithms to detect current camera placements and submit them to the system automatically. The model could be used to validate the new installations as well. Another valid use of such a tool would be in law enforcement. The area of rallies or public speeches could be scanned for CCTV cameras, and in conjunction with a coverage algorithm, blind-spots could be identified.

## 2.3 Facial recognition

Facial recognition is conceptually quite simple. First, an image is captured, and faces are detected from it. Faces in the image are analyzed, and features are extracted. These features are compared to those from a database, which contains known facial features of individuals. A certainty score can be determined with this comparison. (John D. Woodward et al. 2003). These features, called the face-print, are unique enough to determine the theory behind facial recognition can be valid. Although facial recognition is not as accurate as other biometric identification methods such as fingerprints, it holds the advantage of being usually inconspicuous to the target (Andrejevic and Selwyn 2019).

Facial recognition is a highly debated topic in recent times. The European Union was supposedly contemplating a ban for facial recognition systems across public areas for a three to five-year time-period to assess the technology and conduct a risk assessment. (Stolton 2020). This approach was later softened to a reminder that the use of such technology in public places is highly risky for the fundamental rights of citizens. In the artificial intelligence white paper released by the European Commission on 19.2.2020, the commission states that in principle, GDPR prohibits the use of uniquely identifying biometric data processing systems in public places, except in the conditions of GDPR Article 10, which is the Law Enforcement Directive. (European Commission 2020).

National Institute of Standards and Technology (NIST) started a facial recognition algorithm evaluation campaign in 2017. The Facial Recognition Vendor Test -project (Grother, Ngan, and Hanaoka 2019) releases several reports over the year. One of the latest reports is from December of 2019, and it addresses the demographic effects of the participating state-of-the-art algorithms. The findings indicate that the algorithms are biased, and the detection precision across ethnic, age, and gender groups varies highly. NIST found in their evaluation empirical evidence that the majority of these algorithms had demographic differences in the results. NIST informs that the false positive rate difference could be 100 times or more between subjects from different countries. Algorithms developed in western countries showed a high false positive rate with people from Africa and Asia. With algorithms developed in Asia, the results were reversed, making Asian people have the lowest false positive rate. Although false negative rate differences between algorithms were significant, they were not as great as those of false positives. The accuracy results, in general, were highly developer dependent. The report finds that false positive rates were two to five times higher with women than men and that false positives were elevated in the demographics of children and the elderly. False negative errors were present more with women and children, but these results vary highly and are not generally conclusive. In all cases, image quality can skew results highly. (Grother, Ngan, and Hanaoka 2019).

These current results from NIST contribute to the story that facial recognition software can be biased. These results are not surprising as it can be challenging to create an unbiased and equal dataset with all the overlapping demographics.

Several papers have been released on the subject of facial detection and recognition in real-time recordings. It requires speed, which often comes at the cost of precision. Halawa et al.

(2019) recent report shows how the Faster R-CNN (Ren et al. 2015) algorithm is used with face detection. Once a face is detected from a recording, an image can be stored for facial recognition by a slower algorithm, which can extract features from it. Bah and Ming (2019) also showcase their facial recognition algorithm. Their method includes prepossessing the images to capture facial features better and then implementing their LBP algorithm. The authors are adamant that their algorithm is robust and accurate to use in real-life scenarios. Mileva and Burton's (2019) research demonstrates the use of facial recognition in a noisy real-life environment, such as in CCTV camera footage. Facial recognition from CCTV footage is certainly an arduous task and prone to errors. (Mileva and Burton 2019). More research should be conducted with facial recognition to determine the pitfalls and empirically test the algorithms that will be in use, for example, in law enforcement.

If no more regulation is adopted around the world, the market for facial recognition continues to grow and is expected to hit 7 billion USD by the year 2024. That would present more than double the market value growth in just five years since the market value is indicated to be 3,2 billion USD in 2019. (Markets and Markets). The market predictions are estimating that facial recognition would hit over 11 billion USD by 2026 (Bannister 2019).

As the use of such technologies grows and with the malicious intent that the users of the technology might possess, the privacy of one's movements is at stake. The proper use of the material recorded through CCTV cameras can be hard to enforce. Therefore, the proposed model could be attached to a route planner with street view images to gather a database of cameras. The users could be supplied with route options with the least amount of CCTV cameras. Hidden cameras would be an issue to this method, but at least in the EU, complying with the GDPR should, in theory, dramatically reduce the use of hidden camera systems.

## 2.4 (Cyber-)Attacks against CCTV cameras

The proposed model could also be used by security auditors and other professionals in locating indoor and outdoor CCTV cameras during the penetration testing of the client organization's security infrastructure. CCTV cameras are becoming more complex, and hence their attack surface is ever-growing (Costin 2016), as they delve into the IoT-space. A report from

2019 (Philippou 2019) indicates that already about 60% of CCTV camera installations are network-based. These cameras have vulnerabilities, and there are multiple ways of attacking them (Park and Jun 2011). Therefore, CCTV cameras should be properly secured and tested.

Costin (2016) addressed many of these attack vectors in his paper about CCTV security. He established that there are several non-networks related visual layer attacks that CCTV cameras can be vulnerable to. These types of attacks include controlling the camera through images such as QR-codes. After gaining access, obscure data exfiltration may also be possible. (Costin 2016). Following attack taxonomy against CCTV cameras, some works followed. In one example, the data exfiltration/infiltration attacks were implemented and demonstrated for IR and router/HDD LEDs (Guri, Bykhovsky, and Elovici 2017; Guri, Zadov, Daidakulov, et al. 2017; Guri, Zadov, Atias, et al. 2017). In another example, the "secret knock" is used to communicate with an infected imaging-capable device covertly (Mowery et al. 2014).

Networked cameras bring a whole new attack vector to the equation. At the time of writing in February 2020, Shodan.io (*Shodan*) produced over 240000 results with just a camera search term. These are networked cameras (security, webcam, etc.) that are connected to the internet and have a web interface for control. The most obvious problem with this is that default credentials are being used far too often, proposing an easy entry to the control interface. An example of the problem in recent times is the Mirai botnet from 2016. Mirai managed over 500000 devices, and these were harvested to use by leveraging default credentials of internet-connected IoT-devices (Kambourakis, Kolias, and Stavrou 2017).

## 2.5 Enumeration of CCTV cameras

On a grander scale, the proposed model could be used with mapping vehicles to estimate better the amount of CCTV camera installation in a region or a state. A good example would be Google StreetView, which has a busy updating schedule (*Street View Photos Come from Two Sources, Google and Our Contributors.*). If a model like proposed would be implemented into the StreetView system, the enumeration of camera installations could be far more accurate. For example, the estimated 770 million camera installations worldwide are calculated from shipment volumes with data dating back to 2004 (Philippou 2019). While the data is undoubtedly valid, many installations will exceed the estimated lifetime of the cameras and are in use far beyond what is expected. The drawback of the object detection method is dummy camera installations and hidden cameras, which will skew the results.

# 3 OVERVIEW OF COMPUTER VISION

Object detection and classification are valued use cases for deep and machine learning technologies. Object classification is typically referred to as assigning a class to a single object in the image. Object detection classifies multiple objects from a single image and adds object localization. (Brownlee 2019a). Object detection can be conveyed into three distinct steps; region selection, feature extraction, and classification (Zhao et al. 2018).

This chapter builds the background knowledge for the artifact (detection model) creation, and it promotes the arguments that the artifact could be created. The process for choosing proper tools and methods for the design and development of the artifact is overviewed as well.

## 3.1 Object detection

Informative region selection is handled differently depending on the architecture of the algorithm being used. A simple way would be the sliding window method, in which you scan the whole image, but this creates undesirable computational overhead, and nowadays, more efficient methods are employed instead. (Zhao et al. 2018). Recent detectors can be divided into anchor and anchor-free detectors based on their region proposal method (S. Zhang et al. 2019).

Features are essential for recognizing objects with computer vision instead of raw data since it is making the classification easier, and the objects are usually encoded much better (Lienhart and Maydt 2002). The problems lie in the diversity of the situations the objects appear. Lighting conditions, backgrounds, and object category appearances differ greatly. Feature extraction is different with classic object detection frameworks and their modern equivalents. With traditional machine learning frameworks, these features are manually extracted. Exemplar features are SIFT (Lowe 2004), Haar-like (Lienhart and Maydt 2002) and HOG (Dalal and Triggs 2005). (Zhao et al. 2018). For years, constructing feature-extractors consisted of high-skill manual labor and careful consideration to represent raw data as a suitable artifact or a feature vector (LeCun, Bengio, and Hinton 2015). Modern frameworks use deep convolutional neural networks to extract features (Zhao et al. 2018). The appeal is that the features are learned and not constructed by humans. The convolutional neural networks are multi-level learning methods that, through the layering process, construct high-level abstract features from raw input. Numerous non-linear layers in conjunction enable the network to separate the most intricate features and make the network impervious to variations of changes in the object background or lighting condition. (LeCun, Bengio, and Hinton 2015).

The extracted features need to be implicated to a class. Traditionally, classifiers such as Supported Vector Machine (Cortes and Vapnik 1995), AdaBoost (Freund and Schapire 1997) and Deformable Part-based Model (Felzenszwalb et al. 2010) were used. Most modern detectors prefer the use of softmax as a classifier since Girshick's Fast R-CNN (2015) popularized it (Qi, Wang, and Liu 2017).

## 3.2 Feature extraction before deep neural networks

Before deep neural networks took over the feature extraction field, manually created features were used. In this chapter, a few accomplished works are presented.

### 3.2.1 Scale-invariant feature transform (SIFT)

Scale-invariant feature transform (SIFT) (Lowe 2004) focused on four distinct stages to elicit the features from images. The first stage is scale-space extrema detection in which the image is searched for locations with all scales. Potential points of interest are recorded regardless of their scale or orientation. The second stage is used to locate keypoints, which means going through the candidate locations, determining the scale and location of the objects, and selecting the stable locations. The third stage determines one or more orientations for each keypoint location finalizing the properties. Keypoint descriptor finalizes the feature extraction by making them less susceptible to changes in shape or light conditions. SIFT populates the image densely with feature locations. A 500x500 pixel image should produce about 2000 stable features. (Lowe 2004).

### 3.2.2 Histogram of Oriented Gradient (HOG)

The idea behind Histogram of Oriented Gradient (HOG) (Dalal and Triggs 2005) is that edge directions and the distribution of local intensity gradients can define object appearance and shapes quite well even though exact knowledge about the gradients and edges is lacking. The implementation involves distributing the image into spatial regions and acquiring a local one-dimensional histogram of gradient directions or edge positions from each region. The combination of the histograms is the depiction. For better compliance with varying light conditions, a contrast normalization is preferred. The detection windows are laid densely, and they are overlapping to produce the best results. (Dalal and Triggs 2005).

### 3.2.3 Haar-like Features

Haar-like features (Lienhart and Maydt 2002) could be interpreted similarly as a simple kernel in convolutional neural networks except they are not learned, but hand-crafted. Mainly Haar-like features can mostly detect edges, lines, and other simple shapes. The main effective use case for them was face detection. The limitation of such features compared to CNN's is the limited amount of them that is feasible to use at any time, therefore making them subjectable to accuracy hindrances due to changes in the object's appearance. As the features are hand-crafted, they don't require training. These features can be used with fewer computations as well, therefore making them quick. (Lienhart and Maydt 2002). Haar-like features were heavily used with OpenCV for face detection purposes (see Viraktamath, Katti, and Kulkarni 2013; Xianghua Fan et al. 2012; Kadir et al. 2014). One of the most famous model was created by Viola and Jones (2004), which achieved reasonable results despite having the deficiencies of manual features and a shallow network. (Zhao et al. 2018).

## 3.3 Operation of a Convolutional Neural Network

To understand the methods behind convolutional neural networks, an understanding of digital image is needed. A typical digital image is an array of pixel values. If the image is in RGB color, it could be interpreted with three arrays (or channels), one for each of the primary colors; red (R), green (G), and blue (B). Therefore, a color image the size of 480 by 480 pixels is an array the size of 480x480x3. For an 8-bit image, each of these array elements has a value from 0 to 255. This value represents the intensity of the basic color in question at that pixel. (Furht, Akar, and Andrews 2018).

Although convolutional neural networks have been around since the late 1980s, their ascension to the top of the computer vision scene happened in 2012 when AlexNet (Krizhevsky, Sutskever, and Hinton 2017) considerably improved on all the other methods at the time in ImageNet LSVRC-2010 (Russakovsky et al. 2014; ILSVRC 2010) contest (LeCun, Bengio, and Hinton 2015). Convolutional neural networks are especially suited for object detection; the idea being that we go through the image looking for patterns and shapes or 'features' is much more productive than measuring distances between pixels. With this approach, the scale of the object in the picture can vary significantly. (Murphy 2016). State-of-the-art object detection is done using deep convolutional neural networks with supervised training. Convolutional neural networks operate on the feedforward principle, meaning that the information flow is from the input to the output only. (Rawat and Wang 2017).

Convolutional neural networks use several layer types such as convolutional layer, pooling layer and fully connected (FC) layer. The composition of layers is called the network architecture, and it can vary significantly. Convolutional layers are used to elicit features from the images. The first layer in CNN is always a convolutional layer. The purpose of these layers is to reduce the image to make it easier to process while maintaining the features of it. (Nielsen 2015). Figure 1 represents the basic concept of convolutional neural networks.

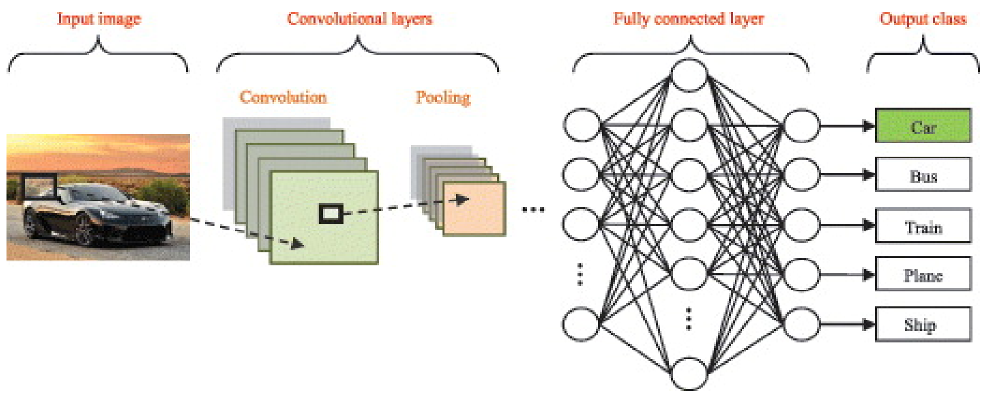

Figure 1. "CNN image classification pipeline" by (Rawat and Wang 2017) is licensed under CC BY 4.0.

### 3.3.1 Advantage of CNNs in object detection

Several advantages of deep convolutional neural networks have made them the top-performing architecture in object detection. Zhao et al. (2018) explain several of these advantages. Deeper networks provide expressive capabilities far exceeding shallower networks. CNN's provide increased training capacity; therefore, the input data could be higher-dimensional with increased capability to look at problems with different viewpoints. CNN architecture enables the use of various related tasks to be performed simultaneously, for example, object localization and classification. CNN's can learn, through multi-stage structured training, the high-level features from the dataset automatically. Hidden factors from the input data can be located through the nonlinearity the activation maps are constructed. (Zhao et al. 2018).

### 3.3.2 Convolution in feature extraction

With the first convolutional layer, the input to the network is the image. The features of it are being captured into activation maps (feature maps). The activation maps are matrices that contain the outputs from filters (kernels). These filters are smaller fixed-size matrices. They contain the weights of the network. To capture the features, a receptive field of the same size as the filter is being slid (convolved) along the image from the top left to the bottom right with a horizontal and vertical step size that is called the stride. Each element of the receptive field is being multiplied element-wise with the filter and then summed to produce a single number. This number can be altered with a bias. For an RGB image, all the layers of the image have a different filter. The outputs of them are summed to produce an element for the activation map. As all the possible receptive field locations are exhausted, we end up with a one-depth squashed convoluted activation map. For example, a 32x32 pixel RGB color image with a 5x5x3 kernel and a stride of 1 produces a 28x28x1 activation map matrix, because within that image, there are 784 locations for a 5x5 receptive field. With 1x1 convolution, the output size would be equal to the input size. There can be several feature maps in each convolutional layer with varying filters. In inference, receiving big numbers from the computation means the likeness of the filter, and the input is bigger. Training the network means updating the weights in the filters to extract the features that represent our object the best. With every layer, the output would shrink due to the convolution. To prevent this kernel shrinkage, zero padding can be inserted to the activation map to retain the size. (Nielsen 2015). Figures 2 and 3 show the starting and the second positions for convolution with a 5x5 receptive field with stride 1.

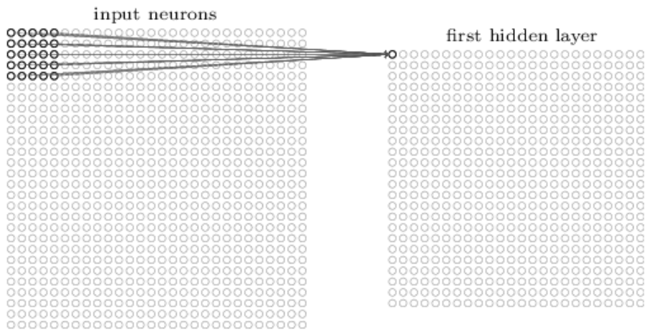

Figure 2. Receptive field and kernel, image from Neural networks and deep learning by (Nielsen 2015), licensed under CC BY-NC 3.0.

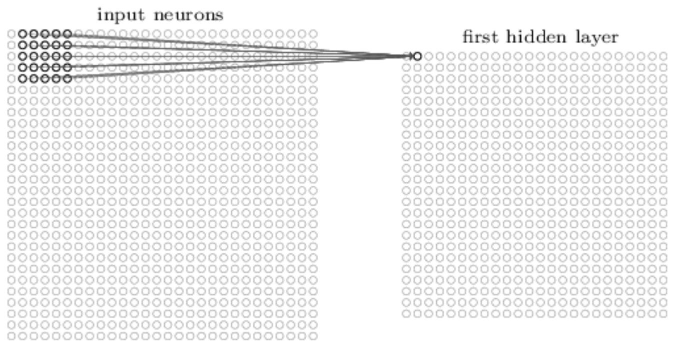

Figure 3. Receptive field and kernel with stride 1, image from Neural networks and deep learning by (Nielsen 2015), licensed under CC BY-NC 3.0.

Several kernels can be used to preserve the spatial dimensions of the image better, and in turnm, it would increase the depth of the activation map. The first convolutional layer is responsible for detecting low-level features such as curves, edges, and lines through simplistic non-linear components. (LeCun, Bengio, and Hinton 2015). To extract higher-level features such as hands, noses etc., a lot more layers are needed, which each adds more lower-level features together. These further convolutional layers take as input the output from the previous map. (Nielsen 2015).

### 3.3.3 Pooling

Pooling layers are needed for reducing the parameter count that is necessary to describe the activation maps. They also help reduce the computations required for training the network, therefore making the process faster. (Murphy 2016). This is why pooling is often called subsampling or downsampling (Rey). The operation could be described as merging similar features to reduce their amount (LeCun, Bengio, and Hinton 2015). There are a few pooling options such as max, average, and sum (Rey). Even though max pooling has an intensity increasing effect on the pixels, it's usually the best performing option. Max pooling layers could be dropped in favor of larger strides. (Murphy 2016).

Max pooling operation involves taking a kernel window from the rectified activation map. For example, we could have a 2x2 window and a stride of 2. We create a new map and take the largest numbers within that window to add to it. With a window size of 2x2 and stride of 2, no overlapping of matrix elements occurs. This effect makes the activation map more tolerant to distortions in the images and also helps to reduce the scale variance problems with object detection. (Rey).

### 3.3.4 Activation functions

Real-world data is non-linear in general. In object detection, the relation between an image label and the image data is not linear; therefore, the activation function step is crucial. Without activation functions, the neural networks would only be capable of handling linear classifications. Different activation functions are used for different layer types. Rectified Linear Unit (ReLU), shown in equation 3.1, is used to add non-linearity as convolution is a linear process. ReLU also replaces the negative pixel values in the activation map with zero. Other non-linear functions, such as sigmoid (equation 3.2) or tanh (equation 3.3), can be used as well. Still, ReLU has gained its popularity due to its properties as a vanishing gradient problem mitigator and decreased training time. A common activation function in today's CNN's is leaky ReLU. Instead of producing zeros from negative inputs, leaky ReLU outputs a negative value, but it is scaling less significantly than the positive values. Leaky ReLU (equation 3.4) avoids getting stuck in the zero-output region that sometimes happens with regular ReLUs. (Murphy 2016).

$$f(x) = max(0, x) \tag{3.1}$$

$$f(x) = \frac{1}{1+e^{-x}} \tag{3.2}$$

$$f(x) = tanh(x) \tag{3.3}$$

$$f(x) = max(-0.1x, x) \tag{3.4}$$

The output layer of an object classification CNN is a fully-connected layer. The output layer has as many neurons as there are classes. The output element represents the probability of the object in question to the object class that the element represents. This probability is calculated for all objects. (LeCun, Bengio, and Hinton 2015). This approach uses the softmax activation function (equation 3.5) as oppose to a hard activation, where classification is binary (Nielsen 2015).

$$\sigma(z)_i = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}} \text{ for } i = 1, ..., K \text{ and } z = (z_1, ..., z_K) \in \mathbb{R}^K \tag{3.5}$$

### 3.3.5 Loss functions - the supervisory components

Training a convolutional neural network involves taking a batch of images and feeding them through the whole network. This operation is often referred to as a forward pass or forward propagation. The output from the last layer goes to the loss function. Object detectors use multi-task loss functions (Girshick 2015) as the loss needs to be computed differently for localization and classification. There can be other losses for architecture-specific tasks such as CenterMask (Lee and Park 2019) has for their center-ness branch. The losses from different tasks are summed to compute the total loss. Localization loss functions vary between architectures as it is dependent on the type of localization method (Zhao et al. 2018). Loss functions produce the error between the prediction and the truth (Qi, Wang, and Liu 2017). To make the network accurate, loss needs to be minimized, which is done by updating the weights of the network. (Nielsen 2015).

As the softmax function is used commonly with CNN's for classification (Rawat and Wang 2017), an easy way to calculate the classification loss is to combine it with cross-entropy to create softmax loss (equation 3.6). This approach is often referred to as the softmax classifier, which includes the score and the loss functions (Qi, Wang, and Liu 2017). Loss results are used in the training of the network. (Rawat and Wang 2017). Although softmax loss is the de facto loss function, other proposed loss functions are being researched, such as L2-SVM Loss (Tang 2013). (Rawat and Wang 2017).

$$L = \frac{1}{N}\sum_i L_i = \frac{1}{N}\sum_i -log\left(\frac{e^{f_{yi}}}{\Sigma_j e^{f_j}}\right), \tag{3.6}$$

where $N$ is the number of training data and $f_j$ is the j-th element of the class scores $f$ and $j \in [1,K]$ as $K$ is the number of classes.

Localization part of the multi-task loss has the most variance with current object detectors. For example, Faster R-CNN (Ren et al. 2015) designates their localization loss as the subtraction of the predicted and ground-truth coordinates as the parameter in the smooth L1 function (equation 3.7). This operation is only done for the positive predictions (Ren et al. 2015).

$$smoothL_1(x) = \begin{cases} 0.5x^2 & \text{if } |x| < 1 \\ |x| - 0.5 & \text{otherwise,} \end{cases} \tag{3.7}$$

Focal loss is a function that was introduced by Lin, Goyal, et al. (2017). Back then, two-stage detectors were far superior in terms of accuracy. Focal loss, in part, bridged the gap. Focal loss is a new type of classification loss function. Essentially it is a dynamically scaling cross-entropy loss. The scaling factor of the function declines in conjunction with the increased confidence with the correct class. Focal loss addresses the imbalance that easily classified negative samples inflict. These easy negatives tend to dominate the gradient and constitute most of the loss. Focal loss down-weights the easy negative samples and hence targets the hard negative samples. The focusing parameter causes the tune to shift. (T.-Y. Lin et al. 2017). Focal loss is defined as equation 3.8. Focal loss is adopted to object detection architectures such as EfficientDet (Tan, Pang, and Le 2019).

$$FL(p_t) = -(1 - p_t)^{\gamma} log(p_t), \tag{3.8}$$

where the term $-(1 - p_t)^{\gamma}$ is the modulating factor with a tunable parameter $\gamma$ and $p_t$ is the estimated probability for the class. (T.-Y. Lin et al. 2017).

### 3.3.6 Learning through backpropagation

After the loss is calculated, backward pass (backward propagation) is needed to determine the demand for the weight update. Weights are adjusted in conjunction with the calculated weight gradient vector (derivative), which indicates how the error would change with small adjustments to the specific weight. The outcome is that the weight is updated in the opposite direction of the gradient. The equation can run through the whole network from the output to the input. After this, the weights can be individually adjusted. (LeCun, Bengio, and Hinton 2015). Weight adjustment (equation 3.9) needs to be computed by taking the derivative over the current weight, multiplying it with the learning rate, and subtracting the value from the current weight. The learning rate is a predetermined value of the CNN architecture, which determines how drastic moves can be made in the attempt to converge to the optimal weights. (Nielsen 2015). The weights can be random at first, but then the training will take more effort, and a bigger learning rate is needed. Transfer-learning is a process where a network is pre-trained to detect objects with datasets such as ImageNet (Russakovsky et al. 2014), but afterwards the network is fine-tuned to detect specific objects, which are relevant to the use case. (Shin et al. 2016).

$$W_{new} = W_{initial} - \sigma_{lr} \frac{dL}{dW_{new}}, \tag{3.9}$$

where the term $\frac{dL}{dW_{new}}$ is the gradient and $\sigma_{lr}$ is the predefined learning rate.

A common procedure for the gradient computation is Stochastic Gradient Descent (SGD). It creates averages over a few examples from the training dataset instead of the dataset as a whole and adjusts the weight accordingly. The estimates of a single set are noisy; therefore, many small sets are being processed until the loss is not decreasing any more. The process overall arrives into a good set of weights rather quickly. (LeCun, Bengio, and Hinton 2015).

Sometimes the gradient can become unstable or stagnant with some layers in the network, which means that the learning happens unevenly between the layers. This phenomenon is called the Exploding/Vanishing Gradient Problem (EVGP). It was first studied by Hochreiter (1991). (Nielsen 2015). EVGP can occur with the weight update operation when the derivative for a few trainable parameters is vast and insignificant for others. The increment of change with the weight is either quite small and insignificant or so large that the precision is lost. It is a fundamental issue with gradient-based optimizations. (Hanin 2018). If the EVGP can be handled through activation function selection and normalization, the depth of the network should be increased as it greatly increases the level of learned features and overall precision (Kaiming He et al. 2015). State-of-the-art object detectors on the top of leaderboards on (*Papers With Code*) are all trained on very deep backbones.

### 3.3.7 Overfitting and regularization

Overfitting is a phenomenon in which the network is learning too specific features from the training dataset, which in turn means that the network is not generalizing well. This process leads to overconfidence and detection errors in the validation. The network learns complicated mappings that are the results of sampling noise, and they exist only in the training data, but they are not good specific generalizing features. (Rawat and Wang 2017). Regularization is a way of preventing overfitting. (Murphy 2016). Dropout is one way of regularization. It can be used to prevent overfitting, especially in fully-connected layers (K. He et al. 2015). Dropout means that certain weights of certain layers are dropped from the training process, and their original weights are preserved during the process (Murphy 2016). The weight preserving depends on the type of dropout layer used as there are several, such as regular, fast, and spatial dropout layer among others (Rawat and Wang 2017).

### 3.3.8 Batch normalization

Batch normalization was introduced in 2015 by Ioffe and Szegedy (2015). It is a method for reducing the internal covariate shift, which is a phenomenon that occurs as the layer's inputs change while training as the parameters of the supplying layer change. This is detrimental to the training speed as learning rates need to be lowered, and certain rigor needs to be addressed with parameter initialization. BN performs normalization, which is simply put a non-linear transformation applied to activations (Rawat and Wang 2017) for every mini-batch in training; it also acts as a regularizer phasing out in some cases the need for dropouts. (Ioffe and Szegedy 2015). Ioffe (Ioffe 2017) extended the BN idea with batch renormalization, which replaces the model output dependency from the mini-batches to the individual samples taken throughout the interference and training processes. This aids performance, especially with small mini-batches (Ioffe 2017).

### 3.3.9 Data augmentation

In object detector training, usually, more data equals better performance, but it is not always feasible to get enough data per class. However, the frameworks often create subsamples of the dataset. The process is called data augmentation. Many ways of creating these subsamples exist, such as flipping, mirroring, skewing, and adjusting the brightness of the images. (Murphy 2016). The idea is to increase the training dataset, prevent overfitting, and increase the identification with varied samples of the object in different light conditions, scale, and position (Guo and Gould 2015).

## 3.4 Contemporary object detection

Today's object detection architectures are commonly classified into two categories; region proposal-based frameworks and regression/classification-based frameworks. Although, as pointed out earlier, the classification could also be drawn to anchor-based and anchor-free detectors based on their region proposal method (S. Zhang et al. 2019). Region proposal-based networks (RPN) use a two-step method. First, they scan the whole scenario and then focus on the interesting regions of the scenario proposed. (Zhao et al. 2018). The second stage consists of performing classification on these candidate locations for objects (T.-Y. Lin et al. 2017). Region-convoluted neural networks (R-CNN) is an architecture from this category. It uses a selective search approach to gather proposals from an image. It then utilizes CNN based feature extraction for the regions to establish a final feature representation for the object in the location. The regions entering the feature extractor are warped or cropped to a fixed size as the fully-connected layers require it. Problems occur when the image is warped too much, or the object is fully or partly in the cropped region. (Zhao et al. 2018). To combat this, He, Zhang, et al. (2015) introduced a new approach, which implemented spatial pyramid matching (SPM) (Lazebnik, Schmid, and Ponce 2006) into a CNN architecture. The authors called it SPP-net for spatial pyramid pooling. The SPP layer, which is located on top of the last convolutional layer, is responsible for pooling the features and generating fixed-length outputs, which can be fed into the fully-connected layers. This method avoids the possible information loss that could occur with conventional cropping or warping of the input. (K. He et al. 2015). Further improvements to this architecture have been introduced, such as Girshick's Fast R-CNN (Girshick 2015) and Ren et al. Faster R-CNN (Ren et al. 2015).

Regression/classification-based frameworks have a different approach to object detection. Instead of several interacting stages, regression/classification-based frameworks rely on one-step mapping from pixels to classification and bounding box coordinates. This can reduce the time spent on inference, and in turn, they are more useful in time-concerning use cases such as detection from video. (Zhao et al. 2018). One-stage detectors generally propose far more object locations than their two-stage counterparts due to the single sweep method, but only a handful contain objects. They tend to suffer from class imbalances due to the domination of easily classified background examples (T.-Y. Lin et al. 2017). You only look once (YOLO) (Redmon et al. 2016) , Single Shot Multibox Detector (SSD) (Liu et al. 2016), RetinaNet (T.-Y. Lin et al. 2017) and their variants can be recognized as the first prominent regression/classification frameworks in object detection.

YOLO was first introduced in 2016 by Redmon et al. (2016). Its speed was tremendous, which meant it was certainly capable of applying detection real-time on video with low latencies. Its architecture was built to see the entire image while detection and training commenced making it encode contextual information about the object, unlike methods like Fast R-CNN. This also means that YOLO generalizes object features well. Although overall accuracy was traded for speed. (Redmon et al. 2016). The current iteration of YOLO is version 3 introduced in 2018 by Redmon and Farhadi (2018). It featured tweaks to the network and a bigger architecture design overall. Accuracy was increased in favor of a slight reduction in speed. (Redmon and Farhadi 2018).

Liu et al. (2016) proposed their take on single deep neural network object detection called SSD in September 2016. It features a default set of boundary boxes of different sizes and scales over activation maps. With detection, it calculates scores for objects per each default box and adjusts its location accordingly. In its time, SSD outperformed Faster R-CNN model (Ren et al. 2015), while remaining easy to train and implement to other projects. (Liu et al. 2016).

RetinaNet (T.-Y. Lin et al. 2017) was a ground-breaking single-stage detector. It outperformed prominent two-stage detectors, while still retaining speed. It featured a ResNet (Kaiming He et al. 2015) and FPN (T.-Y. Lin et al. 2016) backbone to extract the features and two sub-networks to handle classification and the regression of the bounding boxes. The gist behind RetinaNet was the Focal Loss function, which works as a dynamically scaled cross-entropy loss. It closes on zero as the confidence in the proper class increases. The authors were determined that the class imbalance is the reason why one-stage detectors weren't as accurate as the two-stage detectors, and the focal loss would close this gap or even render one-stage detectors as state-of-the-art. (T.-Y. Lin et al. 2017).

### 3.4.1 Backbone networks

Backbone networks are the first section of a modern object detection network. They are basic feature extractors that have been proven with previous work with the fully-connected layers taken out. Many contemporary object detectors use a well-known backbone network of sorts, and the developers concentrate on improving the localization and classification branches. Different types of backbones can be utilized depending on the emphasis on needed qualities. Deeper backbones such as ResNet (Kaiming He et al. 2015), ResNetXT (Xie et al. 2017) and EfficientNet (Tan and Le 2019) are popular among the competition networks, but lighter and more efficient backbones such as MobileNet (Howard et al. 2017) are especially used when resources are limited. (Jiao et al. 2019). Larger backbones tend to generate higher accuracy, but the tradeoff is the deterioration of inference speeds (T.-Y. Lin et al. 2017). Requirements for high accuracy real-time video object detection push the requirements for the design of the backbones to the limit (Jiao et al. 2019).

### 3.4.2 Anchors

To localize multiple objects from an image, several object detection algorithms, such as YOLOv3 (Redmon and Farhadi 2018), RetinaNet (T.-Y. Lin et al. 2017), SDD (Liu et al. 2016) among others, use predefined anchor boxes. These boxes can be regarded as predefined proposal boxes that are calculated from the training dataset. They are essentially differently sized object samples. (Tian et al. 2019). Anchor boxes only have height and width. In general operation, input images are divided into a grid, whose size is equal to the final activation map. Let's say that our image is 608x608 pixels, and our stride is 32; we get an activation map with dimensions of 19x19. If that is our final map, we divide the image into a grid of 19x19 cells. Each cell can be the center of many different shapes of bounding boxes, which depends on the setup of the network. Each of the cells determines the probabilities for each class of which background is one. The cell is assigned a class, which is the maximum probability from all of the anchors in that location. The center cell is responsible for the final prediction. To have aptitude towards different scales for objects, several different grid sizes can also be established. (Redmon and Farhadi 2018). To adjust the prediction to the object location, anchor boxes are used as a reference point. The anchor boxes and the prediction boxes are checked against each other for Intersection-over-Union (IoU). Intersection-over-union is a metric, which is calculated as 3.10 and, in essence, declares the percentage of overlap between the predicted bounding box and the ground-truth bounding box. (Rezatofighi et al. 2019). The placement of the final prediction is determined by the anchor box offset with the prediction. This adjustment is called bounding box regression. (A. Zhang et al., no date).

$$\mathrm{IoU} = \frac{\text{area of overlap}}{\text{area of union}} \tag{3.10}$$

### 3.4.3 Anchor-free

Some recent works (see Duan et al. 2019; Kong et al. 2019; Law et al. 2019; Tian et al. 2019; Zhou, Zhuo, and Krähenbühl 2019; Zhu, He, and Savvides 2019) have opposed the outlook that anchors are the present and the future of object detection. Although the idea is not new (see Huang et al. 2015; Yu et al. 2016), it has recently gained a lot more traction. The implementations on how the predicted bounding boxes are generated and properly localized differ, but the basic idea of not using pre-defined anchors persists.

Among the different implementations, Law and Deng (2019) propose CornerNet. It detects the bounding boxes as a pair of keypoints using a single convolutional neural network. The points detected are top-left and bottom-right of the object. This is accomplished by generating a heatmap of the corners and embedding a vector of each corner to match the corner pairs. The network is trained to detect kindred corners. (Law et al. 2019). CornetNet achieves admirable results on MS COCO (T. Lin et al. 2014) test-dev dataset (Law et al. 2019) , but the vector embedding algorithm adds complexity to the architecture (Tian et al. 2019).

ExtermeNet (Zhou, Zhuo, and Krähenbühl 2019) has a similar bottom-up approach to object detection. Instead of locating corners, ExtremeNet locates the center of the object alongside the most left, right, top, and bottom points of the objects. A geometry-based method is used to group the points; the four extremes are grouped only if a center detection with proper confidence is also found. ExtremeNet provides great results on MS COCO (T. Lin et al. 2014), but the network is quite slow with a testing rate of 3,1 FPS. (Zhou, Zhuo, and Krähenbühl 2019).

Redmon et al. (2016) had an idea introduced in YOLOv1, where bounding boxes are predicted near the centers (anchor points) of an object instead of regressing anchor boxes. This approach introduces low recall rates as only the points near the centers are regarded. Tian et al. (2019) took this approach and broadened the detection to include all of the points within the ground-truth bounding boxes. At first, predictions are made per-pixel basis. This approach has been previously experimented by Long et al. (2014). Next, the predictions are improved by dealing with overlapping bounding boxes and with the use of multi-level prediction. In turn, a 4D vector and a class label are predicted in the feature map level. This is done for each spatial location. This method introduces low-quality bounding boxes, which are suppressed in FCOS (Tian et al. 2019) with a new center-ness branch. It is a single-layer branch that essentially represents the normalized separation of the center of the object to the prediction location. Center-ness is multiplied with the classification score to produce the final score, hopefully eliminating the low-quality predictions. According to Tian et al. (2019), center-ness is the key to the algorithm's success.

### 3.4.4 Object detection through edge-based computing

To achieve real-time object detection in real-world applications, it is not always an option to send image data to a datacenter for inference. Edge-based computing could be a solution to this problem, but large networks are not efficient enough to run on resource-constrained devices. Lightweight networks do not achieve state-of-the-art performance, but they are still producing excellent results (see f.ex Lee and Park 2019; Howard et al. 2017). There are many use cases for them, such as smart cities and self-driving vehicles.

Nagaraj et. al (2017) successfully trained and operated a modified YOLO (Redmon et al. 2016) detector on a NVIDIA Jetson TX2 (NVIDIA Developer 2017) - module in 2017. More recently, in October 2019, Barry et al. (2019) achieved 70-times faster inference speeds than Tiny-YOLOv3 (Redmon and Farhadi 2018) on their xYOLO variant running on Raspberry Pi 3B (*Teach, Learn, and Make with Raspberry Pi – Raspberry Pi*). With the xYOLO, Barry et al. (2019) heavily reduced the input image size and the number of filters. They also employed XNOR-layers (Rastegari et al. 2016), which simplify the computations on the convolutions with binary operations instead of floats (2019). In their testing, the authors recorded about a third lower mean average precision scores, but as previously mentioned, with significantly lower inference times. The model was running on 9,66 FPS on the Raspberry Pi 3B. (Barry et al. 2019).

## 3.5 Frameworks

Object detection architectures are built on common frameworks, which contain a slew of building blocks for state-of-the-art object detectors. This chapter goes through a brief overview of common frameworks.

### 3.5.1 OpenCV

Open Source Computer Vision Library (OpenCV) (*OpenCV*) is a software library for machine learning, especially for computer vision. OpenCV is written in C++ and contains more than 2500 algorithms. It has features for object detection and recognition, tools for image and video processing, and many other functions. (*OpenCV*).

For object detection purposes, OpenCV includes a variety of manually configured feature extractors such as SIFT (Lowe 2004) (Suarez and Leeuwesteijn 2014). OpenCV was used with manual feature detectors to typically feed pre-trained cascade detectors (see Suarez and Leeuwesteijn 2014; Guennouni, Ahaitouf, and Mansouri 2014; Abaya et al. 2014). It includes modules for deep learning frameworks such as TensorFlow (Abadi et al. 2016) and Torch/PyTorch (Paszke et al. 2019). As OpenCV is not used for deep neural network training, these modules allow it to utilize pre-trained deep convolutional neural network models in its ecosystem. (Rosebrock 2017). For the state-of-the-art object detection detectors such as YOLACT++ (Bolya et al. 2019), Cascade R-CNN (Cai and Vasconcelos 2018), and for Detectron2 (Wu et al. 2020) framework, OpenCV is mostly used for creating visualizations. In some cases, it is also adopted for a few tasks with dataset augmentation purposes as it is in EfficientDet (Tan, Pang, and Le 2019) and YOLACT++ (Bolya et al. 2019).

### 3.5.2 Tensorflow and Keras

Tensorflow (Abadi et al. 2016) is Google's open-source software for dataflow programming. Its symbolic math library is ideal for machine learning applications. Tensorflow is flexible and scalable; thus, the computations can be utilized in a variety of systems from mobile phones to supercomputer clusters practically unchanged. Google's first attempt at building a scalable training and inference system was created after the Google Brain project initialization in 2011, it was called DistBelief, and it was implemented in a variety of Google and other Alphabet company projects including Google Search and Maps. Tensorflow was created based on the experience they gathered with DistBelief. (Abadi et al. 2016).

Tensorflow framework contributes both high- and low-level APIs. With it, the user can operate with a single machine or utilize distributed implementations with fault tolerance for resource-intensive processes. (Abadi et al. 2016). Distributed execution is imperative for object detection with large datasets as the system resource requirements are high, and the time spent training can be lowered with increasing hardware capacity.

Tensorboard is a visualization tool for computation graphs. For object detection framework purposes, it is a valuable tool for visualizing the training events such as loss graphs evolution per epoch or iteration. Simple visualization algorithms can produce graph visualizations that are cluttered and unreadable due to the size and complex nature of the models that Tensorflow is used for. Tensorboard addresses these issues by arranging nodes into blocks at a higher level, and also it separates high-degree nodes. The process will allow the user to focus on the core sections of the graphs. (Abadi et al. 2016).

Keras is a high-level API that runs on top of Tensorflow. Keras is written in Python, and its core principles are modularity, minimalism, and extensibility. It was intended to make developing deep learning models quick and easy for research and production purposes. (Brownlee 2016). Keras API simplifies the use of Tensorflow backend to enable common deep learning tasks for the average developer. As Keras implements the use of backends such as Tensorflow, it enables the use of GPU resources for deep learning; thus, introducing much quicker training times. (Brownlee 2019b).

### 3.5.3 PyTorch

The vision behind PyTorch (Paszke et al. 2019) is a machine learning library that has high usability while maintaining admirable performance. PyTorch is a regular Python program with full support for GPUs, and it gives the user full control over it. (Paszke et al. 2019). PyTorch is open source software, and it serves as a Torch engine front-end. Torch is originally written in Lua, and it handles the gradient computations and mathematical function defining (Ketkar 2017). Although PyTorch is a Python program, its core functions are written in C++ for performance reasons. PyTorch is developed primarily by the researchers at Facebook's AI Research lab (FAIR). PyTorch separates its control flow, which contains loops and program branches and its tensor data flows. (Paszke et al. 2019). By searching for state-of-the-art object detection frameworks, one can come to the conclusion that most of them are either built on or ported to the PyTorch framework.

### 3.5.4 Detectron2

Detectron2 (Wu et al. 2020) is the second edition of FAIR's popular open-source object detection platform; Detectron. It's a complete rewrite, and it originated from the successful maskrcnn-benchmark (Massa and Girshick 2020) project. FAIR conveys that the modularity of the framework grants the users the flexibility and extensibility to train and use state-of-the-art object detection algorithms with various system configurations from a single GPU to multi-GPU clusters. (Wu et al. 2020).

Detectron2 (Wu et al. 2020) is written in Python and implemented in PyTorch. Its modularity means that the projects, which use Detectron2 as a base, can separate their code cleanly and customize the Detectron2 modules as they wish. Detectron2 is intended to be a tool for building cutting-edge technology in the field of computer vision. (Wu et al. 2020). Although Detectron2 is a modular and easy to use framework, FAIR claims great training speeds observed with their Mask-RCNN model compared to previous frameworks (*Benchmarks — Detectron2 0.1.1 Documentation*). Detectron2 features decent documentation for the use of the framework and implementation of custom modules (*Detectron 2.0.1.1 Documentation*).

### 3.5.5 Darknet

Darknet is a testing and training framework for computer vision models created by Joseph Redmon (2016). It's an open-source framework intended for fast detection and features easy to use tools. It's written in C and CUDA. YOLO (Redmon et al. 2016) real-time object detectors run natively on Darknet. Backbones such as ResNet's (Kaiming He et al. 2015) can be used as well (Redmon 2016). As the prominent detector on Darknet is YOLOv3 (Redmon and Farhadi 2018), the speed of the detection is a fundamental design choice made by Redmon.

## 3.6 Detector architectures

This chapter deepens the understanding of the specific three object detection architectures chosen to be tested for this thesis.

### 3.6.1 FCOS and CenterMask2

Fully Convolutional One-Stage Object Detection (FCOS) is an anchor free object detection algorithm presented by Tian et al. (2019). The idea is to get rid of the issues that anchor boxes propose to the algorithms. The authors disclose four drawbacks to the anchor box method. The performance of the algorithms is dependent on the amount, size, and aspect ratio of the anchors, making the tuning of the parameter's paramount to the performance. Proper handling of the anchor parameters is not always helpful in dealing with objects with large shape and size variations. In turn, this could prevent proper generalization of the algorithm. Densely placed anchor boxes cause the number of negative samples to be high compared to the positive samples during training. It creates an imbalance between samples. For each anchor box, there is also a need for computations to resolve the IoU against the ground-truth bounding boxes stated in the dataset. (Tian et al. 2019)

FCOS uses Feature Pyramid Networks (FPN) (T.-Y. Lin et al. 2016) architecture, which builds high-level, semantically strong feature maps (Tian et al. 2019). FPN takes an arbitrary size input image and yields in full-convolutional fashion feature maps at proportional sizes and at multiple levels. In turn, high-level feature maps can be obtained with multiple scales with the same spatial size. (T.-Y. Lin et al. 2016). FCOS produces five levels of feature maps P3..P7 of which P3 to P5 are produced by the backbone. P6 and P7 are derived from P5 and P6 respectively by adding one more convolutional layer with a stride of 2 to the network for each map. (Tian et al. 2019).

One of the chosen network architectures for this research was CenterMask (Lee and Park 2019) or more specifically CenterMask2 (Lee and Park 2020), an upgraded implementation for the Detectron2 framework. CenterMask is a segmentation-based architecture built upon the FCOS architecture. In essence, CenterMask is a three-part architecture consisting of VoVNet2 backbone for feature extraction, the detection head lifted from FCOS, and the mask head.

CenterMask adds a spatial attention-guided mask (SAG-Mask) to the FCOS (Tian et al. 2019) detector. The mask branch is similar to the one in Mask R-CNN (Kaiming He et al. 2017), which is a two-stage detector. The spatial attention module (SAM) helps the SAG-Mask branch to center its attention to meaningful pixels and aids in the disqualification of the unimportant ones. FCOS uses the feature maps to predict the bounding boxes, SAG-Mask takes the boxes from FCOS detector as an input and predicts segmentation masks for each region of interest (RoI). Centermask uses an adaptive RoI assignment function that varies with the scale of the input. (Lee and Park 2019). The adaptive RoI function is defined as follows:

$$k = k_{max} - log_2 \frac{A_{input}}{A_{RoI}}. \tag{3.11}$$

$k_{max}$ is an integer that is defined by the final feature map in the backbone. $A_{input}$ and $A_{RoI}$ are determined by the area of the input image and the RoI. The function determines the feature map to be selected $(P_k)$ affecting the detected scale of the object. Centermask limits the available Pk's from P3 to P5 as it showed the best performance in their ablation study. (Lee and Park 2019).

As a backbone, Centermask introduces an improved version of the VoVNet (Lee et al. 2019) backbone, called VoVNetV2. It features One-shot Aggregation (OSA). (Lee and Park 2019). OSA is a feature map aggregation method developed by Lee et al. (2019) for VoVNet. With it, all the convolution layers in the OSA-module have two-way connections, one connection is for the sequential layer, and the other is for a cumulative sweep to aggregate all the features to the final feature map, hence the name; one-shot aggregation. This causes the input for the convolution layers to be constant, which in turn ceases the need for additional dimension reduction between layers. The computational efficiency of the network is therefore increased. OSA-modules in VoVNet prioritizes high-level features by increasing the output channel count in the later layers. (Lee et al. 2019).

OSA-modules tend to slightly decrease accuracy compared to the dense aggregation in which features are aggregated between each layer. This happens due to the disturbance that the gradient endures during backpropagation. (Lee and Park 2019). The degradation of the accuracy may not always be due to overfitting. Instead, it could be an optimization issue. (Kaiming He et al. 2015). Centermask tries to overcome this by adding a residual connection to each of the OSA-modules. This deepens VoVNet and increases performance. (Lee and Park 2019). The idea of residual connections is from ResNet (Kaiming He et al. 2015). They are identity mapping shortcuts over the layers, and they help to preserve the learned features, especially in the case where there is nothing new to learn, by blocking diminishing transformations. (Kaiming He et al. 2015) In the Centermask architecture, these connections facilitate the optimizations in the OSA-modules (Lee and Park 2019).

Another CenterMask feature is effective Squeeze-and-Excitation for improving yet more on the VoVNet (Lee et al. 2019) backbone. (Lee and Park 2019). Squeeze-and-Excitation blocks are originally proposed by Hu et al. (2017). SE block is a channel attention module, which is meant to increase the representational power of the feature extraction network by using adaptive feature recalibration in channel-level. The interdependency of the channels in the convolutional features are unequivocally modelled to achieve discriminatory actions towards informative and less informative features. For residual networks such as ResNet and VoVNet, the SE blocks are placed into the network. The block is placed before the identity map summation. At first, a global average pooling layer is used to extract statistics from the channels. The authors call this operation *squeeze*. The operation the authors refer to as *excitation* is where the adaptive recalibration is formed. Excitation operations involve learning the non-linear synergies and non-mutually-exclusive relationships between channels. To achieve this, the information gathered in *squeeze* is fed to a gating mechanism, which is formed by a ReLU function with two bottlenecking FC layers. Finally, the block output is formed by rescaling the gate output with sigmoid activation. (Hu et al. 2017). Authors of CenterMask point out that the dimension reduction, done by the FC layers in the block to reduce complexity, is causing channel information loss. The first FC layer reduces the channel dimensionality by the reduction rate $(C/r)$, while the second FC layer grows the dimensionality back to the original $C$. In order to maintain this information, Lee and Park propose an effective SE block (eSE), which cuts out the other FC layer. eSE is applied to the OSA-modules in VoVNet2. (Lee and Park 2019).

To achieve real-time processing (>30 FPS), Lee and Park propose CenterMask-Lite. This architecture cuts down on all of the three parts of CenterMask to increase the speed with the cost of accuracy. The FPN channels are reduced from 256 to 128 in the backbone. The backbone used is a more lightweight VovNetV2-19 with 4 OSA modules instead of 5 in the larger VoVnetV2-39 and 57. The mask head layers and channels are also reduced from 4 and 256 to 2 and 128, respectively. The authors demonstrate results that indicate that the Lite detector with VoVNetV2-19 backbone achieves over three times the speed of the flagship CenterMask detector with VovNetV2-99 backbone. (Lee and Park 2019).

### 3.6.2 TridentNet

The base of TridentNet (Li et al. 2019), the two-stage Faster R-CNN, has a typical convolutional feature extractor backbone that leads into a region proposal network (RPN), which searches the anchors for objects, and predicts the bounds and objectness score for them. (Li et al. 2019) RPN does not assign a class to the prediction. To prevent duplicate prediction, RPN uses a method called Non-maximum Suppression (NMS). NMS sorts the proposals by score, makes decisions based on scores and the IoU to determine the best predictions. After the RPN, the network needs to extract features from the proposals to determine the class. Feeding all the proposals through a full pre-trained network, however, would be slow. Faster R-CNN tries to mitigate the speed loss by sharing the convolutional features with the proposal generator (RPN) and the class detector (Fast R-CNN). (Ren et al. 2015). For each proposal, a specific length feature vector is derived from the feature map. The vectors are fed through a series of FC layers to determine the class probabilities and a refined bounding box position (Girshick 2015).

Li et al. (2019) approached their network project through the problem of scale variation of the detected objects. They propose a scale-aware Trident Networks, which aims to create scale-specific activation maps. The network uses multi-branch architecture with equal transformation parameters, but different kernel receptive fields. Besides, a scale-aware training process is applied to specialize every branch with a specific scale of objects. (Li et al. 2019).

TridentNet uses 50-layer and 101-layer ResNet (Kaiming He et al. 2015) variants as a backbone. TridentNet can also benefit from methods such as deformable convolution (Dai et al. 2017) in its backbone. Deformable convolution offsets the standard grid-like sampling. This deformation is free form and should aid, especially with the encoding of high-level features in the later layers as the objects scale and contortion varies between locations. (Dai et al. 2017). The results from Li et al. (2019) show that deformable backbone increases mean average precision with the MS COCO (T. Lin et al. 2014) dataset.

Efforts to increase detection with varied scales of objects have been around since the days of manual feature extraction (Dalal and Triggs 2005). To introduce a similar methodology to deep neural networks, Singh and Davis (2018) proposed a training scheme called Scale Normalization for Image Pyramids (SNIP) (Singh and Davis 2018). In essence, SNIP uses an image pyramid type of feature extraction where input images are taken with different scales, and features are extracted independently from each scale. Image pyramids take full advantage of the representational capability of the model and produce features equally to all scales, but the inefficiency will lead to longer inference times. Previously mentioned feature pyramids (T.-Y. Lin et al. 2016) are more efficient. Still, Li et al. (2019) are adamant that because the features are extracted from different layers of the FPN backbone, they are achieved with different parameters making them an inconsistent alternative. The risk of overfitting could increase, as well. (Li et al. 2019).

TridentNet uses a different approach to extract features of different scales. Using dilated convolutions (Yu and Koltun 2015) different branches share the network structure producing comparable features, but with different scales due to the receptive field variance. Dilated convolution is applied to the branches with varying dilation rates. This means that convolutions are performed at sampled locations, which are chosen sparsely. In turn, this increases the size of the receptive field. The receptive fields are therefore accommodated to different scales of objects. Kernel enlargement is concluded by adding zeros between filter values without increasing parameter count or making more computation. The number of zeroes to insert is dilation rate $d_s - 1$. Therefore, a 3x3 kernel would have the receptive field equivalent of $3 + 2(d_s - 1)$. This multi-branch regime could introduce a lot of new parameters as the branch count increases. TridentNet, however, introduces weight sharing to the process. As the branches are identical, except for the receptive field dilation rate, weight sharing is ideal. The same parameters are trained for different object scales. To combat scale mismatch caused by the predefined dilation rates in the multi-branch architecture, Li et al. (2019) propose scale-aware training, which selects correct ranges in the branches for each proposal and ground-truth pair. (Li et al. 2019).

### 3.6.3 ATSS

S. Zhang et al. (2019) claim that a key difference between anchor and anchor-free detectors is how they characterize their positive and negative samples. Their Adaptive Training Sample Selection or ATSS uses the object's statistical characteristics to choose the sample accordingly. In their paper, Zhang et al. (2019) leveled the playing field between the prominent detectors; anchor-based RetinaNet (T.-Y. Lin et al. 2017) and anchor-free FCOS (Tian et al. 2019) by setting key configurations similarly, therefore introducing almost similar performance (37% vs. 37,8% in MS COCO (T. Lin et al. 2014) minival subset). With the introduction of these similar configurations, S. Zhang et al. (2019) were able to determine that the differences between the two detectors boiled down to how they determine the positive and negative samples (classification) and the starting point for the bounding boxes (regression). The authors allow only one anchor box per location when using RetinaNet (T.-Y. Lin et al. 2017) as this procedure is close to the FCOS (Tian et al. 2019) methodology. Normally RetinaNet (T.-Y. Lin et al. 2017) pushes nine anchors per location. With further testing, they concluded that when the sample selection of the detectors is similar, the effects of the bounding box regression starting point for FCOS is negligible from a performance standpoint. Thus, making the way of classification an irrelevant difference between the detectors. Their tests concluded that training sample selection is an essential aspect of the training performance of any detector and, therefore, an interesting avenue for research. (S. Zhang et al. 2019).

Both anchor-based and anchor-free detectors set predetermined threshold parameters for the sample selection. For anchor-based, it is the IoU threshold, and for anchor-free, the scale range. The parameters are sensitive as they affect the performance greatly. ATSS (S. Zhang et al. 2019) works adaptively. The algorithm computes a new IoU threshold for each ground-truth box. First, it calculates the IoU ($Dg$) of the positive candidates between the ground-truth $g$. The new IoI threshold is the summation of the mean $mg$ and standard deviation $vg$ of the IoU $tg = mg + vg$. This threshold is used to divide the positive and negative samples. For RetinaNet (T.-Y. Lin et al. 2017), the best IoU's are calculated when the anchor box center is close to the center of the object. For FCOS (Tian et al. 2019) and center-ness branches, the closest anchor point to the center produces the best results. The algorithm is limiting the positive sample centers to the ground-truth box as the otherwise poor candidates could be selected, and features outside the objects could be trained. (S. Zhang et al. 2019).

The ATSS (S. Zhang et al. 2019) selection methodology improves RetinaNet (T.-Y. Lin et al. 2017) performance alongside the proposed single-box per location strategy. The lite version of ATSS is already implemented to FCOS (Tian et al. 2019). This approach recognizes all the anchor points within the object box as candidates at first and then selects nine candidates per FPN (T.-Y. Lin et al. 2016) pyramid level as true candidates per ground-truth. The full version of ATSS turns the anchor point into an anchor box to designate the samples, but regresses the positive samples from the anchor point like the original FCOS (Tian et al. 2019). ATSS improves on the performance of original FCOS (Tian et al. 2019). (S. Zhang et al. 2019).

## 3.7 Public datasets and competition

For the results from the general object detection architectures to be comparable, common datasets for training, evaluation, and testing are needed. Several examples for mainstream public datasets exist such as Microsoft Common Objects in Context (MS COCO) (T. Lin et al. 2014; *COCO - Common Objects in Context*) , Pascal VOC (Everingham et al. 2010; *The PASCAL Visual Object Classes*), and ImageNet (Russakovsky et al. 2014; *ImageNet Object Localization Challenge*).

The metrics to evaluate the classification performance of machine learning models are commonly precision, recall, and F1. For object classification, precision is calculated, as shown in equation 3.12. Precision, in essence, is the percentage of correct predictions across all positive predictions. It is a useful metric, but does lack information about the false negative predictions in which real objects are falsely predicted as background. Recall (equation

3.13), on the other hand, provides information about positive predictions that were missed. F-measure is a combination of both; precision and recall. F1 is a variant of F-measure, which handles precision and recall in an equal manner, and it is calculated as seen in equation 3.14. It showcases properties of precision and recall in a single figure. (Brownlee 2020).

$$Precision = \frac{TruePositives}{TruePositives + FalsePositives} \tag{3.12}$$

$$Recall = \frac{TruePositives}{TruePositives + FalseNegatives} \tag{3.13}$$

$$F1 = \frac{2 * Precision * Recall}{Precision + Recall} \tag{3.14}$$

### 3.7.1 Microsoft Common Objects in Context (MS COCO)

Lin et al. (2014) proposed a new dataset for general object detection in 2015 called Common Objects in Context. The most recent update MS COCO 2017 has a fully annotated training dataset containing 118000 images and a 5000 image validation dataset. Also, a 41000 image testing dataset is available. For semi-supervised learning, MS COCO offers 120000 unlabeled images. Images contain several objects increasing the number of object instances in the datasets. The number of classes for state-of-the-art object detector testing with the MS COCO2017 dataset is 80. The metadata from the images is stored in JSON format, and it follows a simple field in a series arrangement. Several properties can be saved in the JSON file with the data requirements such as image name, height, width, and annotation information such as labels and points coordinates. All data for all images in the dataset is stored in a single file. (*COCO - Common Objects in Context*).

MS COCO had four challenges for 2019. The challenges were Detection, Keypoints, Stuff, and Panoptic. MS COCO discloses that the Detection challenge was intended to push the state-of-the-art general object detection forward. The challenge entries were only intended for instance segmentation output, but bounding box detection was also possible. Keypoint detection task was limited to localizing persons and detecting their keypoints. As MS COCO train, val, and test datasets contain over 200000 images with over 250000 instances of persons with keypoint annotation, the amount of available data is surely enough. Stuff category includes the background objects found on the MS COCO dataset, for example, sky, pathways, and grass. The motivation is that 66% of the pixels in the MS COCO dataset belong to this category. Stuff challenge is also limited to segmentation due to the nature of the detection targets. Finally, the Panoptic challenge focuses on both; objects and stuff. The idea is that the detectors create scene segmentations of the images, while motivation being the use cases in augmented reality and autonomous driving. Final leaderboards can be found in the challenge websites, respectively. (*COCO - Common Objects in Context*).

To evaluate the performance of a detector in the Detection category, MS COCO employs 12 characterizing metrics, as seen in Table 3. Average precision (AP) with MS COCO is, in essence, mean average precision (mAP) that is taking the precision average across all the classes. With MS COCO, localization accuracy is built into their precision metrics. Their standard measurement nowadays is average precision and average recall (AR), with the average taken from 10 IoU thresholds from 0.5 to 0.95 with 0.05 interval. AP over 0.5 and 0.75 IoU are also disclosed. AP across scales takes into account the area of pixels in the segmentation mask or bounding box. Small equates to less than $32^2$ pixels, the medium is between $32^2$ and $96^2$, and large is more than $92^2$. Average recall also has the three metrics, while the differentiating factor being the amount of detection per image; 1, 10, and 100. AR across scales is comparable to AP across scales. (*COCO - Common Objects in Context*).

### 3.7.2 Pascal VOC

The Pascal VOC (Everingham et al. 2010) project ran object detection challenges between 2005 and 2012 (*The PASCAL Visual Object Classes*). For the 2012 detection challenge, the dataset contained 20 classes with training and validation datasets consisting of over 11000 images with over 27000 instances (Everingham and Winn 2012) There were several similar classes to increase difficulty (Jiao et al. 2019). Similar to MS COCO, there were detection and segmentation tasks, but also Action Classification and Person Layout Taster tasks with respective datasets. VOC formats its data in individual XML files for each image. The file contains similar information as MS COCO format such as annotation coordinates, image height, width, name, and class. (Everingham and Winn 2012).

Action Classification task involved identifying ten different actions of persons from the images. Categories such as walking and running were implemented. The Person Layout Taster task again involved persons, but now the detection algorithm was intended to identify parts of a person such as heads and legs. (Everingham and Winn 2012).

The evaluation for the detection task was made using precision and recall curves whilst maintaining over 0.5 IoU. They marked difficult objects in the dataset, and these objects were ignored in the evaluation, although final results could be submitted with a separate difficult metric. (Everingham and Winn 2012).

### 3.7.3 ImageNet

ImageNet Large Scale Visual Recognition Challenge (ILSVRC) (Russakovsky et al. 2014) is an ongoing annual object category classification and detection challenge, and it has been run since 2010. It was imagined by Russakovsky et al. (2014) to follow Pascal VOC (Everingham et al. 2010) footsteps in providing a challenging dataset and hosting competitions. ImageNet has 1,2 million images for training plus 150000 images for validation and testing. ImageNet has 1000 classes, of which 200 were originally chosen for object detection challenges. (Russakovsky et al. 2014). ImageNet uses the Pascal VOC format (*ImageNet Object Localization Challenge*). Several consequential backbones have been trained on ImageNet such as ResNet (Kaiming He et al. 2015).

ImageNet originally had classification, detection and video detection challenges (ILSVRC 2010) with 1000, 200, and 30 classes to detect respectively. The most recent competition focused on the detection category (*ImageNet Object Localization Challenge*). ImageNet competition uses total error evaluation, where the average minimum errors are calculated across all images. The error is increasing if the object class is wrong or the IoU is less than 50%. (*ImageNet Object Localization Challenge*).

# 4 METHODOLOGY

The process of artifact design and development was conducted in four phases; dataset gathering, annotation, environment setup, and model training. Several iterations of this process were conducted as more knowledge of the system was received, and the initial results were analyzed. Peffers et al. (2006) define the design and development part of design science as the stage where desired functionality is determined, and the artifact created. As for the artifact in this thesis, three design goals were documented. The most important goal is that the model needs to be accurate in its predictions, especially with classification. Secondly, the model needs to be able to be further trained with dataset improvements. Thirdly, if all possible, the efficiency of the model should be good enough to be effective with real-time detection from video. At least, the model should be efficient enough to be productive to use with detection from video sources. This section is intended to answer the primary research question; how can we accurately and efficiently detect CCTV cameras from images and videos?

Before settling on the three models proposed earlier, YOLOv3 (Redmon and Farhadi 2018) was a contender to be the architecture of choice. YOLOv3 has a large community around it and even professional support (Jocher et al. 2019). With the community support in GitHub (*YOLOV3 GitHub Repositories*), there are plenty of tools to train custom datasets like ours. However, YOLOv3 didn't produce the results needed to satisfy the primary design goal in our testing, even though many avenues were experimented with. This led to the use of Detectron2 (Wu et al. 2020) and the architectures implemented within it.

## 4.1 Dataset gathering

To train an object detection model to detect CCTV cameras, a dataset of them needed to be gathered. A decision was made to classify the cameras into two distinct classes based on their shape; directed and round. Directed cameras include box and bullet (Cameron 2018) type cameras, which record a specific direction. Round cameras include dome (Cameron 2018) shaped cameras and also the cameras, which record a specific direction, but are shaped as a sphere or a dome. This design choice was made, because at this point, the need is for accurate detection of CCTV cameras from the background and not to identify specific camera types.

At first, the idea was to create a dataset with the two camera classes and two negative label classes; light poles/fixtures and background (i.e., not CCTV cameras). The images were gathered from online sources such as Flickr (*Flickr*), Google StreetView (*Street View Photos Come from Two Sources, Google and Our Contributors.*) and OpenStreetCam (*OpenStreet-Cam*). For each of the four classes, 1750 images were gathered with varying scales, light conditions, weather, pose, and shape. Although the scale varied a lot, tiny instances were not included. The sentiment was that, in our use cases, images were to be taken frequently and close enough to each other; therefore, the cameras are not missed, but the model doesn't need to detect extremely small samples. The background class consisted of objects that were present around CCTV cameras. The notion behind the light pole/fixture class was that street lights have similar shapes to the cameras, and negative labels for them could improve detection accuracy. The images were captured as rectangles with varying resolution to ensure compatibility and maximum useful resolution for the training process. A Bash script was made to split the total 7000 images into training and validation datasets. A split of 85% and 15% respectively was used.

As the work progressed, it became clear that an MS COCO (T. Lin et al. 2014) format dataset was the ideal choice. The dataset was altered to the MS COCO format, sample selection was conducted, and more images were gathered. This included more small-scale images and images with several instances for cameras. The decision was made to drop the negative label classes in favor of more background in the images with many instances. This decision was further backed up by the fact that the Detectron2 (Wu et al. 2020) framework dataloader automatically discards images without annotations (*Detectron 2.0.1.1 Documentation*). It was also observed that the negative labels could be a hindrance to the learning and possibly cause longer training times without any gains. In the end, we settled on a training dataset with 2680 images containing 1586 directed class instances and 1425 round class instances. The validation dataset contained 453 images with 320 directed class instances and 200 round class instances marking about 17% split between training and validation datasets. MS COCO format dataset made it easy to use several architectures out of the box, and it allowed instance segmentation detection as well due to polygon annotation possibility.

As for the evaluation, another separate test dataset was gathered. It contained only images from street view sources such as OpenStreetCam (*OpenStreetCam*) and Google StreetView (*Street View Photos Come from Two Sources, Google and Our Contributors.*). The scale varied in this dataset a lot less than with the training or validation datasets as there are no large samples. Still, this dataset included images without any instances of cameras as well. The testing dataset was intended to serve as an indicator of how well the model will work as a detector for specifically street-view sources as it is the primary use case proposed with future work. The testing dataset contains 186 images with 126 instances of directed class and 105 instances of the round class.

## 4.2 Dataset annotation

After the initial dataset gathering process, the annotation of the objects in the images commenced. The objects were identified from the images, and a rectangle box was drawn around them individually. The initial annotation was made using software by Tzutalin (Tzutalin 2015) called LabelImg. LabelImg is a fast way to do rectangle annotation to images in PascalVOC (Everingham et al. 2010) and YOLO format (Redmon and Farhadi 2018), but MS COCO - JSON format is not featured. Initially, the dataset with two classes and two negative classes was annotated. The annotation was done in YOLO format, which has five parameters; object-class integer and normalized x-center, y-center, height, and width of the annotated box. To train YOLOv3, all images must have a corresponding plain text file containing this information. (Redmon and Farhadi 2018).

For the MS COCO (T. Lin et al. 2014) format, a tool called Labelme by Wada (2016) was used. Labelme has drawn inspiration from the work of Russell et al. (2008) similarly labeled a web-based annotation tool called LabelMe. Labelme can annotate with polygon segments to be used in object segmentation architectures. With object segmentation, the outlines of the objects could be identified instead of a mere bounding box around the object. Labelme outputs individual JSON files for each annotated image, and it also includes Python scripts to embed the data as an MS COCO single annotation file format. (Wada 2016). The final version of the dataset was annotated with polygon shapes (Figure 4) and turned into MS COCO format compliance. The individual JSON annotation files were saved for future reference, possible annotation changes, and for easy training/validation dataset splitting changes. The testing dataset was also annotated similarly.

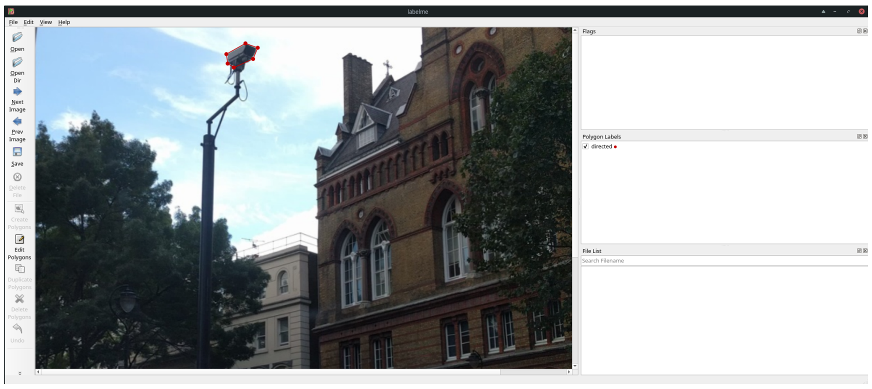

Figure 4. Polygon annotation with Labelme

## 4.3 Physical infrastructure and virtual environments

Object detector training requires a lot of system resources, especially with the larger backbones. The models were trained on the CSC Puhti (located in Kajaani) supercomputer cluster courtesy of Finnish Grid and Cloud Infrastructure (FGCI), and some testing was also conducted on the University of Jyväskylä's (JYU) own GPU supercomputer cluster. For our task, we employed four Nvidia Tesla V100 32GB GPGPUs on the Puhti cluster and four Nvidia Tesla P100 16GB GPGPUs on the JYUs cluster. Training times were recorded. The inference was concluded with a single Nvidia RTX 2060 6GB GPU as the intention was to determine the efficiency of the model with lesser hardware for further use cases as well as with the supercomputer hardware with a Tesla V100 GPGPU. Average inference times per image were recorded from both environments.

The Nvidia Teslas (*NVIDIA Tesla Supercomputing*) are used for stream processing. They are general-purpose graphics processing units (GPGPU). The V100's are based on Nvidia's Volta architecture, and they feature 5120 CUDA (Compute Unified Device Architecture) cores each. The P100's are based on the Pascal architecture with 3584 CUDA cores, respectively. CUDA is an Nvidia platform and programming model, which enables implementing general-purpose computing easily for any task. (Ebersole 2012). CUDA technology significantly improves the performance of model training to a point where CPUs are not a valid option (Wang, Wei, and Brooks 2019). The VRAM overhead of the Teslas enabled faster training speeds as the batch size of the network could be higher. Although, for this thesis, the batch sizes were not optimized.

The frameworks tested and used for this thesis were implemented in PyTorch. A Python environment needed to be set up for the clusters with several needed libraries and packages. In order to achieve the setup, a virtual environment is required. To set it up, Conda (*Conda — Conda 4.8.3 Documentation*), an open-source virtual environment, and package manager, were installed. Conda enabled us to install and use different matching versions of packages and libraries. It also isolated them to the specific virtual environment. Miniconda3 package was used as the features of the bigger Anaconda3 were not required.

The first efforts to train a model were done with Ultralytics PyTorch implementation (Jocher et al. 2019) of the prominent YOLOv3 (Redmon and Farhadi 2018) detector. Ultralytics' implementation was chosen instead of the original Darknet (Redmon 2016) framework because of the ease of modifications that comes with Python-based code and due to the support that their Github page provides. Ultralytics' YOLOv3 required several dependencies such as numpy, OpenCV, torch, matplotlib, pycocotools, tqdm, and pillow. The packages used in our environment are listed in Table 1. The package dependencies were also met. With the YOLO format dataset, the framework worked flawlessly, and modifications were easy due to the exceptional documentation and community support.

| Name | Version | Description |
|---|---|---|
| Python | 3.8.1 | Python core |
| numpy | 1.18.1 | Scientific computing package |
| OpenCV | 4.2.0.32 | Computer vision library (Section 3.5.1) |
| PyTorch | 1.4.0 | Machine learning framework (Section 3.5.3) |
| torchvision | 0.5.0 | Computer vision package |
| matplotlib | 3.1.3 | Graph visualizations |
| pycocotools | 2.0 | Tools for MS COCO |
| tqdm | 4.42.1 | Progress bar for terminal use |
| pillow | 7.0.0 | Imaging library |

Table 1. YOLOv3 environment

The framework of choice for the final selected architectures was Detectron2 (Wu et al. 2020) implemented in PyTorch. The train of thought behind this preference was that Detectron2 employs a lot of state-of-the-art architectures, and it has robust features for implementing real-world use cases. As a Python-based framework, Detectron2 is easily modifiable and has good documentation (*Detectron 2.0.1.1 Documentation*). For future work, Detectron2 can be modified to limit the framework to the features that are required for the use case in question. Thus the process can be streamlined even further and even automate the necessary steps for model use and updating.

Detectron2 (Wu et al. 2020) lists several dependencies as requirements for their framework; Python3, PyTorch with matching torchvision, OpenCV for visualization, and pycocotools. Our Miniconda3 4.8.2 environment included all the packages from Table 1 and included some extra packages, which were needed for either Detectron2 or the architectures implemented in it. These packages are listed in Table 2. Their dependencies were also installed. The packages were obtained and installed through the conda package manager and pip. Detectron2 was obtained from their Github repository (Wu et al. 2019) and built from source. Puhti cluster used Slurm workload manager (*SchedMD | Slurm Support and Development*) for their batch processing. The appropriate Slurm scripts were made for each training session. The training was limited to a single node with four Tesla GPGPUs, although the framework is capable of distributing the workload for a lot more nodes. With four Teslas, however, the training was consistent without any errors.

| Name | Version | Description |
|---|---|---|
| albumentations | 0.4.3 | Image augmentation library |
| cudatoolkit | 10.1.243 | Software for GPU programming |
| cython | 0.29.15 | C extension |
| ninja | 1.9.0 | Small build system |
| pandas | 1.0.1 | Data analysis library |
| requests | 2.23.0 | HTTP library |
| scipy | 1.4.1 | Mathematics, science, and engineering library |
| yacs | 0.1.6 | Configurations management system |
| Tensorboard | 2.1.1 | Training data capture (Section 3.5.2) |
| Detectron2 | 0.1.1 | Object detection framework (Section 3.5.4) |
| QCC | 8.3.0 | Gnu Compiler Collection loaded as a module in Puhti cluster |
| CUDA | 10.1.168 | Nvidia CUDA loaded as a module in Puhti cluster |

Table 2. Detectron2 environment

## 4.4 Detector selection and model training

Several tests were conducted with the Ultralytics YOLOv3 (Jocher et al. 2019) detector. With the proper dataset configuration in place, the config file needed a few modifications. YOLOv3 (Redmon and Farhadi 2018) features three anchors per cell; therefore, the filters in the yolo layers must be set, as shown in equation 4.1.

$$filters = (4+1+\text{num of classes})*3, \tag{4.1}$$

where 4 equals to the number of box coordinates and 1 is the maximum object confidence.

In our testing, filters were set to 21 and the number of classes to two in the config. This was done for all of the three yolo layers. The batch size was set for 64, while subdivision was one. These were superb choices for the Tesla V100 setup. Several different layer configurations were tested, such as the standard config, standard with SPP, and ResNeXT-50. The hyperparameters for the standard config were also optimized using the built-in hyperparameter evolution. The parameters were trained for 50 generations per ten epochs, and the best were selected. With our dataset, YOLOv3 never reached the precision we were looking for, and therefore, other architectures were looked into.

A robust framework around the detector was deemed a necessity. Therefore, the reasoning behind the selection of CenterMask (Lee and Park 2019), ATSS (S. Zhang et al. 2019) and TridentNet (Li et al. 2019) was their top 10 performance in PapersWithCode (*Papers With Code*) leaderboards at the time in early March 2020, Detectron2 framework implementation or compatibility with it and the differences in their architectures. Comparative evaluation between the architectures was made easy because of the consistency of the framework.

The setup for our custom dataset with Detectron2 (Wu et al. 2020) was simple. At first, the dataset needed to be registered in the main function of the training script and set in the configuration file. Secondly, the class count required to be changed from 80 to two in the configurations file. Other modifications to the configurations were made, such as the maximum iteration count and steps for the learning rate schedule. The training was commenced as transfer training, and pretrained weights were downloaded for each backbone. The final weights selected for testing were chosen by overseeing the Tensorboard training data and logs from the training from each session. The ideal weights were deemed to be the ones that converge near to the lowest losses and highest precision without overtraining as the intention is to resume the training from these weights with updates to the dataset. Detectron2 expresses training progress as iterations instead of epochs. One iteration is one batch of images being processed in the network while an epoch is the whole dataset going through the same. After the first training session with Centermask2 and 180000 iterations, it became clear that training doesn't require over 100000 iterations with this dataset, and further training was conducted with fewer iterations. The results were deteriorating after 50000 iterations, possibly due to overfitting, but with the architecture differences, this threshold was exceeded with every training session to be sure.

### 4.4.1 CenterMask2

Centermask2 (Lee and Park 2019) was trained with VoVNetV2 backbone with the V-57-eSE and V-99-eSE variants. The lightweight V-39-eSe was also tested. The maximum iterations were cut from 270000 to 180000, and the steps were lowered accordingly. The batch size was left at 16 to remain under the VRAM threshold. Although, for future training, batch size should be optimized for faster training speeds. The base learning rate was 0.01, and the smaller sides for images in multi-scale were 640, 672, 704, 736, 768, and 800 pixels. The Lite backbone takes 580 and 600-pixel size input and features the same changes. The learning schedule was also changed for the Lite to two thirds from the default.

### 4.4.2 ATSS

Two backbones were used with ATSS (S. Zhang et al. 2019); ResNet-50 (Kaiming He et al. 2015) and ResNeXt-101 (Xie et al. 2017) with multi-scale training and deformable convolutions. The latter backbone has ResNeXt cardinality set at 64 and the bottleneck width at 4d. Cardinality and bottleneck width are hyperparameters for aggregated residual transformations. They are improvements to ResNet (Kaiming He et al. 2015) backbone made for ResNeXt by Xie et al. (2017). The cardinality is the number of paths inside the ResNeXt block. Bottleneck width is the number of individual channels in the bottleneck layers. The concept behind the ResNeXt block is split-transform-aggregate. The idea is that instead of growing the network deeper or wider, increasing the cardinality will increase accuracy and keep the complexity of the network low. (Xie et al. 2017)

Maximum iterations with ATSS (S. Zhang et al. 2019) were lowered to 90000 (1x schedule), with the steps of 60000 and 80000. The base learning rate was left at 0.01. Batch size of 16 was used throughout. With the bigger backbone, multi-scale input image sizes were 640 and 800 pixels. Multi-scale testing was not implemented.

### 4.4.3 TridentNet

TridentNet (Li et al. 2019) training was conducted with the ResNet-101 (Kaiming He et al. 2015) C4 backbone. The C4 variant uses the features extracted from the ResNet's fourth stage, and the fifth serves as the RoI heads (Kaiming He et al. 2017). The training scheme was also altered for TridentNet as the 3-time configuration was not deemed necessary. The same 90000 max iterations and corresponding steps were used as in the ATSS training. The standard three-branch scheme for the TridentNet multi-branch was used. Multi-scale training was turned on with 640, 672, 704, 736, 768, and 800 pixels for the smaller side. The Base learning rate was set at 0.02, with a batch size of 16.

# 5 RESULTS

All the models were evaluated with the testing and validation datasets. As mentioned earlier, the testing dataset features images from street view maps, which correspond to the intended use for the detector model. The inference was done for a minimum of 1200 pixels (small side) images. The image size that the models were trained on (i.e., between 600 and 800 pixels), was also tested. The results for the validation datasets were slightly better with the trained image size for large objects but far worse for the small ones. With the more practically more important testing dataset, the results with the smaller image sizes were consistently worse. Therefore, the presented results are only from the larger image inference tests.

The metrics presented here can be interpreted as the evaluation activity of the design science research method. Peffers et al. (2006) denote that the research should show how well the artifact performs as a solution to the problem. Our testing dataset especially displays that. The metrics presented here exhibit comparable results of the models created for this thesis, but however, they do not tell the whole story about the performance of the detectors. The quality of the datasets and settings used affect the outcome a lot. The visual results demonstrate how the solution works for our problem, which is another design science research method - activity. Peffers et al. (2006) imply that simulation is an effective way of conducting a demonstration. This was our approach. This chapter will answer the second research question; how do the models fare against each other?

Table 3 shows the full detector and backbone combination with the training iteration count for each model that was deemed the best performing of the whole session. The rest of the tables use the abbreviations presented in Table 3.

| Abbreviation | Detectron configuration | Model iterations |
|---|---|---|
| CM2 Lite V-39 | CenterMask2 Lite with V-39-eSE-FPN backbone | 40000 |
| CM2 V-57 | CenterMask2 with V-57-eSE-FPN backbone | 40000 |
| CM2 V-99 | CenterMask2 with V-99-eSE-FPN backbone | 45000 |
| ATSS R-50 | ATSS with ResNet-50 FPN backbone | 30000 |
| ATSS X-101 | ATSS with ResNeXt-101 dcnv2 64x4d FPN backbone | 37500 |
| Trident R-101 | TridentNet with ResNet-101 C4 backbone | 45000 |

Table 3. Detector configuration and model iterations

## 5.1 Training time

Table 4 shows the average training time per iteration for the models. The time spent on training per iteration varies a lot with the configuration and architecture. For example, batch size, learning rate, training schedule, backbone architecture, and GPU model and count can affect the time spent per iteration. Therefore, Table 4 is not the definitive training speed indicator, but it is nevertheless informative data.

CenterMask2 Lite V-39 is a really fast network to train. For any significant dataset changes or for attempts at real-time detection from video, it could be an excellent choice. ATSS with ResNeXt is by far the most arduous to train. ATSS (S. Zhang et al. 2019) with the ResNet-50 (Kaiming He et al. 2015) and CenterMask2 V-57 (Lee and Park 2019) are quite fast as well while CenterMask2 V-99 (Lee and Park 2019) and TridentNet (Li et al. 2019) have intermediate training speeds.

## 5.2 Inference time

The inference times were recorded as an average between the validation and test dataset inference. They are shown in Table 4. The same configurations and setups were used; therefore, the times are comparable. The inference was done for images with the small side between 1200 and 1333 pixels.

Inference times do not present any anomalies. The lone two-stage detector, TridentNet (Li et al. 2019), is the second slowest just beating ATSS (S. Zhang et al. 2019) with the bigger backbone with the slower setup. With the V100 GPGPU setup, TridentNet is clearly the slowest. CenterMask2-Lite (Lee and Park 2019) is by far the fastest, although not surprisingly. With the large images, the inference rate is still nowhere near-real-time (> 30FPS) with either of the setups. Thus smaller images and datacenter hardware should be used for video detection. The bigger CenterMask V-99 (Lee and Park 2019) is quite fast in comparison to the other two detectors with large backbones. The Tesla GPGPU had a considerable time reducing effect, especially with the larger backbones. This is due to the performance difference of the units as well as the VRAM difference. With more VRAM, more images could be loaded as a batch. CenterMask-Lite was also tested with a smaller image size of 600 pixels with the V100 GPGPU. The result was on average $0.035 s/image$, which is close to the real-time video detection threshold.

| Detector | Avg. training time (s) / iteration | Avg. inference time (s) / image (RTX2060) | Avg. inference time (s) / image (V100) |
|---|---|---|---|
| CM2 Lite V-39 | 0.28 | 0.079 | 0.061 |
| CM2 V-57 | 0.81 | 0.155 | 0.089 |
| CM2 V-99 | 1.22 | 0.220 | 0.109 |
| ATSS R-50 | 0.48 | 0.133 | 0.064 |
| ATSS X-101 | 2.40 | 0.470 | 0.132 |
| Trident R-101 | 1.59 | 0.375 | 0.169 |

Table 4. Detector training and inference times

## 5.3 Metrics

The models were evaluated with the pycocotools MS COCO evaluator built into Detectron2 (Wu et al. 2020). The evaluator was the same for all of the models. The metrics presented are the standard MS COCO (T. Lin et al. 2014) evaluation metrics presented earlier. The latest MS COCO metrics heavily weigh on the localization with the IoU threshold from 0.5 to 0.95. With our immediate use cases, the exact localization might not be necessary; therefore, the precision with 0.5 IoU is the metric with the most importance. Although, the metrics with IoU from 0.5 to 0.95 are used for the state-of-the-art detector evaluation; therefore, they are necessary. Table 5 displays the glossary of the metrics presented in later tables. Average recall with 100 detections is not applicable since there are no images with over ten instances per image in any of the datasets. The testing dataset does not contain any large objects; thus, that metric is omitted from those tables.

| Abbreviation | Metric |
|---|---|
| AP@0.5 | Average precision with IoU 0.5 and over |
| AP@0.5:0.95 | Average precision with IoU 0.5 to 0.95 |
| APs | Average precision IoU 0.5 to 0.95, small object area < $32^2$ |
| APm | Average precision IoU 0.5 to 0.95, medium object $32^2 <$ area $< 96^2$ |
| APl | Average precision IoU 0.5 to 0.95, large object area < $96^2$ |
| AR 1 | Average recall IoU 0.5 to 0.95, one detection per image |
| AR 10 | Average recall IoU 0.5 to 0.95, maximum of 10 detections per image |
| ARs | Average recall IoU 0.5 to 0.95, small object area < $32^2$ |
| ARm | Average recall IoU 0.5 to 0.95, medium object $32^2 <$ area $< 96^2$ |
| ARl | Average recall IoU 0.5 to 0.95, large object area < $96^2$ |

Table 5. MS COCO (T. Lin et al. 2014) metrics used for evaluation

Table 6 reveals the metrics for the testing dataset for bounding box detection. The results implicate that classification and false positives are at a great level. The AP@0.5 metric has excellent scores from about mid 80 percentile to low nineties. The localization suffers a bit; therefore, we see the metrics with IoU from 0.5 to 0,95 take a hit with 62% being the highest score on CenterMask2 (Lee and Park 2019). The metrics showcase that smaller objects seem to be an issue with all of the models in comparison to the medium ones. More instances in an image does not seem to affect the models too much.

| Detector | AP@0.5 | AP@0.5:0.95 | APs | APm | AR 1 | AR 10 | ARs | ARm |
|---|---|---|---|---|---|---|---|---|
| CM2 Lite V-39 | 85,0% | 59,1% | **61,0%** | 65,4% | **59,6%** | 69,9% | **65,6%** | 75,9% |
| CM2 V-57 | 88,5% | **62,6%** | 60,9% | **68,9%** | 59,3% | **70,2%** | 65,0% | 77,6% |
| CM2 V-99 | 87,7% | 59,4% | 59,9% | 65,1% | 57,9% | 69,1% | 62,5% | **78,4%** |
| ATSS R-50 | 77,1% | 50,9% | 49,9% | 58,7% | 53,7% | 62,8% | 55,6% | 72,9% |
| ATSS X-101 | **90,8%** | 57,5% | 57,4% | 61,9% | 57,0% | 66,6% | 60,3% | 75,3% |
| Trident R-101 | 84,6% | 53,8% | 54,4% | 57,8% | 53,5% | 61,7% | 56,5% | 69,0% |

Table 6. Results for bounding box detection with the testing dataset

Table 7 shows the segmentation detection metrics from the CenterMask2 (Lee and Park 2019) models with the testing dataset. The segmentation localization on the CenterMask2 models doesn't seem to affect the detection precision or recall too highly, except in the case of small objects.

| Detector | AP@0.5 | AP@0.5:0.95 | APs | APm | AR 1 | AR 10 | ARs | ARm |
|---|---|---|---|---|---|---|---|---|
| CM2 Lite V-39 | 86,1% | 57,2% | **55,5%** | 65,1% | **56,8%** | **67,3%** | **62,9%** | 73,3% |
| CM2 V-57 | **88,4%** | **58,6%** | 53,4% | **67,8%** | 56,3% | 66,3% | 60,5% | 74,3% |
| CM2 V-99 | 86,0% | 57,6% | 53,4% | 67,7% | 55,8% | 66,9% | 60,5% | **75,7%** |

Table 7. Results for segmentation detection with the testing dataset

Table 8 exhibits the detection metrics for the validation dataset with bounding box detection. The metrics with the validation dataset are not surprisingly better than with the testing dataset except in the case of small objects. The scores are considerably lower, with every model tested in that category. With average mean precision scores close or over 90% in the lower localization accuracy, one can certainly be satisfied with these initial results.

| Detector | AP@0.5 | AP@0.5:0.95 | APs | APm | APl | AR 1 | AR 10 | ARs | ARm | ARl |
|---|---|---|---|---|---|---|---|---|---|---|
| CM2 Lite V-39 | 88,4% | 61,1% | 50,0% | 65,3% | 59,8% | 64,6% | 71,5% | 58,0% | 73,8% | 73,3% |
| CM2 V-57 | **93,5%** | 70,3% | **57,6%** | 72,6% | 74,6% | 71,2% | 77,5% | **61,7%** | 78,7% | 82,9% |
| CM2 V-99 | 91,6% | 70,5% | **57,6%** | 72,5% | 75,0% | 71,8% | 77,7% | 61,5% | 78,5% | 84,2% |
| ATSS R-50 | 89,7% | 66,8% | 47,6% | 71,2% | 69,5% | 68,1% | 75,1% | 54,1% | 78,0% | 79,2% |
| ATSS X-101 | **93,5%** | **71,5%** | 56,5% | **74,6%** | **75,7%** | **72,1%** | **78,6%** | 60,3% | **80.5%** | **84,4%** |
| Trident R-101 | 90,7% | 67,7% | 51,5% | 69,0% | 74,9% | 71,2% | 76,2% | 54,4% | 77,9% | 83,5% |

Table 8. Results for bounding box detection with the validation dataset

Table 9 demonstrates the segmentation detection metrics from the CenterMask2 (Lee and Park 2019) models with the validation dataset. The detectors have a comparable reduction in the metrics with segmentation predictions than they did with the testing dataset.

| Detector | AP@0.5 | AP@0.5:0.95 | APs | APm | APl | AR 1 | AR 10 | ARs | ARm | ARl |
|---|---|---|---|---|---|---|---|---|---|---|
| CM2 Lite V-39 | 87,1% | 62,3% | 46,8% | 66,0% | 64,4% | 64,8% | 71,0% | 56,3% | 72,6% | 74,9% |
| CM2 V-57 | **92,2%** | 69,8% | **52,9%** | 72,1% | 75,6% | 70,5% | 75,9% | **59,4%** | 76,9% | 81,8% |
| CM2 V-99 | 90,6% | **70,4%** | 52,7% | **72,9%** | **76,7%** | **71,1%** | **76,5%** | 58,9% | **77,0%** | **84,2%** |

Table 9. Results for segmentation detection with the validation dataset

### 5.3.1 Training losses

Reasoning to the metrics can be found from the training losses. Table 10 features the training losses at the selected iteration number. FCOS (Tian et al. 2019) based detectors ATSS (S. Zhang et al. 2019), and CenterMask2 (Lee and Park 2019) both converge to really low classification losses and low localization losses, but their center-ness branch loss is oscillating through the whole training process. As the classification score and center-ness scores are multiplied in testing to filter low-quality bounding, our quite high center-ness loss will definitely affect performance. TridentNet (Li et al. 2019) shows really low losses throughout. This might indicate that the model is overfitting.

| Detector | Classification | Localization | Center-ness | Mask | Total |
|---|---|---|---|---|---|
| CM2 Lite V-39 | 0.004 | 0.030 | 0.587 | 0.04 | 0.660 |
| CM2 V-57 | 0.004 | 0.015 | 0.588 | 0.02 | 0.63 |
| CM2 V-99 | 0.003 | 0.020 | 0.588 | 0.03 | 0.64 |
| ATSS R-50 | 0.014 | 0.067 | 0.590 | - | 0.67 |
| ATSS X-101 | 0.036 | 0.050 | 0.590 | - | 0.68 |
| Trident R-101 | 0.005 | 0.007 | - | - | 0.012 |

Table 10. Training losses

## 5.4 Visual results

A tool by Jagin (Gilewski 2020) called Detectron2-pipeline was used for producing the images for the visual evaluation in the cases of Centermask2 (Lee and Park 2019), and TridentNet (Li et al. 2019). With ATSS (S. Zhang et al. 2019), their own demo script was modified for the purpose. Due to VRAM restraints, 800-pixel input sizes were used. Images are chosen from the test dataset and they are captured from OpenStreetCam (*OpenStreetCam*). CenterMask2 draws both bounding box and segmentation masks over the image. The confidence level is displayed next to the bounding box. The same weights were used as in the metrics results.

As the CenterMask2 (Lee and Park 2019) representative, the V-57 backbone version was chosen. ATSS (S. Zhang et al. 2019) results were presented with the ResNeXt-101 (Xie et al. 2017) deformable convolutions - backbone and TridentNet (Li et al. 2019) used the

ResNet-101 (Kaiming He et al. 2015) backbone. The same image samples were taken from the test dataset. In the images, CenterMask2 is on the left, ATSS in the middle and TridentNet on the right.

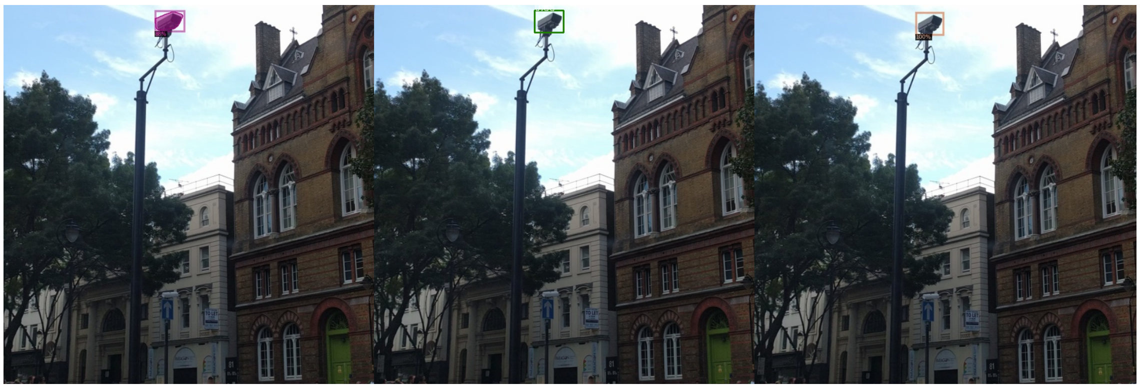

Figure 5. Visual results (left to right): CenterMask2 V-57 88%, ATSS X-101 88%, TridentNet R-101 100%

Figure 5 is an easy example of a directed class camera with no background interference. Every model detects the camera admirably, and the light fixture on the bottom of the model is correctly not flagged.

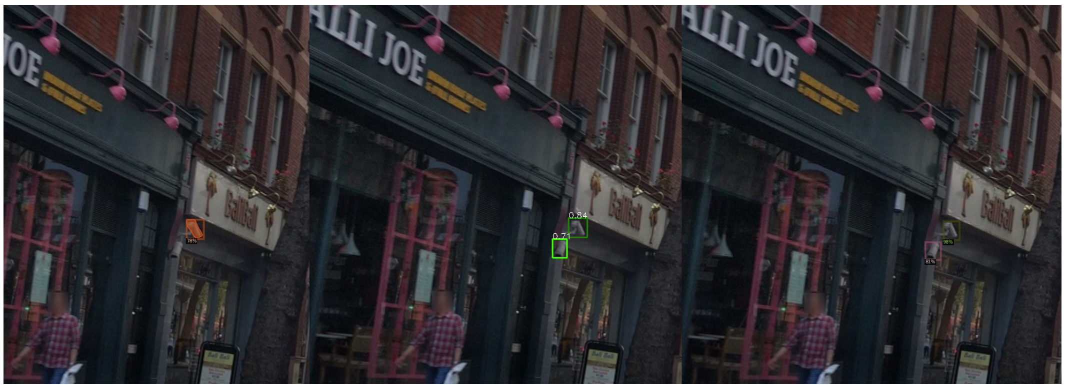

Figure 6. Visual results (left to right): CenterMask2 V-57 78% + NaN, ATSS X-101 84% + 71%, TridentNet R-101 98% + 81%

Figure 6 is a medium difficulty image with two directed class instances. The CenterMask2 (Lee and Park 2019) model doesn't recognize the lower camera, but the confidence on the upper one is acceptable. It seems that more front-facing directed cameras are needed for the dataset, although the image is quite dark as well. The other two models work better with this, and TridentNet achieves great results. Again, light fixtures are correctly not classified.

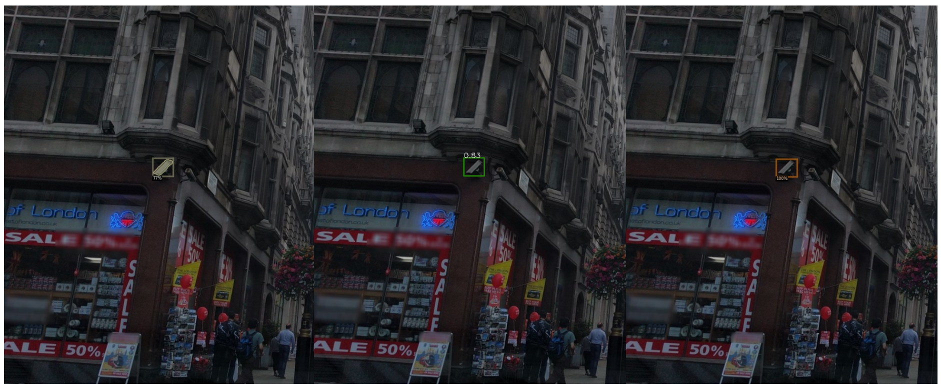

Figure 7. Visual results (left to right): CenterMask2 V-57 77%, ATSS X-101 83%, TridentNet R-101 100%

Figure 7 is another dual directed class instance image. Every model has a false negative on the back-facing camera, although the object shouldn't be that difficult to detect. The side-facing camera is detected properly.

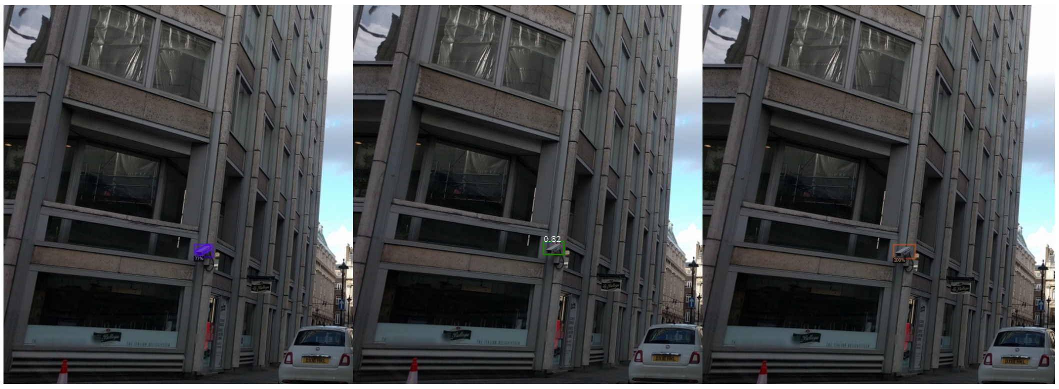

Figure 8. Visual results (left to right): CenterMask2 V-57 77%, ATSS X-101 82%, TridentNet R-101 100%

Figure 8 is an easy example of a side facing directed class instance. No issues with this one. TridentNet (Li et al. 2019) achieves 100% confidence.

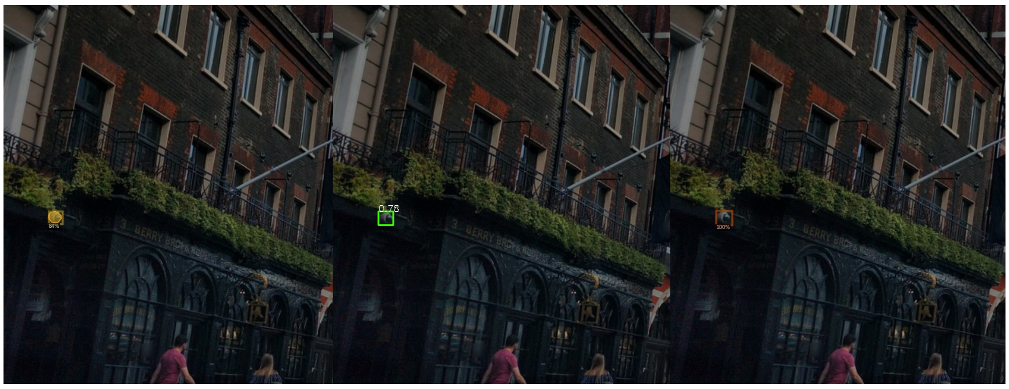

Figure 9. Visual results (left to right): CenterMask2 V-57 84%, ATSS X-101 78%, TridentNet R-101 100%

Figure 9 features a side installation of a round class instance. Every model detects the camera properly.

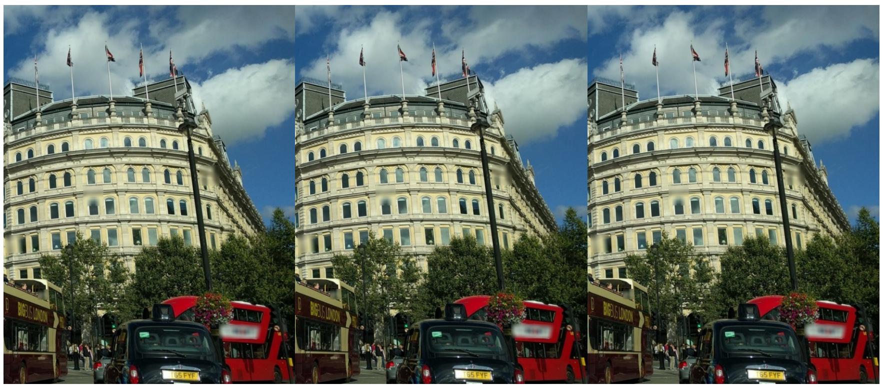

Figure 10. Visual results (left to right): CenterMask2 V-57 NaN, ATSS X-101 NaN, TridentNet R-101 NaN

Figure 10 shows a side facing directed class instance, which blends into the background. No model detects the object, and the outcome is three false negatives. The dataset requires harder instances to train with.

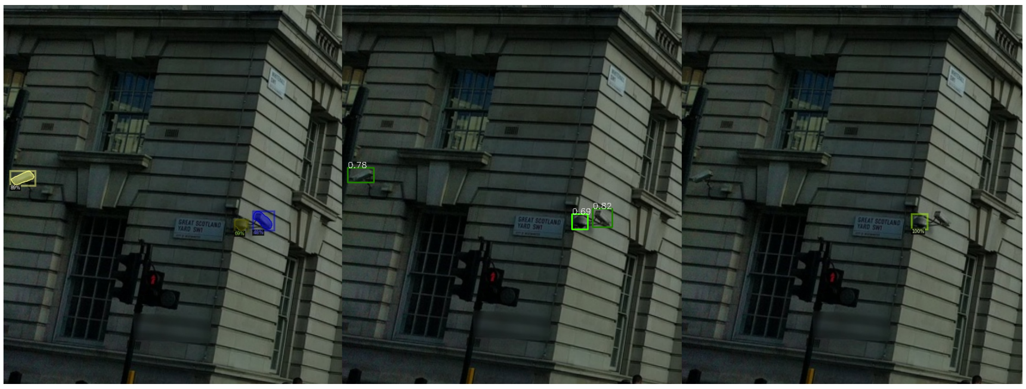
Figure 11. Visual results (left to right): CenterMask2 V-57 89% + 69% + 48%, ATSS X-101 78% + 69% + 82%, TridentNet R-101 Nan + 100% + NaN

Figure 11 has two directed class instances and one round class instance. ATSS (S. Zhang et al. 2019) achieves respectable results with CenterMask (Lee and Park 2019) not far behind with just the rightest instance having low confidence. TridentNet (Li et al. 2019) produces two false negatives with the directed class instances.

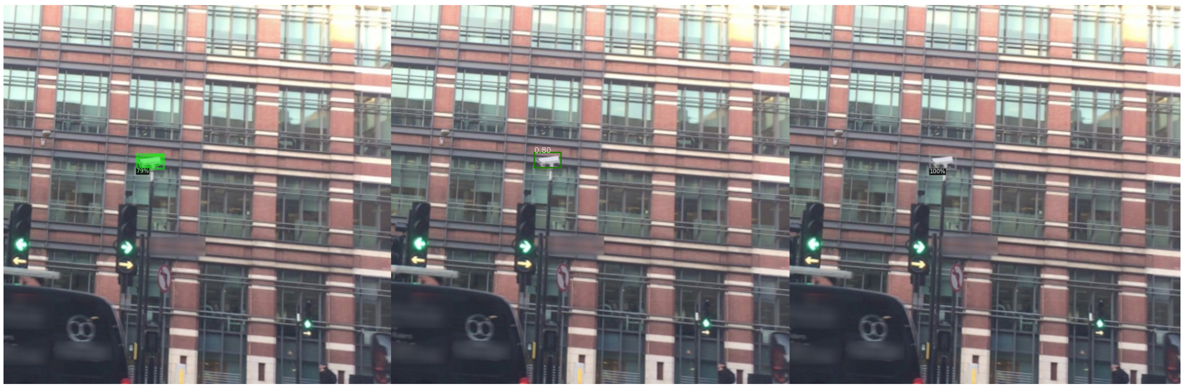
Figure 12. Visual results (left to right): CenterMask2 V-57 79%, ATSS X-101 80%, TridentNet R-101 100%

Figure 12 features a smaller scale side facing directed class instance. No issues in detection with any of the models. TridentNet (Li et al. 2019) provides yet another 100% confidence detection.

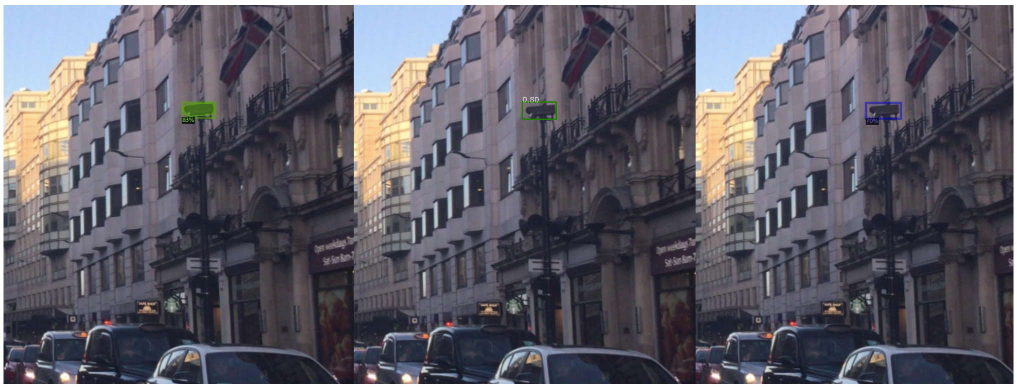

Figure 13. Visual results (left to right): CenterMask2 V-57 83%, ATSS X-101 80%, TridentNet R-101 70%

Figure 13 has yet another side facing directed class instance with lower light conditions. CenterMask (Lee and Park 2019) displays the highest confidence with 83% while TridentNet (Li et al. 2019) has the lowest with 70%.

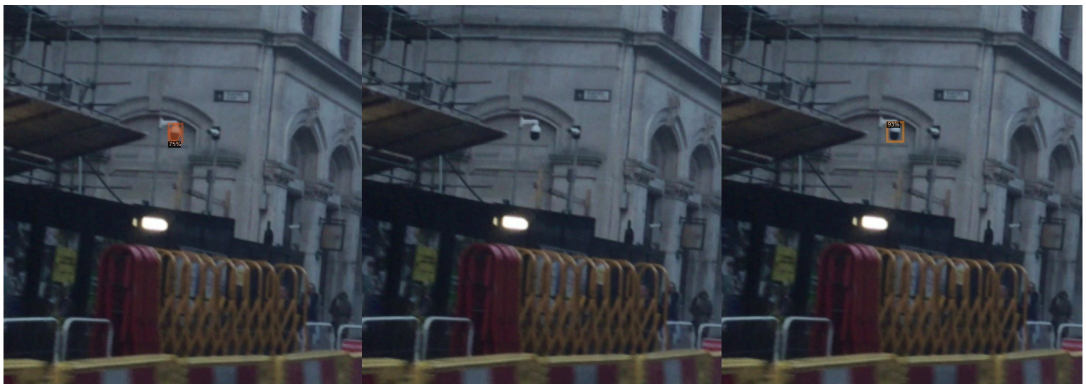

Figure 14. Visual results (left to right): CenterMask2 V-57 75%, ATSS X-101 NaN, TridentNet R-101 93%

Figure 14 showcases a rather blurry image with a round class instance on a smaller scale. ATSS (S. Zhang et al. 2019) introduces a false negative, but the other two produce decent results.

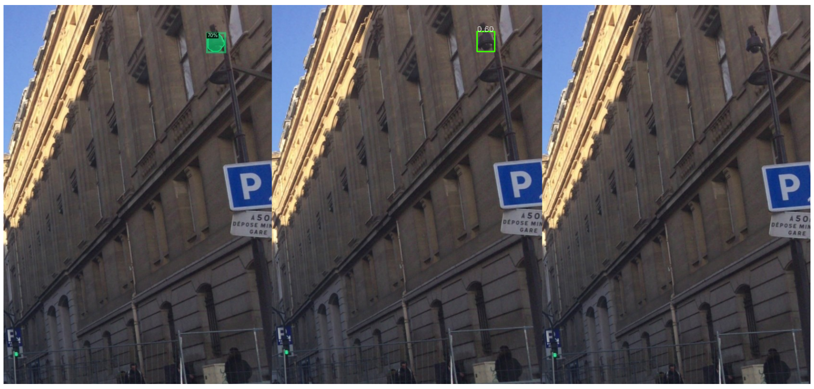

Figure 15. Visual results (left to right): CenterMask2 V-57 70%, ATSS X-101 60%, TridentNet R-101 NaN

Figure 15 features a slightly large round class instance, but yet the confidence of CenterMask (Lee and Park 2019) with 70% and ATSS (S. Zhang et al. 2019) with 60% are not great. TridentNet (Li et al. 2019) even classifies the instance as background (i.e., non-camera class). A curious sample, which should be added to the training set.

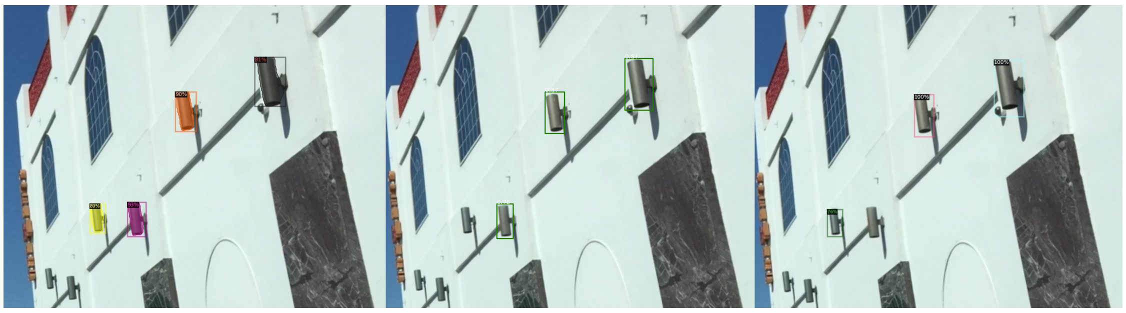

Figure 16. Visual results (left to right): CenterMask2 V-57 NaN, ATSS X-101 NaN, TridentNet R-101 NaN

Figure 16 is the worst performing sample of the whole dataset. Not one detector captures the intended side facing round class instance, and all of the detectors produce multiple false positives with the light fixtures. These types of lights need to be introduced into the training dataset as background (i.e., non-camera class).

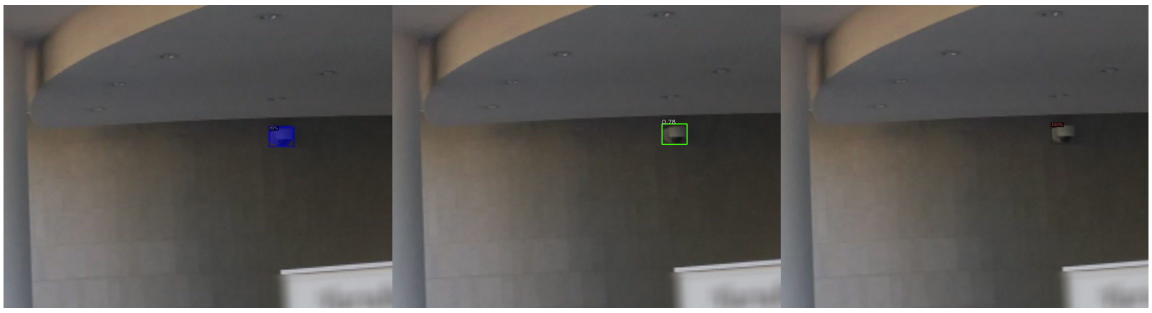

Figure 17. Visual results (left to right): CenterMask2 V-57 89%, ATSS X-101 78%, TridentNet R-101 100%

Figure 17 is a blurry example of a round class instance. There are no detection issues with this one.

### 5.4.1 Altered images

Figures 7,10, and 11 were chosen for visual testing with modified images as they contain false negatives. The idea is that by editing the images, these false negatives could be detected properly. All of the images were modified by adjusting the contrast, enabling automatic contrast editing with equalizer, changing exposure, and finally hue-saturation. All of the samples have been arranged as follows (left to right); original, contrast, equalizer, exposure, and saturation.

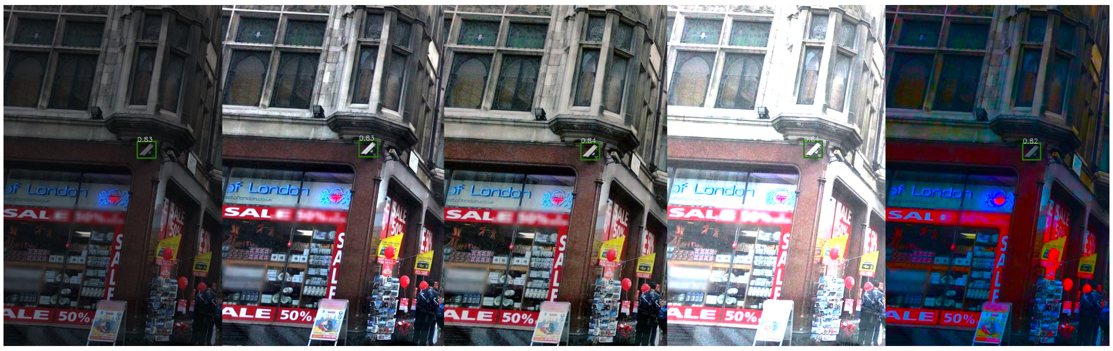

Figure 18. ATSS; Figure 7 alterations (left to right): original, contrast, equalizer, exposure, and saturation.

Figure 18 showcases the results from the first image with ATSS (S. Zhang et al. 2019) detector. None of these modifications worked with this sample. The false negative is still present, and there are no major differences with confidence of the true positive.

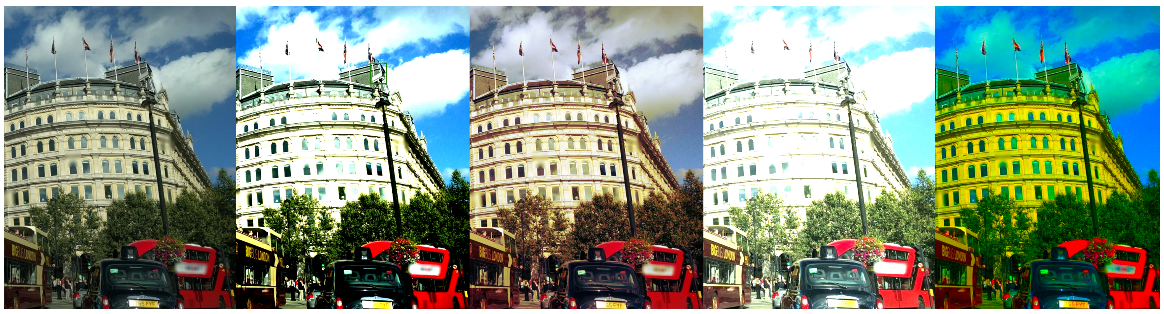

Figure 19. ATSS; Figure 10 alterations (left to right): original, contrast, equalizer, exposure, and saturation.

Figure 19 contains the results from the second image with ATSS (S. Zhang et al. 2019) detector. Increased contrast seems to bring out a true positive detection, although the confidence is not that high. Other modifications have no effect.

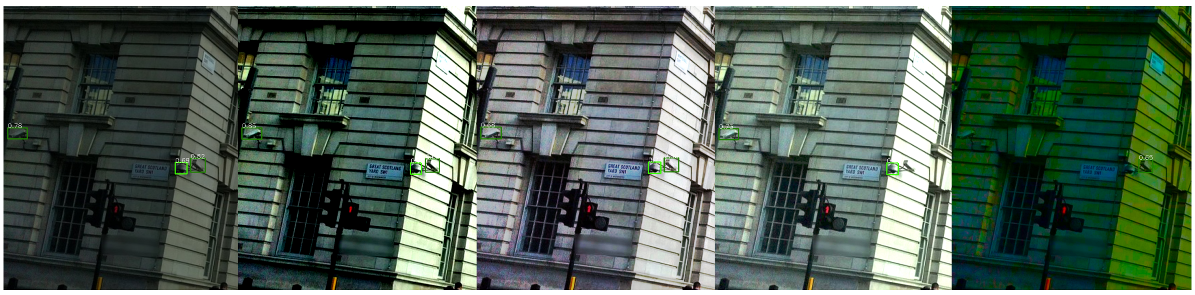

Figure 20. ATSS; Figure 11 alterations (left to right): original, contrast, equalizer, exposure, and saturation.

Figure 20 displays the results from the final image with ATSS (S. Zhang et al. 2019) detector. Contrast modifications seem to increase the confidence of the detections. Other modifications make detection worse.

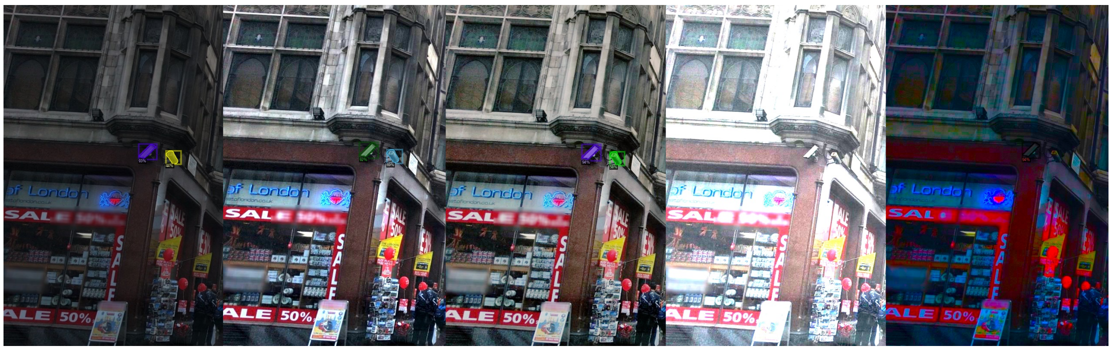
Figure 21. CenterMask; Figure 7 alterations (left to right): original, contrast, equalizer, exposure, and saturation.

Figure 21 presents the increased confidence with CenterMask (Lee and Park 2019) detection with the contrast and equalizer modifications.

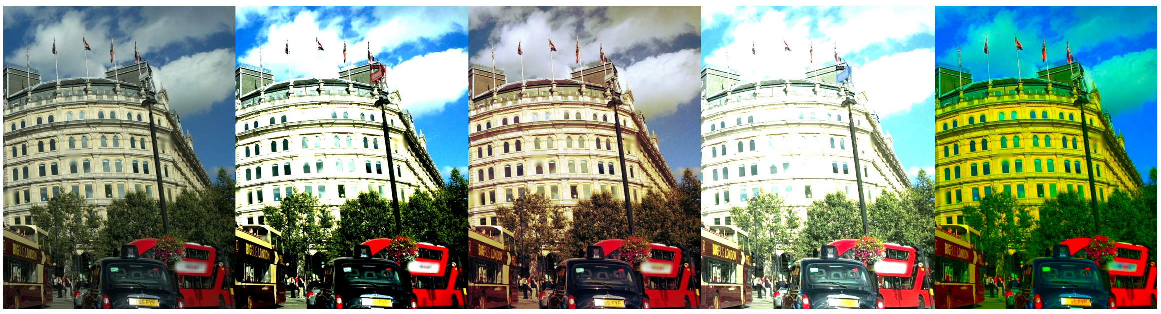
Figure 22. CenterMask; Figure 10 alterations (left to right): original, contrast, equalizer, exposure, and saturation.

Figure 22 has the true positive detections with contrast and exposure modifications, although the confidence is still low.

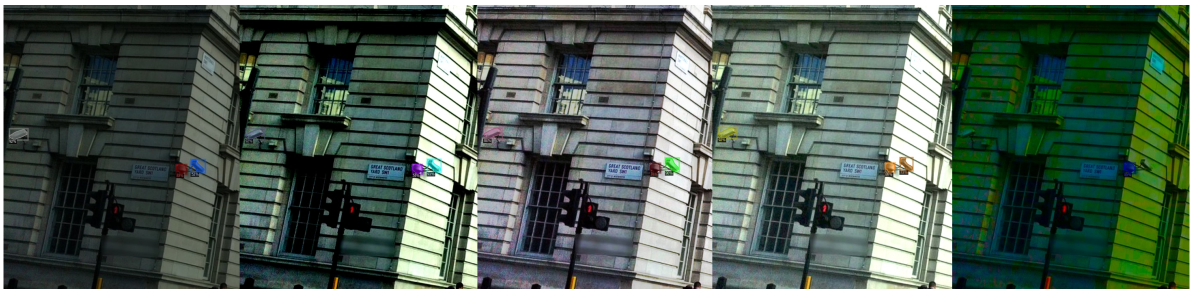

Figure 23. CenterMask; Figure 11 alterations (left to right): original, contrast, equalizer, exposure, and saturation.

With Figure 23, we can observe much better results with contrast, equalization, and exposure modifications. Equalizer seems to bring out the best results here.

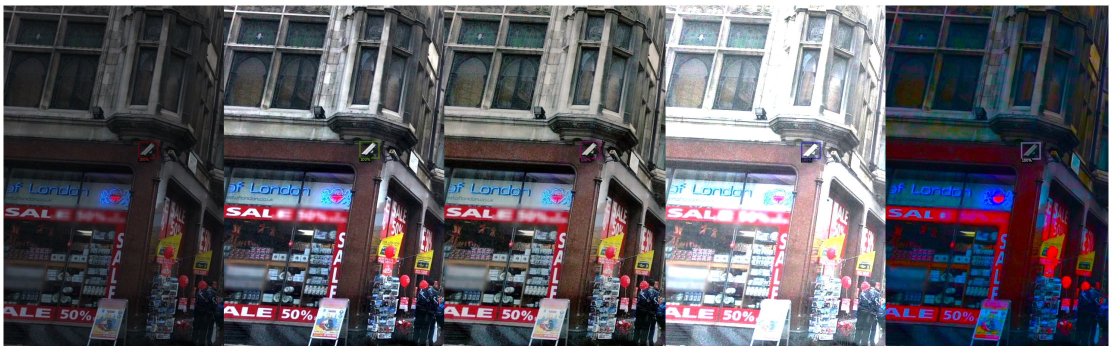

Figure 24. TridentNet; Figure 7 alterations (left to right): original, contrast, equalizer, exposure, and saturation.

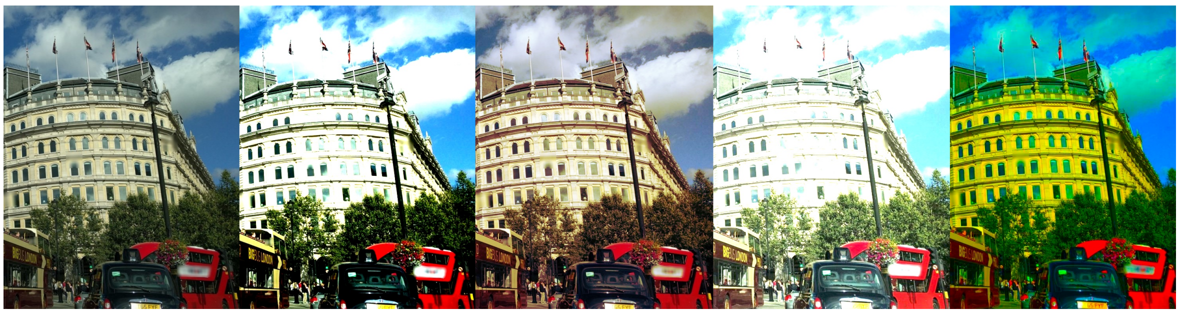

Figure 25. TridentNet; Figure 10 alterations (left to right): original, contrast, equalizer, exposure, and saturation.

TridentNet (Li et al. 2019) detector doesn't seem to get better results with the modifications as seen with Figures 24 and 25, with no improvements on the detection. Although with equalization, one false negative is converted to positive, the confidence of the one true detection is lowered in return (Figure 26).

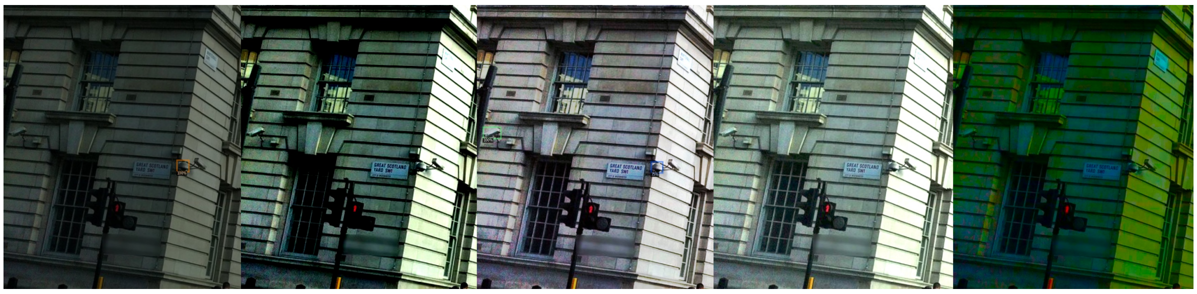

Figure 26. TridentNet; Figure 11 alterations (left to right): original, contrast, equalizer, exposure, and saturation.

Altered images had diverse results. ATSS (S. Zhang et al. 2019) seems to detect better with images with larger contrast. CenterMask (Lee and Park 2019) enjoys better performance with many of the modifications, but increasing the contrast of the images appears to be the best bet. TridentNet (Li et al. 2019) however doesn't gain performance from the modifications to a meaningful effect and they may harm the results more.

# 6 DISCUSSION

This chapter discusses the overall performance of the detectors and the limitations of the models presented. In this chapter, we also answer the third research question: how can we improve on the models?

## 6.1 Performance of the model and improvements

The results from the tests are promising. The models show that even with street view images with varied quality (e.g., blurred, distortions of white-balance/hue/saturation/contrast), detection can be undoubtedly precise. The results from the testing dataset show that the model can perform under the circumstances of the proposed use case of detecting cameras for the privacy route system. The confidence of the detected cameras is quite high, especially with our detector build on-top of the TridentNet architecture (Li et al. 2019). In general, localization brings in more error than the classification, which is promising as the localization error can be easier to handle in real-life use cases.

Results presented here serve as a decent baseline to build on. The metrics, visual results, and training losses point out a few flaws that can be dealt with. False negatives are mostly due to poor light conditions, edges blending into the background, and similar-looking objects. These flaws could be improved upon with the introduction of more and better samples for the training and validation datasets; darker and blurrier images are needed, and harder samples need to be implemented into the dataset in larger quantities. False positives can also be a problem, and more different types of objects (e.g., light fixtures, light poles, other street, and building objects) need to be implemented into the training dataset as background (i.e., non-cameras class). The results from the altered images (see Section 5.4.1) show that contrast plays an important role in the success of detection; thus, smartly increasing or improving the contrast of images could help increase detection success rates. Other dataset augmentation (e.g., Section 3.3.9) methods could be a possibility as well to alleviate the problems, and it might also help with overfitting.

The training process can also be improved. The models could be trained on a higher image resolution. This would affect the scale of the objects detected, but would most likely improve on the results. Although training speeds would take a hit with this and batch sizes might need an adjustment. The base learning rate and learning schedules could also be adjusted to suit the dataset and setup better. Lower training speeds, a more aggressive schedule, and higher maximum iterations should bring out improvements.

### 6.1.1 Sensor fusion

One way of improving detection results could be a technique commonly known as sensor fusion (Gustafsson 2010). The idea is to come to a conclusion based on several data sources. This method could be applied to our project by running the images through several detector models. Even with improvements to the detector models, perfect precision might be impossible to achieve. As shown in this thesis, the precision and false detections from samples vary; thus, it could be beneficial to use two or more detector models to cross-validate the detections. As inference times with GPU clusters are low, running the images through several models can be achieved with ease. It could also be favorable to tune the detectors for different image scales and alterations (e.g., contrast, hue - as presented in Section 5.4.1) with detector selection and specific datasets.

## 6.2 Future work

In addition to the improvements proposed to improve the detector performance, other future research is planned to be conducted. The implementation of the detector model to the privacy route creator framework is the near term first step with future work. A web scraper for the geotagged images from street view sources needs to be developed for this use case. The updatability requirement can also be implemented in the near term as the transfer training resuming scripts can be created and implemented with ease. Our evaluator should be developed to focus on the metrics we value the most and base the further development on them.

As the dataset requires more samples, a web scraper for relevant images from sources such as Flickr (*Flickr*) should be developed. A 360-degree camera is intended to be used for dataset collection with personally captured images. The limitations lie with the environment and the number of cameras available. Although, the amount of variations for camera poses is significantly increased and the light conditions can be affected by the time-of-the-day during image capturing.

The 360-degree camera is intended to be used for a demonstration as well. A lite version of a detector could be running in a backend server, a more resource-constrained device such as a Raspberry Pi (*Teach, Learn, and Make with Raspberry Pi – Raspberry Pi*) will act as the mediator between the backend detector and the 360-degree camera. The cameras should also be indicated to the operator through a led, buzzer or a screen attached to the Raspberry Pi.

More use cases can be implemented in the future as the detectors improve and possibly achieve near-perfect precision. The possibility of a multi-detector application is also being deliberated.

# 7 CONCLUSION

This thesis explored the ways to create a precise object detector to find CCTV cameras from images (e.g., street view) and video. It offers a baseline dataset of CCTV cameras with two distinct classes, three varied models trained on that dataset to detect the cameras, and the evaluation results in the form of MS COCO (T. Lin et al. 2014) metrics and visual evaluation. Four research questions were pointed out:

- RQ1: “How can we accurately and efficiently detect ’CCTV camera’ objects from images and videos?”
- RQ2: “How do the various trained models fare against each other?”
- RQ3: “How can we improve on the models?”
- RQ4: "What are the main challenges and applicative use-cases of such technology?"

To answer the main research question , several detectors were trained with different architectures (e.g., CenterMask2, ATSS, TridentNet) with the same dataset and then further evaluated for their performance and speed with two datasets with different images. The supporting research questions two and three were answered by evaluating the models with de facto metrics from MS COCO object detection competition as well as with visual evaluation by inspecting the images. The improvement ideas were derived from these results and presented. To answer the supporting research question number fore, applicative use cases were proposed, and CCTV camera-related issues were presented.

During the literature review part, the basic operation of convolutional neural networks and the tools and methods for state-of-the-art object detectors were overviewed. An overview of the architectures and distinct features of the selected detectors were presented. This knowledge was applied in the methodology section. Three design goals were set; proper precision for the use cases, a possibility for the model to be updated, and efficiency for detection from video sources. Detector’s precision show promise, and primary goals were met. Updatability of the model can be accomplished due to transfer training with the Detectron2 (Wu et al. 2020) framework. Although none of the models can accomplish real-time detection from video, CenterMask-Lite (Lee and Park 2019) produced results that were close and detection from video is still possible with sub-real-time framerates.

Our state-of-the-art detectors reached great results surpassing 90% precision, and a detector can be implemented in the current state for some use cases. Near-term improvements were discussed and can be implemented. The chosen framework is suitable for the proposed projects. It is customizable and offers choices for different architectures; therefore, improvements due to architecture evolution can be implemented easily. Overall, the detectors chosen to be trained and evaluated offer great performance. At this point in development, it is hard to pick a winning detector model as great results can be achieved with all of them, and the reasoning for choosing just one should be made on a use case basis after further improvements.

## 7.1 Acknowledgements

We acknowledge grants of computer capacity from the Finnish Grid and Cloud Infrastructure (persistent identifier urn:nbn:fi:research-infras-2016072533).

# Bibliography

Abadi, Martín, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, et al. 2016. “TensorFlow: Large-Scale Machine Learning on Heterogeneous Distributed Systems”. arXiv: 1603.04467 [cs.DC].

Abaya, W. F., J. Basa, M. Sy, A. C. Abad, and E. P. Dadios. 2014. “Low cost smart security camera with night vision capability using Raspberry Pi and OpenCV”. In *2014 International Conference on Humanoid, Nanotechnology, Information Technology, Communication and Control, Environment and Management (HNICEM),* 1–6.

Akbar, M. A., and T. N. Azhar. 2018. “Concept of Cost Efficient Smart CCTV Network for Cities in Developing Country”. In *2018 International Conference on ICT for Smart Society (ICISS),* 1–4.

Alshammari, Abdullah, and Danda B. Rawat. 2019. “Intelligent Multi-Camera Video Surveillance System for Smart City Applications”. In *2019 IEEE 9th Annual Computing and Communication Workshop and Conference (CCWC),* 0317–0323. doi:10.1109/CCWC.2019.8666579.

Andrejevic, Mark, and Neil Selwyn. 2019. “Facial recognition technology in schools: critical questions and concerns”. *Learning, Media and Technology.* ISSN: 1743-9884. doi:10.1080/17439884.2020.1686014.

Bah, Serign, and Fang Ming. 2019. “An improved face recognition algorithm and its application in attendance management system”. *Array* 5. doi:10.1016/j.array.2019.100014.

Bannister, Adam. 2019. *The Video Surveillance Report 2019 The Cloud, AI, Face Recognition and Brexit.* Technical report.

Barry, D., M. Shah, M. Keijsers, H. Khan, and B. Hopman. 2019. “xYOLO: A Model For Real-Time Object Detection In Humanoid Soccer On Low-End Hardware”. In *2019 International Conference on Image and Vision Computing New Zealand (IVCNZ),* 1–6.

*Benchmarks — Detectron2 0.1.1 Documentation.* `https://detectron2.readthedocs.io/notes/benchmarks.html`.

Bischoff, Paul. 2019. *Surveillance Camera Statistics: Which City Has the Most CCTV Cameras?*

Bolya, Daniel, Chong Zhou, Fanyi Xiao, and Yong Jae Lee. 2019. *YOLACT++: Better Real-Time Instance Segmentation.*

Brownlee, Jason. 2016. *Deep Learning With Python: Develop Deep Learning Models on Theano and TensorFlow Using Keras.* Machine Learning Mastery.

Brownlee, Jason. 2019a. *A Gentle Introduction to Object Recognition With Deep Learning.*

Brownlee, Jason. 2019b. *TensorFlow 2 Tutorial: Get Started in Deep Learning With Tf.Keras.*

Brownlee, Jason. 2020. *How to Calculate Precision, Recall, and F-Measure for Imbalanced Classification.*

Cai, Zhaowei, and Nuno Vasconcelos. 2018. *Cascade R-CNN: Delving into High Quality Object Detection.*

Cameron. 2018. *A Guide to Essential Types of Security Cameras and When to Use Each Type of Camera.* `https://www.securityindustry.org/2018/11/20/security-camera-variants-their-situational-designs/`.

*COCO - Common Objects in Context.* `http://cocodataset.org/#home`.

*Conda — Conda 4.8.3 Documentation.* `https://docs.conda.io/projects/conda/en/latest/`.

Cortes, Corinna, and Vladimir Vapnik. 1995. "Support-Vector Networks". *Mach. Learn.* (USA) 20 (3): 273–297. ISSN: 0885-6125. doi:`10.1023/A:1022627411411`. `https://doi.org/10.1023/A:1022627411411`.

Costin, Andrei. 2016. "Security of CCTV and Video Surveillance Systems: Threats, Vulnerabilities, Attacks, and Mitigations". In *Proceedings of the 6th International Workshop on Trustworthy Embedded Devices,* 45–54. TrustED '16. Vienna, Austria: Association for Computing Machinery. ISBN: 978-1-4503-4567-5. doi:`10.1145/2995289.2995290`.

Dai, Jifeng, Haozhi Qi, Yuwen Xiong, Yi Li, Guodong Zhang, Han Hu, and Yichen Wei. 2017. “Deformable Convolutional Networks”. arXiv: `1703.06211 [cs.CV]`.

Dalal, N., and B. Triggs. 2005. “Histograms of oriented gradients for human detection”. In *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR’05),* volume 1, 886–893 vol. 1.

*Detectron 2.0.1.1 Documentation.* `https://detectron2.readthedocs.io/`.

Duan, Kaiwen, Song Bai, Lingxi Xie, Honggang Qi, Qingming Huang, and Qi Tian. 2019. “CenterNet: Keypoint Triplets for Object Detection”, volume abs/1904.08189. arXiv: `1904.08189`. `http://arxiv.org/abs/1904.08189`.

Ebersole, Mark. 2012. *What Is CUDA | NVIDIA Official Blog.* `https://blogs.nvidia.com/blog/2012/09/10/what-is-cuda-2/`.

Erdem, Ugur Murat, and Stan Sclaroff. 2004. “Optimal Placement of Cameras in Floorplans to Satisfy Task Requirements and Cost Constraints”.

European Commission. 2020. *WHITE PAPER On Artificial Intelligence - A European Approach to Excellence and Trust.*

European Commission. *A New Era for Data Protection in the EU What Changes after May 2018.*

European Data Protection Board. 2020. *Guidelines 3/2019 on Processing of Personal Data through Video Devices.*

Everingham, Mark, Luc Van Gool, C. K. I. Williams, J. Winn, and Andrew Zisserman. 2010. “The PASCAL Visual Object Classes (VOC) challenge”.

Everingham, Mark, and John Winn. 2012. “The PASCAL Visual Object Classes Challenge 2012 (VOC2012) Development Kit”: 32.

*Facebook AI Research.* `https://ai.facebook.com/`.

Felzenszwalb, P. F., R. B. Girshick, D. McAllester, and D. Ramanan. 2010. “Object Detection with Discriminatively Trained Part-Based Models”. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 32 (9): 1627–1645.

*Flickr.* https://www.flickr.com/.

Freund, Yoav, and Robert E Schapire. 1997. “A Decision-Theoretic Generalization of On-Line Learning and an Application to Boosting”. *J. Comput. Syst. Sci.* (USA) 55 (1): 119–139. ISSN: 0022-0000. doi:10.1006/jcss.1997.1504. https://doi.org/10.1006/jcss.1997.1504.

Furht, Borko, Esad Akar, and Whitney Angelica Andrews. 2018. “Introduction to Digital Imaging”. In *Digital Image Processing: Practical Approach,* 1–5. SpringerBriefs in Computer Science. Cham: Springer International Publishing. ISBN: 978-3-319-96634-2. doi:10.1007/978-3-319-96634-2_1.

*GDPR Enforcement Tracker - List of GDPR Fines.* http://www.enforcementtracker.com.

Gilewski, Jarosław. 2020. *Detectron2 Pipeline.*

Girshick, Ross. 2015. “Fast R-CNN”. In *International Conference on Computer Vision ({ICCV}).*

*YOLOV3 GitHub Repositories.* https://github.com.

*Street View Photos Come from Two Sources, Google and Our Contributors.* https://www.google.com/streetview/explore/.

Grother, Patrick, Mei Ngan, and Kayee Hanaoka. 2019. *Face Recognition Vendor Test Part 3:: Demographic Effects.* Technical report. Gaithersburg, MD: National Institute of Standards and Technology.

Guennouni, S., A. Ahaitouf, and A. Mansouri. 2014. “Multiple object detection using OpenCV on an embedded platform”. In *2014 Third IEEE International Colloquium in Information Science and Technology (CIST),* 374–377.

Guo, Jian, and Stephen Gould. 2015. “Deep CNN Ensemble with Data Augmentation for Object Detection”. arXiv: 1506.07224 [cs.CV].

Guri, Mordechai, Dima Bykhovsky, and Yuval Elovici. 2017. *aIR-Jumper: Covert Air-Gap Exfiltration/Infiltration via Security Cameras & Infrared (IR).* arXiv: 1709.05742 [cs.CR].

Guri, Mordechai, Boris Zadov, Eran Atias, and Yuval Elovici. 2017. *LED-it-GO: Leaking (a lot of) Data from Air-Gapped Computers via the (small) Hard Drive LED.* arXiv: 1702.06715 [cs.CR].

Guri, Mordechai, Boris Zadov, Andrey Daidakulov, and Yuval Elovici. 2017. *xLED: Covert Data Exfiltration from Air-Gapped Networks via Router LEDs.* arXiv: 1706.01140 [cs.CR].

Gustafsson, Fredrik. 2010. *Statistical sensor fusion.* Studentlitteratur. ISBN: 978-91-44-05489-6.

Halawa, Lavin J., Adi Wibowo, and Ferda Ernawan. 2019. "Face Recognition Using Faster R-CNN with Inception-V2 Architecture for CCTV Camera". In *2019 3rd International Conference on Informatics and Computational Sciences (ICICoS),* 1–6. doi:10.1109/ICICoS48119.2019.8982383.

Hanin, Boris. 2018. "Which Neural Net Architectures Give Rise to Exploding and Vanishing Gradients?"

He, K., X. Zhang, S. Ren, and J. Sun. 2015. "Spatial Pyramid Pooling in Deep Convolutional Networks for Visual Recognition". *IEEE Transactions on Pattern Analysis and Machine Intelligence* 37 (9): 1904–1916.

He, Kaiming, Georgia Gkioxari, Piotr Dollár, and Ross Girshick. 2017. "Mask R-CNN". arXiv: 1703.06870 [cs.CV].

He, Kaiming, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2015. "Deep Residual Learning for Image Recognition". arXiv: 1512.03385 [cs.CV].

Hochreiter, Sepp. 1991. "Untersuchungen Zu Dynamischen Neuronalen Netzen". PhD thesis, Institut fur Informatik Technische Universität Munchen.

Howard, Andrew G., Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. 2017. "MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications". arXiv: 1704.04861 [cs.CV].

Hu, Jie, Li Shen, Samuel Albanie, Gang Sun, and Enhua Wu. 2017. "Squeeze-and-Excitation Networks". arXiv: 1709.01507 [cs.CV].

Huang, Lichao, Yi Yang, Yafeng Deng, and Yinan Yu. 2015. “DenseBox: Unifying Landmark Localization with End to End Object Detection”. arXiv: 1509.04874 [cs.CV].

ILSVRC 2010. *ImageNet Large Scale Visual Recognition Competition 2010.* http://www.image-net.org/challenges/LSVRC/2010/.

*ImageNet Object Localization Challenge.* https://kaggle.com/c/imagenet-object-localization-challenge.

Ioffe, Sergey. 2017. “Batch Renormalization: Towards Reducing Minibatch Dependence in Batch-Normalized Models”. arXiv: 1702.03275 [cs.LG].

Ioffe, Sergey, and Christian Szegedy. 2015. “Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift”. arXiv: 1502.03167 [cs.LG].

*IPVM Camera Calculator V3.* https://calculator.ipvm.com/.

Jiao, Licheng, Fan Zhang, Fang Liu, Shuyuan Yang, Lingling Li, Zhixi Feng, and Rong Qu. 2019. “A Survey of Deep Learning-Based Object Detection”. *IEEE Access* 7:128837–128868. ISSN: 2169-3536. doi:10.1109/access.2019.2939201. http://dx.doi.org/10.1109/ACCESS.2019.2939201.

Jocher, Glenn, guigarfr, perry0418, Ttayu, Veitch-Michaelis Josh, Gabriel Bianconi, Fatih Baltacı, Daniel Suess, WannaSeaU, and IlyaOvodov. 2019. *Ultralytics/Yolov3: Rectangular Inference, Conv2d + Batchnorm2d Layer Fusion.* Zenodo. doi:10.5281/zenodo.2672652.

John D. Woodward, Jr., Christopher Horn, Julius Gatune, and Aryn Thomas. 2003. *Biometrics : A Look at Facial Recognition.* Documented Briefing / Rand Corporation. RAND Corporation. ISBN: 0-8330-3302-6.

Kadir, K., M. K. Kamaruddin, H. Nasir, S. I. Safie, and Z. A. K. Bakti. 2014. “A comparative study between LBP and Haar-like features for Face Detection using OpenCV”. In *2014 4th International Conference on Engineering Technology and Technopreneuship (ICE2T),* 335–339.

Kambourakis, G., C. Kolias, and A. Stavrou. 2017. “The Mirai botnet and the IoT Zombie Armies”. In *MILCOM 2017 - 2017 IEEE Military Communications Conference (MILCOM),* 267–272.

Kenichi Yabuta and Hitoshi Kitazawa. 2008. “Optimum camera placement considering camera specification for security monitoring”. In *2008 IEEE International Symposium on Circuits and Systems,* 2114–2117.

Ketkar, Nikhil. 2017. “Introduction to PyTorch”. In *Deep Learning with Python: A Hands-on Introduction,* 195–208. Berkeley, CA: Apress. ISBN: 978-1-4842-2766-4. doi:10.1007/978-1-4842-2766-4_12.

Kong, Tao, Fuchun Sun, Huaping Liu, Yuning Jiang, and Jianbo Shi. 2019. “FoveaBox: Beyond Anchor-based Object Detector”. arXiv: 1904.03797 [cs.CV].

Krizhevsky, Alex, Ilya Sutskever, and Geoffrey E. Hinton. 2017. “ImageNet Classification with Deep Convolutional Neural Networks”. *Commun. ACM* 60 (6): 84–90. ISSN: 0001-0782. doi:10.1145/3065386. https://doi.org/10.1145/3065386.

Law, Hei, Yun Teng, Olga Russakovsky, and Jia Deng. 2019. “CornerNet-Lite: Efficient Keypoint Based Object Detection”. arXiv: 1904.08900 [cs.CV].

Lazebnik, S., C. Schmid, and J. Ponce. 2006. “Beyond Bags of Features: Spatial Pyramid Matching for Recognizing Natural Scene Categories”. In *2006 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR’06),* 2:2169–2178.

LeCun, Yann, Yoshua Bengio, and Geoffrey Hinton. 2015. “Deep Learning”. *Nature* 521 (7553): 436–444. ISSN: 1476-4687. doi:10.1038/nature14539.

Lee, Youngwan, Joong-won Hwang, Sangrok Lee, Yuseok Bae, and Jongyoul Park. 2019. “An Energy and GPU-Computation Efficient Backbone Network for Real-Time Object Detection”. arXiv: 1904.09730 [cs.CV].

Lee, Youngwan, and Jongyoul Park. 2019. “CenterMask : Real-Time Anchor-Free Instance Segmentation”. arXiv: 1911.06667 [cs.CV].

Lee, Youngwan, and Jongyoul Park. 2020. *CenterMask: Real-Time Anchor-Free Instance Segmentation.*

Li, Yanghao, Yuntao Chen, Naiyan Wang, and Zhaoxiang Zhang. 2019. “Scale-Aware Trident Networks for Object Detection”. arXiv: 1901.01892 [cs.CV].

Lienhart, R., and J. Maydt. 2002. “An Extended Set of Haar-like Features for Rapid Object Detection”. In *Proceedings. International Conference on Image Processing,* volume 1. doi:10.1109/ICIP.2002.1038171.

Lin, Tsung-Yi, Piotr Dollár, Ross Girshick, Kaiming He, Bharath Hariharan, and Serge Belongie. 2016. “Feature Pyramid Networks for Object Detection”. arXiv: 1612.03144 [cs.CV].

Lin, Tsung-Yi, Priya Goyal, Ross Girshick, Kaiming He, and Piotr Dollár. 2017. “Focal Loss for Dense Object Detection”. arXiv: 1708.02002 [cs.CV].

Lin, Tsung-Yi, Michael Maire, Serge J. Belongie, Lubomir D. Bourdev, Ross B. Girshick, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C. Lawrence Zitnick. 2014. “Microsoft COCO: Common Objects in Context”. *CoRR* abs/1405.0312. arXiv: 1405.0312. http://arxiv.org/abs/1405.0312.

Liu, Wei, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott Reed, Cheng-Yang Fu, and Alexander C. Berg. 2016. “SSD: Single Shot MultiBox Detector”, 21–37. Springer International Publishing. ISBN: 9783319464480. doi:10.1007/978-3-319-46448-0_2. http://dx.doi.org/10.1007/978-3-319-46448-0_2.

Long, Jonathan, Evan Shelhamer, and Trevor Darrell. 2014. “Fully Convolutional Networks for Semantic Segmentation”. arXiv: 1411.4038 [cs.CV].

Lowe, David G. 2004. “Distinctive Image Features from Scale-Invariant Keypoints”. *Int. J. Comput. Vision* 60 (2): 91–110. ISSN: 0920-5691. doi:10.1023/B:VISI.0000029664.99615.94. http://dx.doi.org/10.1023/B:VISI.0000029664.99615.94.

Markets and Markets. *Facial Recognition Market by Software & Services - 2024.* https://www.marketsandmarkets.com/Market-Reports/facial-recognition-market-995.html.

Massa, Francisco, and Ross Girshick. 2020. *Faster R-CNN and Mask R-CNN in PyTorch 1.0.* Facebook Research.

Mileva, Mila, and Anthony Michael Burton. 2019. “Face search in CCTV surveillance”. *Cognitive research: principles and implications* 4 (3). ISSN: 2365-7464.

Mowery, Keaton, Eric Wustrow, Tom Wypych, Corey Singleton, Chris Comfort, Eric Rescorla, J. Alex Halderman, Hovav Shacham, and Stephen Checkoway. 2014. “Security Analysis of a Full-Body Scanner”. In *23rd USENIX Security Symposium (USENIX Security 14),* 369–384. San Diego, CA: USENIX Association. ISBN: 978-1-931971-15-7. https://www.usenix.org/conference/usenixsecurity14/technical-sessions/presentation/mowery.

Murphy, John. 2016. *An Overview of Convolutional Neural Network Architectures for Deep Learning.*

Murray, Alan T., Kamyoung Kim, James W. Davis, Raghu Machiraju, and Richard E. Parent. 2007. “Coverage optimization to support security monitoring”. *Comput. Environ. Urban Syst.* 31:133–147.

Nagaraj, S., B. Muthiyan, S. Ravi, V. Menezes, K. Kapoor, and H. Jeon. 2017. “Edge-based street object detection”. In *2017 IEEE SmartWorld, Ubiquitous Intelligence Computing, Advanced Trusted Computed, Scalable Computing Communications, Cloud Big Data Computing, Internet of People and Smart City Innovation (SmartWorld/SCALCOM/UIC/ATC/CBDCom/IOP/SCI),* 1–4.

Nielsen, Michael A. 2015. *Neural Networks and Deep Learning.* Determination Press.

*NVIDIA Tesla Supercomputing.* https://www.nvidia.com/en-gb/data-center/tesla/.

NVIDIA Developer. 2017. *Jetson TX2 Module.* https://developer.nvidia.com/embedded/jetson-tx2.

*OpenCV.* https://opencv.org/.

*OpenStreetCam.* https://openstreetcam.org/.

*Papers With Code : Object Detection.* https://paperswithcode.com/task/object-detection.

Park, Tae-Sung, and Moon-Seog Jun. 2011. "User Authentication Protocol for Blocking Malicious User in Network CCTV Environment". In *2011 6th International Conference on Computer Sciences and Convergence Information Technology (ICCIT),* 18–24.

Paszke, Adam, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, et al. 2019. "PyTorch: An Imperative Style, High-Performance Deep Learning Library". arXiv: 1912.01703 [cs.LG].

Peffers, Ken, Tuure Tuunanen, Charles Gengler, Matti Rossi, Wendy Hui, Ville Virtanen, and Johanna Bragge. 2006. "The design science research process: A model for producing and presenting information systems research". *Proceedings of First International Conference on Design Science Research in Information Systems and Technology DESRIST.*

Philippou, Oliver. 2019. *Video Surveillance Installed Base Report.* Technical report.

Qi, X., T. Wang, and J. Liu. 2017. "Comparison of Support Vector Machine and Softmax Classifiers in Computer Vision". In *2017 Second International Conference on Mechanical, Control and Computer Engineering (ICMCCE),* 151–155.

Rastegari, Mohammad, Vicente Ordonez, Joseph Redmon, and Ali Farhadi. 2016. "XNOR-Net: ImageNet Classification Using Binary Convolutional Neural Networks". arXiv: 1603.05279 [cs.CV].

Rawat, Waseem, and Zenghui Wang. 2017. "Deep Convolutional Neural Networks for Image Classification: A Comprehensive Review". *Neural Computation* 29 (9): 2352–2449. doi:10.1162/neco_a_00990.

Redmon, Joseph. 2016. *Darknet: Open Source Neural Networks in C.* http://pjreddie.com/darknet/.

Redmon, Joseph, Santosh Divvala, Ross Girshick, and Ali Farhadi. 2016. "You Only Look Once: Unified, Real-Time Object Detection". In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR),* 779–788. Las Vegas, NV, USA: IEEE. ISBN: 978-1-4673-8851-1. doi:10.1109/CVPR.2016.91.

Redmon, Joseph, and Ali Farhadi. 2018. “YOLOv3: An Incremental Improvement”.

Ren, Shaoqing, Kaiming He, Ross Girshick, and Jian Sun. 2015. “Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks”. arXiv: `1506.01497 [cs.CV]`.

Rey, Javier. *Object Detection with Deep Learning: The Definitive Guide | Tryolabs Blog.*

Rezatofighi, Hamid, Nathan Tsoi, JunYoung Gwak, Amir Sadeghian, Ian Reid, and Silvio Savarese. 2019. “Generalized Intersection over Union: A Metric and A Loss for Bounding Box Regression”. arXiv: `1902.09630 [cs.CV]`.

Rosebrock, Adrian. 2017. *Deep Learning with OpenCV - PyImageSearch.* `https://www.pyimagesearch.com/2017/08/21/deep-learning-with-opencv/`.

Russakovsky, Olga, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, et al. 2014. “ImageNet Large Scale Visual Recognition Challenge”. arXiv: `1409.0575 [cs.CV]`.

Russell, BryanC., Antonio Torralba, KevinP. Murphy, and WilliamT. Freeman. 2008. “LabelMe: A Database and Web-Based Tool for Image Annotation”. *International Journal of Computer Vision* 77 (1-3): 157–173. ISSN: 0920-5691. doi:`10.1007/s11263-007-0090-8`. `http://dx.doi.org/10.1007/s11263-007-0090-8`.

*SchedMD | Slurm Support and Development.* `https://www.schedmd.com/`.

Shin, Hoo-Chang, Holger R Roth, Mingchen Gao, Le Lu, Ziyue Xu, Isabella Nogues, Jianhua Yao, Daniel J. Mollura, and Ronald M. Summers. 2016. “Deep Convolutional Neural Networks for Computer-Aided Detection: CNN Architectures, Dataset Characteristics and Transfer Learning”. *IEEE Transactions on Medical Imaging* 35:1285–1298.

*Shodan.* `https://www.shodan.io/`.

Singh, Bharat, and Larry S. Davis. 2018. “An Analysis of Scale Invariance in Object Detection - SNIP”. In *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition,* 3578–3587. Salt Lake City, UT: IEEE. ISBN: 978-1-5386-6420-9. doi:`10.1109/CVPR.2018.00377`.

Stolton, Samuel. 2020. *LEAK: Commission Considers Facial Recognition Ban in AI 'White Paper'.*

Suarez, Oscar Deniz, and Arie Leeuwesteijn. 2014. *OpenCV Essentials.* Community Experience Distilled. Birmingham, England: Packt Publishing. ISBN: 978-1-78398-424-4.

Tan, Mingxing, and Quoc V. Le. 2019. "EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks". arXiv: 1905.11946 [cs.LG].

Tan, Mingxing, Ruoming Pang, and Quoc V. Le. 2019. "EfficientDet: Scalable and Efficient Object Detection". arXiv: 1911.09070 [cs.CV].

Tang, Yichuan. 2013. "Deep Learning using Linear Support Vector Machines". arXiv: 1306.0239 [cs.LG].

*Teach, Learn, and Make with Raspberry Pi – Raspberry Pi.* https://www.raspberrypi.org.

*THE CCTV MAP.* https://thecctvmap.wordpress.com/.

*The PASCAL Visual Object Classes.* http://host.robots.ox.ac.uk/pascal/VOC/.

Tian, Zhi, Chunhua Shen, Hao Chen, and Tong He. 2019. "FCOS: Fully Convolutional One-Stage Object Detection". arXiv: 1904.01355 [cs.CV].

Tzutalin. 2015. *LabelImg.*

Viola, Paul, and Michael J. Jones. 2004. "The system of face detection based on OpenCV". In *International Journal of Computer Vision,* 57:137–154. doi:https://doi.org/10.1023/B:VISI.0000013087.49260.fb.

Viraktamath, Shivashekharayya, Aditya Katti, and Pavan Kulkarni. 2013. "Face Detection and Tracking using OpenCV", 1:45–50. doi:10.9756/SIJCNCE/V4I3/0103540102.

Wada, Kentaro. 2016. *labelme: Image Polygonal Annotation with Python.* https://github.com/wkentaro/labelme.

Wang, Yu Emma, Gu-Yeon Wei, and David Brooks. 2019. "Benchmarking TPU, GPU, and CPU Platforms for Deep Learning". arXiv: 1907.10701 [cs.LG].

Wu, Yuxin, Alexander Kirillov, Francisco Massa, Wan-Yen Lo, and Ross Girshick. 2019. *Detectron2.* Facebook Research.

Wu, Yuxin, Alexander Kirillov, Francisco Massa, Wan-Yen Lo, and Ross Girshick. 2020. *Detectron2: A PyTorch-Based Modular Object Detection Library.* `https://web.archive.org/web/20200305225304/https://ai.facebook.com/blog/-detectron2-a-pytorch-based-modular-object-detection-library-/`.

Xianghua Fan, Fuyou Zhang, Haixia Wang, and Xiao Lu. 2012. "The system of face detection based on OpenCV". In *2012 24th Chinese Control and Decision Conference (CCDC),* 648–651.

Xie, Saining, Ross Girshick, Piotr Dollar, Zhuowen Tu, and Kaiming He. 2017. "Aggregated Residual Transformations for Deep Neural Networks". In *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR),* 5987–5995. Honolulu, HI: IEEE. ISBN: 978-1-5386-0457-1. doi:`10.1109/CVPR.2017.634`.

Yu, Fisher, and Vladlen Koltun. 2015. "Multi-Scale Context Aggregation by Dilated Convolutions". arXiv: `1511.07122 [cs.CV]`.

Yu, Jiahui, Yuning Jiang, Zhangyang Wang, Zhimin Cao, and Thomas Huang. 2016. "UnitBox: An Advanced Object Detection Network". *Proceedings of the 2016 ACM on Multimedia Conference - MM '16.* doi:`10.1145/2964284.2967274`. `http://dx.doi.org/10.1145/2964284.2967274`.

Zhang, Aston, Zachary C. Lipton, Mu Li, and Alexander J. Smola. No date. *Dive into Deep Learning.*

Zhang, Shifeng, Cheng Chi, Yongqiang Yao, Zhen Lei, and Stan Z. Li. 2019. "Bridging the Gap Between Anchor-based and Anchor-free Detection via Adaptive Training Sample Selection". arXiv: `1912.02424 [cs.CV]`.

Zhao, Zhong-Qiu, Peng Zheng, Shou-tao Xu, and Xindong Wu. 2018. "Object Detection with Deep Learning: A Review". arXiv: `1807.05511 [cs.CV]`.

Zhou, Xingyi, Jiacheng Zhuo, and Philipp Krähenbühl. 2019. “Bottom-up Object Detection by Grouping Extreme and Center Points”. arXiv: `1901.08043 [cs.CV]`.

Zhu, Chenchen, Yihui He, and Marios Savvides. 2019. “Feature Selective Anchor-Free Module for Single-Shot Object Detection”. In *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR),* 840–849. Long Beach, CA, USA: IEEE. ISBN: 978-1-72813-293-8. doi:`10.1109/CVPR.2019.00093`.

# Appendices

## A License and copyrights clarifications

The images presented in Figure 4 to Figure 26 are from OpenStreetCam (*OpenStreetCam*), and they are licensed under CC BY-SA 4.0.