U^2OF: Using Saliency Maps And Optical Flow To Mitigate Concept Drift In Thermal Images

A Master Thesis

Report Group

AALBORG UNIVERSITY

STUDENT REPORT

Aalborg University
Department of Electronic Systems

AALBORG UNIVERSITY
STUDENT REPORT

Department of Electronic Systems
Aalborg University
http://www.aau.dk

Title:
U^{2}OF: Using Saliency Maps And Optical Flow To Mitigate Concept Drift In Thermal Images

Theme:
Scientific Theme

Project Period:
Spring Semester 2024

Project Group:
Group 1042

Participant(s):
Kazim Kerem Kocan
Laurits Winter

Supervisor(s):
Andreas Aakerberg

Copies: 1

Page Numbers: 106

Date of Completion:
May 31, 2024

Abstract:

Concept drift is a challenge for many data-driven systems which are deployed in a real-world object detection system. Systems which use thermal cameras are also acutely affected by this as the thermal signature for pedestrians is affected by the changing thermal radiation in the scene over seasons. This work proposes a method to mitigate concept drift by preprocessing the images using saliency maps from U^{2}Net [1] and Gunnar Farnebäck optical flow [2], stacking them on top of the original image. The preprocessed images are then used in pedestrian detection using YOLOv5 [3]. While the proposed methods did not show significant improvement further ablation studies have shown that it is more confident in detection and disproportionally penalized by finding pedestrians who are not annotated. There is qualitative evidence showing that the proposed method can find pedestrians with a higher confidence in big and small scales which affects the proposed method negatively for non-annotated pedestrians. Further work is needed to identify if this can also be shown quantitatively, and if the proposed method has mitigated concept drift.

List of acronyms

Contents

Preface

Aalborg University, May 31, 2024

Kazim Kerem Kocan
<kkocan19@student.aau.dk>

Laurits Winter
<lwinte19@student.aau.dk >

Chapter 1

Introduction And Motivation

Computer vision is a common field of research and is widely utilized in many social and industrial contexts, spanning from surveillance and hobby drones, to robot control and cosmetic image filters. In industry particularly, detection-related methods are widespread and object detection is a centrepiece of a wide range of applications such as human-computer interaction, automated surveillance, and many more [4]. Taking an object detection model outside the confines of a perfect-world assumption, and into the real world has often shown a significant drop in performance, especially because conditions change over time [4]–[7]. This change in conditions is called concept drift.

Concept drift itself is the root of many problems for systems not explicitly built to counter it. The concept may change over time, for a variety of reasons. To combat some of the problems associated with changing light conditions and visibility, research in the *Thermal InfraRed* (*TIR*) domain has been ongoing for a while. *TIR* has also made it easier to comply with the European *General Data Protection Regulations* (*GDPR*), due to the loss of facial details in the *TIR* domain. There has been ongoing research on using *TIR* instead of the usual *VIsual-optical Spectrum* (*VIS*) for this purpose [6], [8]–[10]. There have been researched multi-modal systems that use both *TIR* and *VIS* images [9], [10], and *TIR* images alone [7]–[9]. Both methods have their positives and negatives. While the multi-modal systems might be easier to get great performance and cover for the possible concept drift in *TIR* images it has the downside of being more expensive implementation-wise and data-wise and also still having the problem of not complying with *GDPR* inherently. With the use of *TIR* images only, the positives are that it inherently complies with *GDPR* laws and it is cheaper to implement as there is no need for *VIS* images.

This however comes with the downside that there has not been done much research on getting great performance in real-world scenarios especially over time as the appearance of the scene changes, causing concept drift [6]. This problem mostly lies in using qualitative thermography, where the image shows the relative difference of IR radiation in the *Field Of View* (*FOV*). If quantitative thermography were to be used the changing temperature could be modelled, and mitigated with a few downsides such as visual artefacts.
Quantitative thermography, therefore, could be seen as a solution, but the technology to construct such a device is substantially more complicated. For many tasks, the absolute temperature readings are redundant for the purpose of the camera. These reasons therefore do not justify the cost of using quantitative thermography, making qualitative thermography much more common in deployed systems [7].

With these facts, the problem can be described as such:

Thermal infrared images from qualitative thermography are subject to the problem of concept drift, wherein the appearance of the scene and

object(s) changes depending on the season, time of day, and weather conditions. These factors limit the potential of downstream computer vision algorithms, as they were often not intended to deal with this drift in the concept.

In this work, a preprocessing methodology was proposed by adapting the original *TIR* image to a more complex image that has stacked the output from a saliency map model and optical flow to mitigate concept drift. These preprocessed images are then used for pedestrian detection using YOLOv5 [3]. The *Long-term Thermal Drift* (*LTD*) dataset is used to train and evaluate the performance of the proposed model against the baseline. The pipeline for the preprocessed method is described and the outcome of using it has been discussed. Results of the proposed method show that there isn't any significant improvement in concept drift against the baseline, however further ablation studies have shown that this might be because the proposed method has superior detection ability. In the ablation study, it has been shown that *LTD* dataset isn't fully annotated in all frames and the proposed method is disproportionally penalized for finding pedestrians who are not annotated. It shows that there might be merit in adapting the original domain to a more complex domain to mitigate concept drift, but further research should be acutely aware of the missing annotations.

Chapter 2

Background

This chapter explains some background knowledge needed for this masters thesis

2.1 Thermal Imaging And Their Characteristics

This section will cover how thermal imaging systems work, describing thermal radiation and how the technology uses these to get an image.

2.1.1 Thermal Radiation

Infrared radiation is also often called thermal radiation [11] and is an electromagnetic radiation that lies between the *VIS* and microwaves and has some peculiar characteristics that make them suitable for several technical applications. These technical applications include contactless temperature measurement, night vision, thermal isolation of buildings, the temperature distribution of combustion processes etc. All objects emit thermal radiation with a temperature above 0K and are dependent on their wavelength and determined by their temperature [11], [12].

The spectral range of infrared is commonly divided into 5 subdivisions which can be seen from table 2.1

Subdivision	Wavelength
Near-infrared (NIR)	0.75-1.4µm
Short-wavelength infrared (SWIR)	1.4-4µm
Mid-wavelength infrared (MWIR)	3-8µm
Long-wavelength infrared (LWIR)	8-15µm
Far infrared (FIR)	15-1000µm

Table 2.1: The subdivision of the infrared spectral range [13]

Objects in the temperature range of 190K to 1000K emit radiation in the *Mid-wavelength infrared (MWIR)* and *Long-wavelength infrared (LWIR)* spectral range, and are often referred to as *TIR* [11], where at ambient temperature ($\sim$294K) the infrared maximum is 10µm [12]. The subdivision for thermal cameras are however different as some wavelengths are absorbed by molecules in the atmosphere and the transmittance rate is lowered which can be seen in figure 2.1

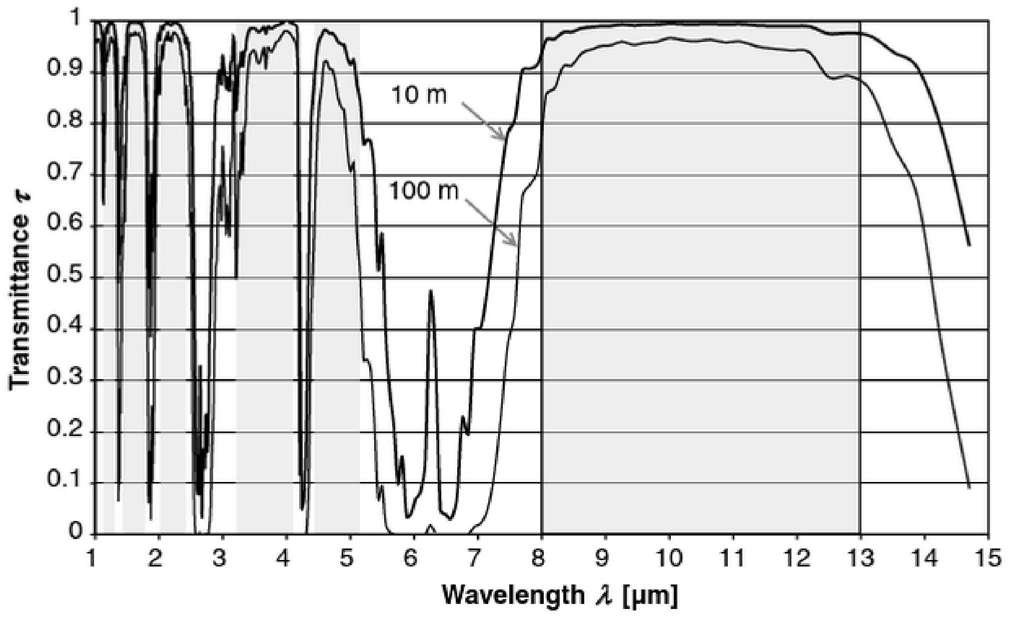


Figure 2.1: The typical transmittance rate in the atmosphere during summer in central Europe [12]

Due to low transmittance rates in some wavelengths, the spectral range for a thermal camera is further subdivided into the ranges shown in table 2.2

Subdivision	Wavelength
Short-wave	0.7-1.4μm
Mid-wave	3-5μm
Long-wave	8-14μm

Table 2.2: The subdivision of the infrared spectral range for a typical thermal camera [11]

2.1.2 Thermal Cameras

Thermal cameras operate relatively similarly to *VIS* cameras, the difference lies in that instead of taking in light as electromagnetic radiation it takes in infrared [14]. They both pass a lens system and other optical components such as the sensor before forming an image, here also both systems use a *Focal Plane Array* (*FPA*) as the sensor [11], [14]. The difference however changes much of how the imaging system should work and these differences are [14]:

1. Infrared is thermally radiated whereas an *VIS* imaging reflects the scattered light that is going to be captured.

2. The sensor for the imaging system must work within the infrared spectrum and transmit it. Therefore the optical materials needed are usually not made from glass but from semiconductors.
3. The detectors are not silicon-based as it is for a *VIS* system. Depending on the wavelength that is needed for the imaging system the materials are chosen from multiple types and materials with different techniques including cooled photo-electron as well as uncooled thermal detectors.
4. The pixel signal is not spectral filtered, meaning that the images are monochrome. With the use of image processing the image can either be shown as greyscale or false color images.
5. There are multiple temperature sensors within a quantitative infrared camera which with additional parameters input from the user such as the emissivity, ambient temperature, object distance, and relative humidity and with some calibration allows evaluation of the signal.

There are generally 2 main detector types used for thermal cameras, these being photon detectors and thermal detectors [11], [14]. Photon detectors convert the absorbed electromagnetic radiation directly into an electrical signal by the use of a positive-negative (p-n) junction, which is a method that allows electric current to only pass in one direction. This is done by having 2 semiconductors where one is positively charged with excess holes and the other negatively charged with excess electrons. This property can then be used to have the photon make electron-hole pairs which give an electrical signal. A p-n junction is a fundamental part of semiconductors and is how LED emits light, and the opposite with a photon detector. Thermal cameras however convert the absorbed radiation into thermal energy which causes a rise in temperature in the detector. Here the temperature-dependent electrical resistance in a bolometer is one of the most common forms used [11], [14].

A photon detector works in the *MWIR* range making the thermal contrast high which makes it very sensitive to small differences in temperatures. A photon detector also allows for a higher frame rate than thermal detectors. The biggest drawback of photon detectors is the need for them to be cooled to below 77K. With these requirements, there is a higher need for service and maintenance of the system making it much more expensive than thermal detectors [11]. Thermal detectors on the other hand work in *LWIR* range and mainly have two different detector types, these being ferroelectric detectors and microbolometers. Microbolometers have a higher temperature sensitivity, with the added advantage of also having a smaller pixel size allowing a higher spatial resolution. These make microbolometers have the largest market share [11].

2.2 Concept Drift

This section will cover what the concept drift problem is, how it is challenging in different fields, and where concept drift is seen.

Many data-driven systems that in some way want to predict a future value have a challenge that this prediction might not be accurate. This is further exacerbated when the data is ever-changing over time. This means that the system used to predict the future value might become less accurate as time passes and eventually not work at all. In machine learning, data mining and predictive analytics these changes in the data distribution that might happen over time are called concept drift[5]. In other fields like pattern recognition and signal processing, it is known as covariate shift or non-stationarity respectively [5].

In stochastic processes, there are stationary processes whose unconditional joint probability does not change over time. This means that the change over time does not alter the mean but remains constant however long that time is. This is how most standard data mining and machine learning algorithmic techniques have been researched and developed over time by assuming that the data is *Independent and Identically Distributed* (*IID*) [5]. An example of assuming data is *IID* could be that a qualitative *TIR* image showing a human in winter will be shown the same in summer even with changing conditions in the scene. This assumption is false as the temperature of a human might remain identical, but the temperature in the scene in which the image is taken has changed and therefore so has the image. This can be shown by figure 2.2 from the *LTD* dataset [6].

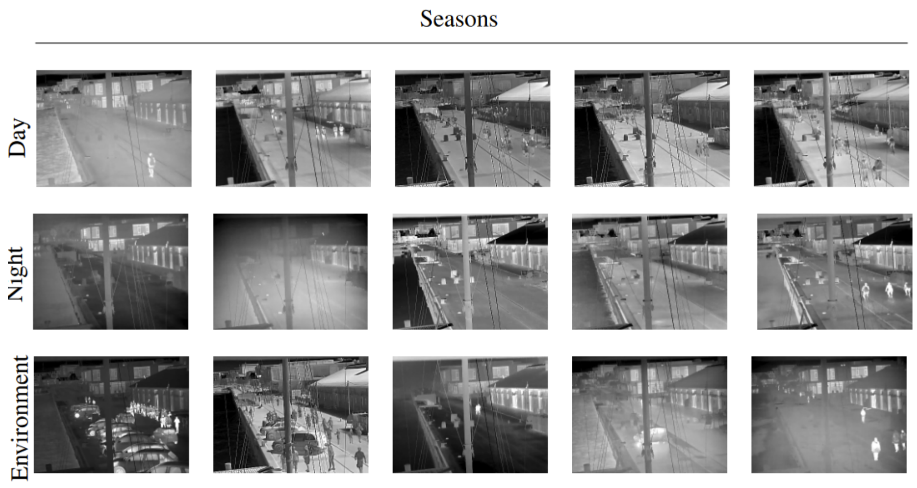


Figure 2.2: Examples of the data found in the *LTD* dataset, which shows for the day and night rows from left to right the changes for the data in February, March, April, June and August. The third row shows changes based on weather conditions and human activity [6].

To further define concept drift take the example of a data stream $\{(x_0, y_0), ..., (x_i, y_i), ..., (x_t, y_t)\}$ being collected until time t, and where x_i is the input and y_i is the output. The data (x_t, y_t) follows a joint probability distribution $p_{t_i}(x, y)$ where the concept is defined at time t. Concept drift can thus be defined by the eq 2.1 [15]:

$$p_{t_i}(x, y) \neq p_{t_{i+n}}(x, y) \tag{2.1}$$

Where the eq 2.1 shows that if the concept at any time t_i does not equal the concept at time t_{i+n} then concept drift must have occurred. On the flip side if they both equal each other then the same concept exists at different times.

Based on changes in the joint probabilities concept drift can be categorized into three classes [15]:

1. Class prior concept drift: $p_{t_i}(y) \neq p_{t_{i+n}}(y)$
2. Virtual concept drift: $p_{t_i}(x|y) \neq p_{t_{i+n}}(x|y)$
3. Real concept drift: $p_{t_i}(y|x) \neq p_{t_{i+n}}(y|x)$

For this project real concept drift will be focused on as the output prediction is conditioned on the input image, where a human in winter appears bright and through the change in seasonal time a human appears dark in summer or vice versa. This project will try to solve how the concept drift can be mitigated so these joint probabilities will be close together or ideally the same.

2.3 Domain Adaptation

This section covers the core principles and terminologies of domain adaptation.

Domain adaptation and domain translation are often used interchangeably. For clarity, domain translation will refer to any alteration attempting to imitate an existing domain, while domain adaptation will not attempt to conform to an existing domain directly. This includes methods in which the adapted domain could be shifted to a known domain afterwards.

Domain adaptation is the process of adapting data from the original domain to a desired target domain[16]. in the simplest form, it may be amplification of chosen features, such as brightness, or contrast in an image. Domain adaptation encompasses many methods of various complexity but is best known for their relevance in *Artificial Inteligence* (*AI*). Learning models are often highly susceptible to concept drift or lack sufficient data in the target domain for training. For both of these cases, domain adaptation can help mitigate the problem. Domain adaptation methods can be classified on many different properties, some of which are covered here.[17]

2.3.1 Homogeneity & Heterogeneity

A domain adaptation implementation can be classified based on the difference between the source domain and the target domain. A Homogeneous domain adaptation method is any method in which the source domain shares many properties and features with the target domain. Examples of this are image-to-image methods, such as camera filters and edge detection. Heterogeneous domain adaptation methods encompass all methods in which the target domain is significantly different from the source domain. These are more loosely defined, as the adaptation can still be in the same general domain, such as an image-to-image method, but can also be more abstract methods such as text-to-image, speech-to-text or similar. Heterogeneous methods are typically harder to develop, due to the added complexity and abstraction. Many heterogeneous methods are *AI* based, due to the complexity of the models being nearly impossible to achieve with handmade methods.

2.3.2 One- And Multi-step

Domain adaptation methods can be developed with intermediary domains. these domains are often used to simplify the method, by separating it into multiple steps, a "multi-step" model. The appeal of this lies in the potential for the procedure to be modular and more explainable, as the output of each state can be observed. Alternatively, one-step models can be used, which are commonly found in simpler methods or runtime-optimized implementations. These translate directly from the source domain to the target domain.

2.3.3 Supervision

As domain adaptation tasks are often based on *AI*, the data needed for training comes with various properties regarding labelling. A fully labelled dataset is considered supervised. If the labelling is only partial, it is considered semi-supervised. This affects the usable methods for training. Furthermore, there is also weakly supervised supervision where part of the labels are incorrect. This is very troublesome, as it will drastically affect the accuracy of the *AI*. Lastly, there is unsupervised, which is for data without labels. While hard to use, this is sometimes a better case than weakly supervised. It requires unique solutions and can be very time-consuming to train.

2.3.4 Common Implementation Approaches

When developing a domain adaptation method, there are various common approaches. The most common, and practical is to look for domain invariant features, ie. anything that is consistent between both the source and target domain. If an AI can learn these features, it can translate them to the target domain. These are critical for many unsupervised methods, as it is the only consistency available to it. Another common approach is GAN models, in which the generator attempts to translate from source to target domain, and the discriminator compares the output of the generator with a data element already in the target domain. It attempts to guess the correct one and evaluates how certain it is. This is then fed back to the generator such that it can learn from the mistakes it made.

2.4 Summary

Concept drift is a challenge that comes up in many data-driven systems in some way when it wants to predict a future value from previous data. This is especially the case in machine/deep learning systems because they have assumed the data to be *IID*, which means that the concept doesn't change over time. This project will focus on real concept drift as the concept of a person changes based on the seasons because of how thermal cameras operate. Thermal cameras operate similarly to *VIS* where they pass a lens system and use *FPA* as the sensor. The difference between how a *VIS* and a *TIR* camera works makes it so that concept drift appears differently between the systems. The characteristic of thermal cameras, which enables them to operate effectively in low-light conditions by utilizing the infrared spectrum, renders them superior for capturing nighttime images. However, their reliance on this spectrum means that variations in temperature within the scene can pose a challenge, particularly if the appearance of objects is expected to remain static. This is how concept drift is a challenge for thermal cameras. One way to mitigate this is to use domain adaptation, which for clarification in this work refers to adapting data from the original domain to another domain eg. amplifying chosen features.

Chapter 3

Related Works

This chapter describes pre-existing works of relevance to the project. Since many solutions for object detection include domain adaptation, the chapter will be split into two sections. This will be object detection in *TIR* without domain adaptation and domain adaptation in the context of *VIS* and *TIR*.

3.1 Object Detection

Object detection is one of the most widely used downstream tasks for *TIR* images. A major focus of object detection lies in its application for pedestrian detection, as well as for identifying related objects such as bicycles and cars. One reason for this emphasis on pedestrians is their integral role in security, tracking, emergency response, and video surveillance, both in private and public settings. Pedestrian detection is also where many of the problems for concept drift arise as the thermal radiation from pedestrians doesn't change daily or seasonally over time while the relative temperature in the scene changes.

3.1.1 Object Detection In Thermal Imagery

Object detection for *TIR* can be divided into two categories: the first involves object detection solely using *TIR* images, while the second entails object detection using both *TIR* and *VIS* images this is often called multispectral object detection.

Multispectral Object Detection

König et al. [18] presented a multi-spectral *Region Proposal Network* (*RPN*) that uses a pre-trained VGG-16 [19], where these proposals are then evaluated using a boosted decision trees classifier to reduce potential false positive detection. This effectively fused the information from the two spectra.
Guan et al. [20] and C. Li et al. [21] both introduced a multi-spectral model that is illumination-aware and they proposed an illumination-aware weighting mechanism to adaptively weight the confidences from the two modalities to merge them. Following the illumination-aware model C. Li et al. [22] also introduced a unified *Convolutional Neural Network* (*CNN*) fusion architecture that consists of a multi-spectral *RPN* and a following multi-spectral classification network to improve performance.
R. Li et al. [23] proposed a multi-spectral cross-modal pedestrian detector that used data from *VIS* and *TIR* to enhance performance by maintaining the unique characteristics of both modalities while extracting and supplementing homogeneous features from each other.

Thermal Images Only Object Detection

Galarza-Bravo et al. [24] presented an architecture based on Faster R-CNN using two independent *RPN* focusing on near or far-away pedestrians. Chen et al. [25] introduced an attention-guided encoder-decoder *CNN* to generate multi-scale features and an attention module to re-weight them. Dai et al. [26] proposed a model dubbed TIRNet which is comprised of a Customized *Single Shot Detector* (*SSD*) with a *Residual Branch* (*RB*)

Cao et al. [27] improved upon RefineDet by designing a dual-pass fusion block to fuse features from different levels and a channel-wise enhance module that assigns different weights to different channels.
Patel et al. [28] employed a more computationally efficient object detector for night-time images using a Depthwise Deep Convolutional Neural Network and improved accuracy using Tversky and *Intersection over Union* (*IoU*) as loss functions.
Heo et al. [29] proposed an adaptive boolean-map-based saliency that created clearer silhouettes which was used to boost the features of pedestrians based on the season.
Ghose et al. [30] further researched saliency maps using deep learning where they augmented thermal images with saliency maps of pedestrians and used them as an attention mechanism.

Summary

While multi-spectral doesn't directly mitigate concept drift in the thermal domain it does however sidestep it by having the possibility to use the visual domain. This however has the downside of being more expensive implementation-wise and data-wise while also not complying with *GDPR* laws. There may be some promise in possibly using saliency maps to mitigate concept drift as it creates a new domain in which pedestrians can be highlighted. Exemplified by the implementation of Heo et al. [29] where saliency maps were used to boost the features depending on the season. However, the implementation is done with an algorithm so it might be useful to explore this with a deep learning model. There seems to however be a lack of studies that focus on mitigating concept drift in the thermal domain directly, especially long term. This is a motivating factor to research how to mitigate concept drift for pedestrian detection in the thermal domain.

3.2 Domain Adaptation And Translation

Based on the shortcomings presented above, many domain adaptation and translation methods exist in an attempt to mitigate concept drift, and improve overall accuracy. This section will cover a variety of domain adaptations, not limited to *VIS* and *TIR*, to highlight the extent of existing methods, and unaddressed concept drifts

3.2.1 Domain Adaptation And Translation In Thermal Imagery

Kieu et al. [31] did preliminary work on the performance of common pedestrian detection methods for *VIS* data, when adapted to utilize *TIR* data. The performance surpassed the original in bad weather and lighting conditions and matched them under ideal conditions. Following this, Kieu et al. [32] expanded on their work, by further modifying the training data, by adding day/night metadata, obtained from a classifier, further improving performance. Finally, Kieu et al. [33] proceeded to develop a least-squares GAN-based method to generate *TIR* data from *VIS* data, achieving state-of-the-art performance.

Munir et al. [34] implemented a method to do domain adaptation, using style consistency. They found that low-level features provided better performance for object detection tasks.
Akkaya et al. [35] proposed a method to achieve domain adaptation, without image pairs for the source and target domains, utilizing an adversarial network to make a general model for both domains, with state-of-the-art performance.
Zoetgnande et al. [36] proposed a combination of optical flow, a domain classifier, an activity recognition classifier and an I3D-based feature extractor to make a generalized representation between *VIS* and *TIR* for video. Their findings show good potential for this methodology
Johansen et al. [7] attempted to condition and guide two object detectors to become weather-aware to increase the performance of the object detectors in the thermal domain. The results showed that such an approach may not necessarily be ideal.

3.2.2 General Domain Adaptation And Translation

Lou et al. [37] proposed a cross-domain object detection, utilizing data from Synthetic Aperture Radar (SAR) and *VIS*, the result is a significant improvement for SAR-based methods, as the *VIS* provides additional context to the 3D data from the radar. Li et al. [38] proposed utilizing auto-encoders to learn a general representation of an object in many similar source domains, to address overfitting. The results show a general representation which works on most similar domains, at state-of-the-art performance.
Murez et al. [39] proposed a method where the source and target domain are mapped to an intermediary domain, in which they are indistinguishable. The method can then map back to either domain, resulting in a generalized representation of both domains. The performance is state of the art.
Zhou et al. [40] proposed an alternative approach to do cross-domain object detection. The method is based on a student-teacher model.

Summary

Overall, previous works have shown that *TIR* is a viable domain for vision tasks, and with the advantage of obscured identifying features, it is more suited to comply with *GDPR*. The works have shown extensive exploration of various methods and forms of concept drift. Consistent for all these works is the relatively short timescale for their solutions, with only a few reaching beyond day/night, making them susceptible to long-term drift from conditions such as seasonal temperatures and weather. This paper will be centred around this void in the existing research and will be utilizing methods inspired by previous works.

Chapter 4

Problem Statement

Concept drift is a challenging problem in many fields, which is also the case for object detection where e.g. the concept of a person can change depending on occlusion or the thermal signature for thermal images. A way to mitigate concept drift is to use domain adaptation, which adapts the data from the original domain to another domain eg. by amplifying chosen features. Doing this can mitigate concept drift found in the representation. With prior knowledge gained from how related works in chapter 3 has tried to solve problems related to concept drift by employing different methods like saliency maps [29], [30], channel weighing [27], or using domain adaptation methods that changes the domain of the image by combining different domains like optical flow [36]. With this knowledge together with the background knowledge 2 concept drift in thermal images may be mitigated by extracting information from the original domain and adapting it to a more complex domain. This leads to the following problem statement:

Can seasonal concept drift in thermal images be mitigated by adapting the original domain to a complex representation for pedestrian detection

The hypothesis for this is that by creating a more complex domain from the original by employing different methods it might be possible to mitigate concept drift by introducing new features that aren't found by the object detector.

Chapter 5

Technical Analysis

This chapter explains the dataset used in this work, as well as information about different methods and models that can be used to create new domains for the image.

5.1 Dataset

This section will describe the dataset used for this work.

5.1.1 LTD-TIR

The *LTD TIR* data [6] consists of 2-minute clips with a resolution of 288×384 selected every 30 minutes in a day for 8 months for a total of 298 hours. The starting point of the data is May 2020 until September 2020 with a second part from January 2021 up to May 2021. The footage was recorded on the harbor front in Aalborg, Denmark with the approximate longitude and latitude coordinates given as (9.9217, 57.0488) [6]. The camera used to capture the image is a Hikvision DS-2TD2235-25/50 thermal camera which captures *LWIR* saved in the mp4 format as 8-bit uncalibrated grayscale. All footage is stationary and is taken from a birdseye view position, as this dataset focuses on long-term drift, and therefore attempts to eliminate other factors. An example of images can be seen in figure 5.1. The dataset is populated with extensive metadata, covering GPS position, time of day, temperature, humidity, rain, wind direction, wind speed, sun radiation and sunshine hours. The publically available dataset can be found **here**. For this work, an extended dataset was shared by the supervisor which consists of 844638 annotated *TIR* images split into a train, test, and validation set.

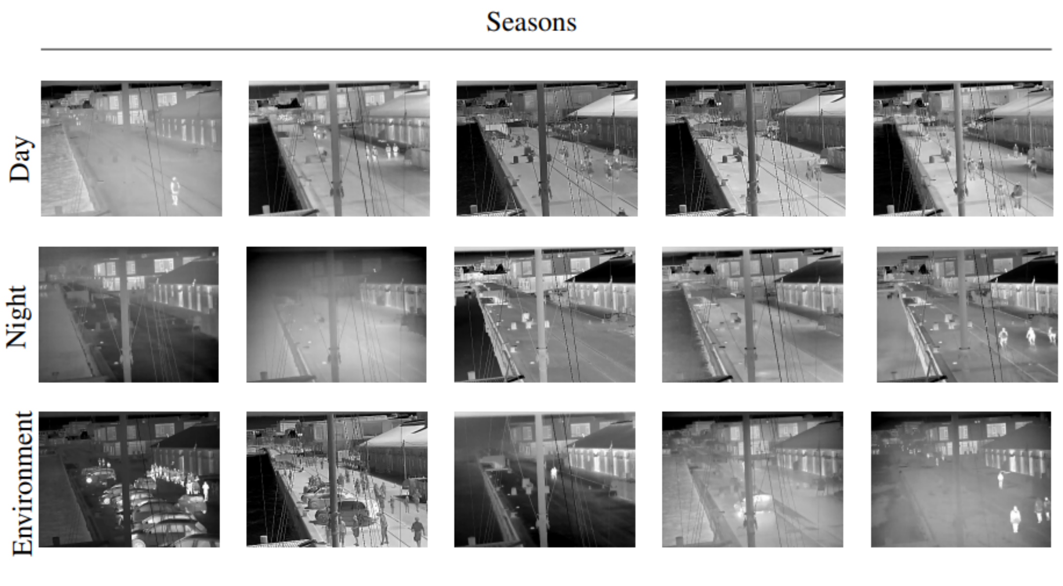


Figure 5.1: Example images the original LTD paper [6]

5.2 Saliency Maps

This section will cover what saliency maps are, and how they are used as well as discuss PiCANet, R^{3}Net, and U^{2}Net which are of interest.

A saliency map is an image that maps the importance of pixels for the human visual system, the goal of a saliency map is to display the pixels in an image that a person pays attention to. This is usually done with a greyscale image where the brightness of the image is proportional to the attentional priority when viewing a scene. The idea of saliency maps has then been adopted as a sort of biomimicry which mimics how the human visual system efficiently processes important information from the visual world [41], [42]. To create saliency maps there have been proposed many different saliency detection methods which can improve the efficiency of computer vision systems and have shown results in image segmentation, object recognition, visual tracking, gaze estimation, action recognition, etc [42]. An example of saliency maps from the DUT-Omron dataset [43] can be seen in figure 5.2.

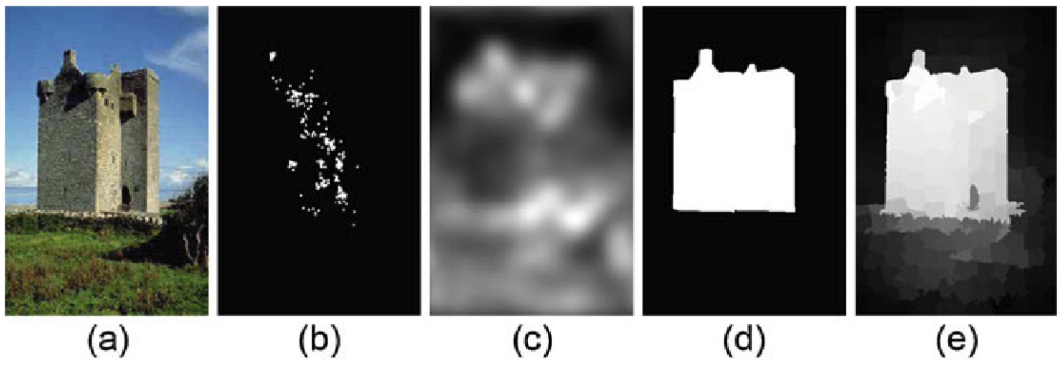


Figure 5.2: (a) Sample image from the DUT-Omron dataset [43]. (b) Ground truth for eye fixation prediction. Each white spot is an eye fixation position for a participant in a free-viewing experiment. (c) Saliency map by a state-of-the-art method for eye fixation prediction [44]. (d) Ground truth for salient region detection. (e) The saliency map using Saliency Detection via Absorbing Markov Chain [45]

5.2.1 PiCANet

PiCANet by Liu et al. [46] is one of the models used by Ghose et al. [30] as an attention mechanism and improved the performance for pedestrian detection using only thermal images. PiCANet is a pixel-wise contextual attention network which generates an attention map for each pixel which corresponds to its relevance in its region. This is done by using two pixel-wise attention modes, which are [46]:

- Global attention: generates attention over the whole feature map

- Local attention: generates attention on a local region centred around the corresponding pixel

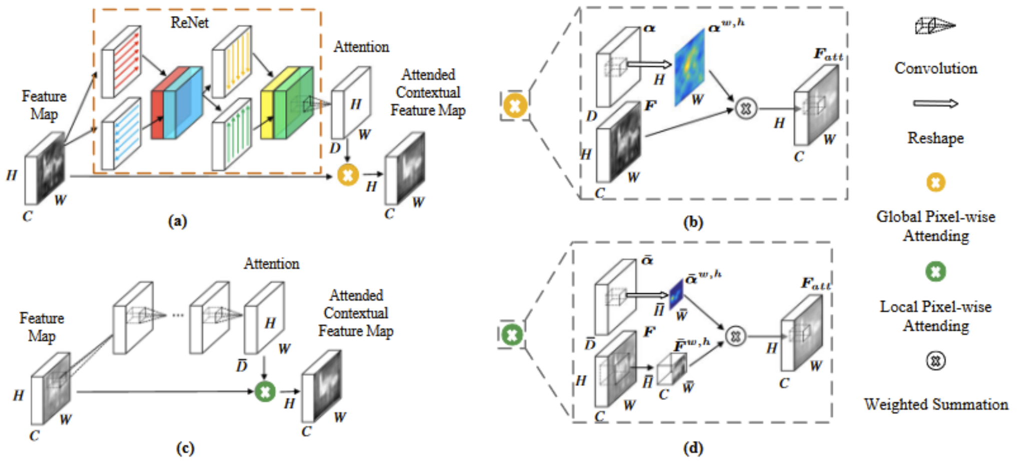

ReNet
Attention
Attended Contextual Feature Map
Feature Map
(a)
(b)
(c)
(d)
Convolution
Reshape
Global Pixel-wise Attending
Local Pixel-wise Attending
Weighted Summation

Figure 5.3: (a) The architecture for global PiCANet. (b) Illustration of the detailed global attending operation. (c) The architecture for the local PiCANet. (d) Illustration of the detailed local attending operation [46]

The PiCANet for global attention can be seen in figure 5.3(a), it works by using a ReNet model [47] seen in the orange dashed box which originally uses 4 *recurrent Neural Network* (*RNN*)s to sweep horizontally and vertically in both directions to get the global context. In the PiCANet implementation, however, they use a bidirectional *Long Short-Term Memory* (*LSTM*). First, they concatenate both horizontal directions and then concatenate both vertical directions, resulting in a pixel being able to memorize the context of the horizontal and vertical axis. By doing these sweeps and blending the contexts from the four directions the information of each pixel is propagated to all other pixels thus obtaining global context. This blended feature map is transformed into D channels with a vanilla convolutional layer, where $D = W \times H$ [46]. The attained feature vector for each pixel is then used in the global attending operation seen in figure 5.3(b), where it first is normalized via a softmax function to generate the global attention weight $a^{w,h}$ and finally the features at all locations are weighted summed with the global attention weights to get the contextual feature F_{att} The PiCANet for local attention can be seen in figure 5.3(c), where instead of using ReNet to get the local feature map. The architecture instead uses several convolutional layers on the feature map on the local neighbouring context centred at the pixel. It then follows the same pattern as the global attention PiCANet making a feature vector, normalizing via a softmax, and making a weighted sum. However, the difference is that instead of doing it on a global scale it does it locally denoted by a line over the variable in figure 5.3c and d [46].

A U-Net [48] architecture is used to integrate the PiCANets hierarchically for salient object detection which can be seen in figure 5.4(a). In this U-Net, they use VGG-16 [19] as the backbone and the encoder of the U-Net is a *Fully Convolutional Network* (*FCN*). The decoder for the network uses the idea of skip connections in U-Net but also embeds the global and local PiCANets. Since the global PiCANet requires an input feature map to have a fixed size it has therefore been fixed to 224 × 224 [46].

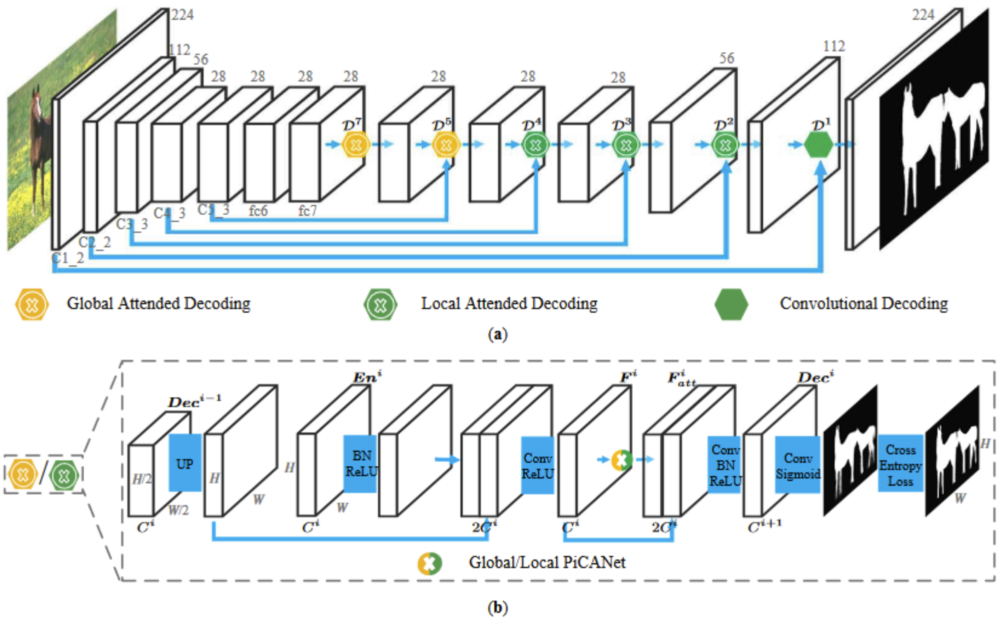


Figure 5.4: (a) The PiCANet U-Net architecture for saliency object detection with the VGG 16-layer backbone. It only shows the skip-connected encoder layers of the VGG network. "C" means convolution and $D*$ means decoding module. The spatial sizes are marked above the cuboids representing the feature maps. (b) Illustration that represents the attended decoding module. En^i denotes a convolutional feature map from the encoder module and $Dec*$ denotes a decoding feature map. F^i denotes a fusion feature map while F^i_{att} denotes the attended contextual feature map. "Up" denotes upsampling [46].

Figure 5.4(b) shows the attended decoding module, where the decoding feature map is generated by fusing an intermediate encoder feature map with the preceding decoding feature map. This is done by upsampling the preceding decoding feature map using a deconvolutional layer with bilinear interpolation and then concatenating it with the encoder feature map before using a convolutional and *Rectified Linear Unit* (*ReLU*) layer. With the fused feature map, a global or local PiCANet is used to obtain the attended feature map and then it is concatenated with the fused feature map.

To create the decoded feature map the concatenated attended feature and fused feature map are fused by using a convolutional, batch normalization, and *ReLU* layer [46].

5.2.2 R^{3}Net

R^{3}Net [49] is the other model used by Ghose et al. [30]. It uses a *Residual Refinement Block* (*RRB*) to learn the residuals between the ground truth and the saliency map at each recurrent step. Alternatively, it uses a recurrent residual refinement network (R^{3}Net), which employs a series of *RRB*s to refine the saliency maps utilizing low-level and high-level features progressively.

The R^{3}Net as seen in figure 5.5 starts with using ResNeXt [50] as the feature extraction network that produces a set of feature maps with low-level and high-level semantic information at different scales. The low-level semantic information extracts the fine structures of salient regions, while the high-level semantic information extracts the salient objects. The first three feature maps are then upsampled and concatenated to form the low-level integrated features denoted as L and the 4th and 5th layers are upsampled and concatenated to form the high-level integrated features denoted as H [49].

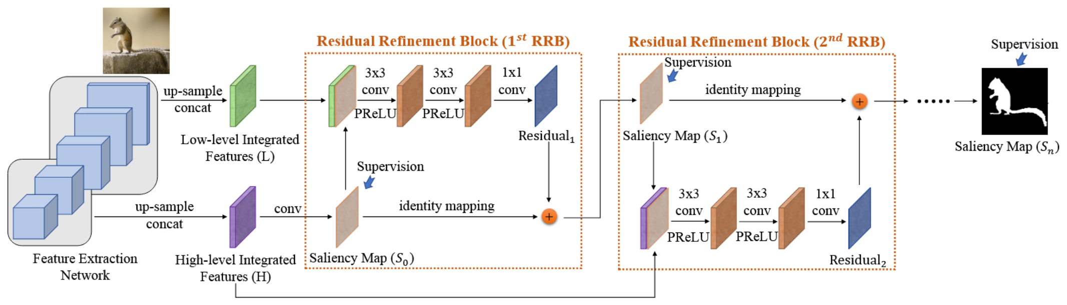


Figure 5.5: Illustration of the R^{3}Net architecture. The feature maps at the first 3 layers are upsampled and concatenated to generate low-level integrated features denoted as L. The feature maps at the last 2 layers are upsampled and concatenated to form the high-level integrated features denoted as H. The initial saliency map is generated using H, which is then sequentially refined by the *RRB* [49].

The network first predicts an initial saliency map denoted as S_0 from the high-integrated features capturing the locations of salient objects, but neglecting fine detail. This initial saliency map is then refined using the residuals from the L obtained by using a feature fusing network that consists of 3 convolution layers and 3 *Parametric Rectified Linear Unit* (*PReLU*) activation functions. This refinement of the initial saliency map S_0 results in the saliency map S_1 which contains a large number of non-saliency cues introducing non-salient regions into S_1. S_1 is therefore further refined by using the residuals from the H similar to how the residuals from the L were used to refine S_0. The refinement of S_1 using the residuals from H eliminates non-saliency details that are not located in the semantic salient regions. To further refine the saliency prediction multiple sequences of *RRB*s that alternatively incorporate L and H several times to get the final saliency map [49].

5.2.3 U^{2}Net

U^{2}Net [1] is a state-of-the-art salient object detector inspired and built upon U-Net [48] and does not need a backbone. It uses *ReSidual U-blocks* (*RSU*) to increase the contextual information from different scales and the depth of the whole architecture without significantly increasing the computational cost.

The structure of the *RSU* is inspired by U-Nnet, which can be seen in figure 5.6 where L is the number of layers in the encoder, C_{in} and C_{out} denote the input and output channels and M denotes the number of channels in the internal layers of *RSU* [1].

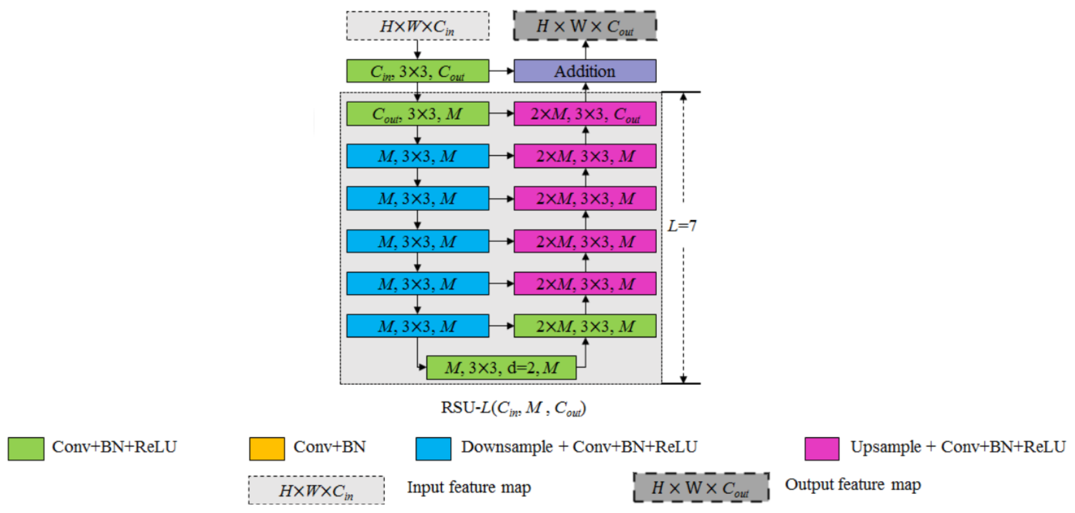


Figure 5.6: The structure of the U-block *RSU* for U^{2}Net [1]

The *RSU* mainly consists of three components these being:

1. An input convolution layer
2. A U-Net-like symmetric encoder-decoder structure
3. A residual connection

The input convolution layer is a plain convolution for local feature extraction and transforms the input feature map to an intermediate feature map $\mathcal{F}_1(x)$ with the output channel C_{out}. The U-net-like encoder-decoder structure with a height of L takes the intermediate feature map as its input and learns to extract and encode the multi-scale contextual information $\mathcal{U}(\mathcal{F}_1(x))$. U represents the U-Net structure shown in figure RSU, and a larger amount of layers in the RSU results in more pooling operations, a larger range of receptive fields, and richer local and global features. Finally, the residual connection fuses the local features extracted from the input convolution layer together with the multi-scale features from the U-Net-like encoder-decoder structure by summation $\mathcal{F}_1(x) + \mathcal{U}(\mathcal{F}_1(x))$ [1].

The main architecture for U^{2}Net uses a two-level nested U-structure where the top level is a big U-structure that consists of 11 stages filled by RSUs as the bottom level of the U-structure. The main architecture of U^{2}Net can be seen in figure 5.7 [1].

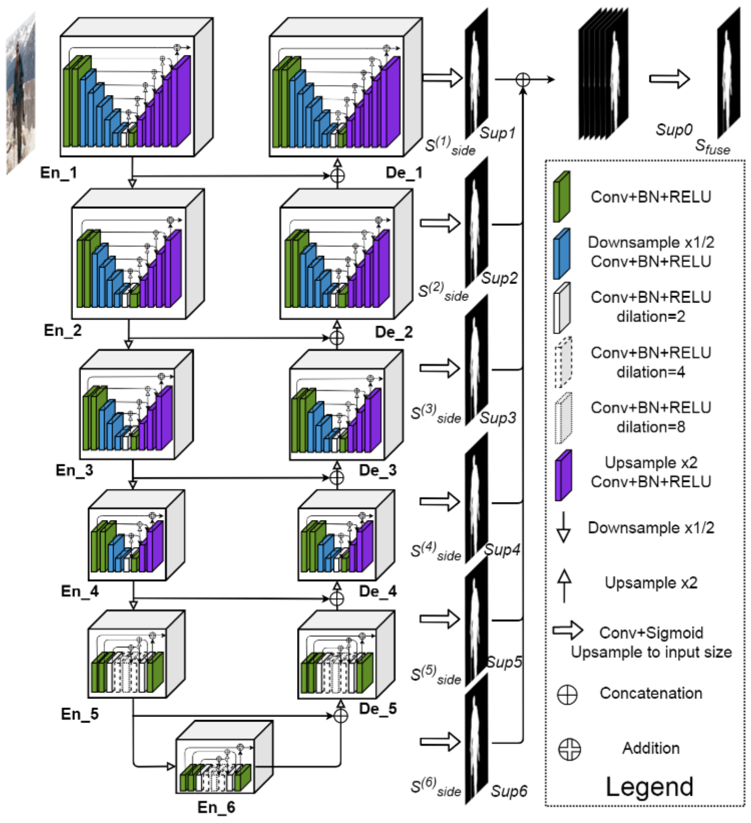


Figure 5.7: The main architecture of U^{2}Net. The main design is the architecture like U-Net [48], where each stage for the encoder and decoder uses the U^{2}Nets RSU [1]

As illustrated in figure 5.7 the main architecture of U^{2}Net consists of 3 parts these being:

1. A six stage encoder
2. A five stage decoder
3. Saliency fusion module attached to the decoder stages and the last encoder stage

The encoder stages 1-4 uses RSU-7, RSU-6, RSU-5, and RSU-4 where the numbers denote the layers in the RSUs. The layers are configured according to the spatial resolution of the input feature maps, and therefore the encoder stages 5 and 6 are configured differently. The RSU for the encoder stages 5 and 6 are configured differently because the resolution of the feature maps in them is relatively low and therefore further downsampling them leads to contextual loss. For this reason, the encoder stages 5 and 6 are configured to use RSU-4F, where "F" denotes that the RSU is dilated and that pooling and upsampling operations are replaced with dilated convolutions. This results in intermediate feature maps of RSU-4F having the same resolution as its input feature maps [1].

The decoder stages are symmetrical with the decoder stages with respect to encoder 6. This means that the decoder stages 1-4 uses the same RSUs as the encoder stages 1-4, and the decoder stage 5 uses RSU-4F. Each decoder stage also takes the output of the concatenation between the upsampled feature maps from the previous stage and those from its symmetrical encoder stage as its input [1].

The saliency fusion module is used to generate saliency probability maps. This is done by first generating side output saliency probability maps S_{side}^{stage} for all stages by using 3×3 convolution layer and a sigmoid function. The logits of the side output saliency maps are then upsampled to the input image size before fusing them with a concatenation operation followed by a 1×1 convolution layer and a sigmoid function to generate the final saliency probability map $\mathcal{S}_{fuse}$ [1]. U^{2}Net [1] is a supervised model that tries to minimize the sum of all side output saliency probability maps which can be seen in figure 5.7. This is done using binary cross-entropy between the predicted side output saliency probability maps and the ground truth mask.

5.2.4 Performance Evaluation

The performance of the models discussed can be seen in table 5.1 and 5.2 for several evaluation methods and datasets. All the discussed models have an emphasis on finding salient objects by differentiating them using global and local feature information. The evaluation metrics seen in table 5.1 and 5.2 are the maxF_β-score, *Mean Absolute Error* (MAE), weighted F_β^ω-score, *Structure-measure* (S_m).

The maxF_β-score evaluates the precision and recall of the system, where the β has been set to 0.3 weighing precision more than recall and the maximum is reported. The *MAE* denotes the average per-pixel difference between a predicted saliency map and its ground truth mask. The weighted F^{ω}_{β}-score is complementary to the maxF_{β}-score for overcoming unfair comparison. The S_m is the structure similarity of the predicted non-binary saliency map and the ground truth [1].

Method	DUT-OMRON				DUTS-TE				HKU-IS			
	maxF_{β}	MAE	F^{ω}_{β}	S$_m$	maxF_{β}	MAE	F^{ω}_{β}	S$_m$	maxF_{β}	MAE	F^{ω}_{β}	S$_m$
PiCANet	0.794	0.068	0.691	0.826	0.851	0.054	0.747	0.851	0.921	0.042	0.847	0.906
R^{3}Net	0.795	0.063	0.728	0.817	0.828	0.058	0.763	0.817	0.915	0.036	0.877	0.895
U^{2}Net	0.823	0.054	0.757	0.847	0.873	0.044	0.804	0.861	0.935	0.031	0.890	0.916

Table 5.1: Comparison between PiCANet, R^{3}Net, U^{2}Net on DUT-OMRON,DUTS-TE, and HKU-IS in terms of maxF_{β}-score (↑), MAE(↓), weighted F^{ω}_{β}-score(↑), and S_m(↑). Red, Green, and Blue indicate the best, second, and worst performance [1].

Method	ECSSD				PASCAL-S				SOD			
	maxF_{β}	MAE	F^{ω}_{β}	S$_m$	maxF_{β}	MAE	F^{ω}_{β}	S$_m$	maxF_{β}	MAE	F^{ω}_{β}	S$_m$
PiCANet	0.931	0.046	0.865	0.914	0.856	0.078	0.772	0.848	0.854	0.103	0.722	0.789
R^{3}Net	0.934	0.040	0.902	0.910	0.834	0.092	0.761	0.807	0.850	0.125	0.735	0.759
U^{2}Net	0.951	0.033	0.910	0.928	0.859	0.074	0.797	0.844	0.861	0.108	0.748	0.786

Table 5.2: Comparison between PiCANet, R^{3}Net, U^{2}Net on ECSSD, PASCAL-S, and SOD in terms of maxF_{β}-score (↑), MAE(↓), weighted F^{ω}_{β}-score(↑), and S_m(↑). Red, Green, and Blue indicate the best, second, and worst performance [1].

Table 5.1 and 5.2 show that U^{2}Net is the best-performing model in nearly all metrics. Therefore U^{2}Net will hence be used to create a new domain as part of our solution.

5.3 Preliminary Saliency Maps Expirements

For this work, a preliminary experiment was conducted to compare the outputs from U^{2}Net [1] and PiCANet [46]. This is to evaluate the difference between the two models. The saliency map output from PiCANet has a changed size and aspect ratio which can be seen in figure 5.8. There is quite a difference in the output when comparing the output from PiCANet against the output from U^{2}Net using the same image which can be seen in figure 5.9. While both used the bounding box as a mask for training, U^{2}Net has a saliency map which resembles a bounding box, whereas PiCANet more closely tries to make a mask around the detected pedestrian. There might be some merit in using PiCANet instead, but for this work, U^{2}Net will continue to be used as the performance was superior as mentioned previously in the previous section 5.2.4.

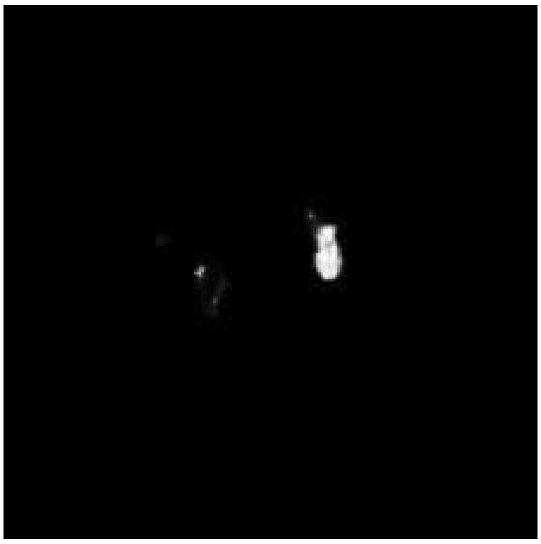


Figure 5.8: The saliency map obtained by PiCANet

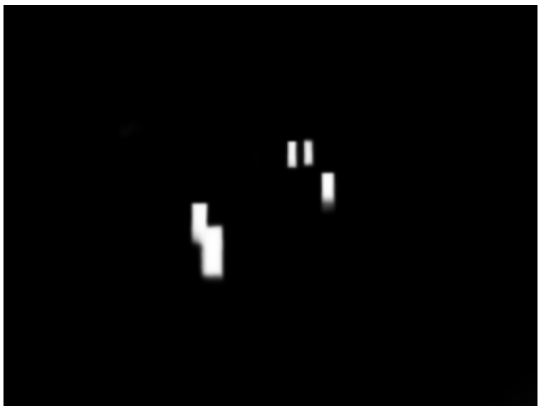


Figure 5.9: The saliency map obtained by U^2Net

5.4 Optical Flow

5.4.1 Lucas-Kanade

The Lucas-Kanade method is perhaps the most rudimentary optical flow method and a great introduction to their concept. It is built to detect movement based on small patches of features, attempting to match that same patch to nearby pixels in the next frame, using linear equations. This method relies on planar motion, and as such, may struggle with complex movement of 3D objects[51]. The method itself is not perfect, and as such, more than one region may match the feature it is attempting to locate. In this case, it will attempt to minimize the difference between the frames, as it is built with smaller motion in mind. In some cases, other optimization methods may apply. It is common for Lucas-Kanade to be implemented in a pyramidal structure, meaning the implementation will be run with different window sizes, such that both big and small features are tracked [52]. This is to ensure that the desired motion is detected independent of size. Based on the nature of how Lucas-Kanade is built, The computation time should increase quadratically, as the search window does, to accommodate the expected deviation. Because of this, along with the existence of better performing models, Lucas Kanade is not preferred for most tasks.

5.4.2 Gunnar Farnebäck

The Gunnar Farnebâck method is an addition to existing optical flow models, meant to improve performance. The model itself is based on a very simple principle, named pattern recognition. It observes the motion of objects, or groups of objects, in the optical flow over the entire frame, and attempts to predict their movement pattern. This is done by attempting to fit their movement to a polynomial, from which it can extrapolate an expected position [2]. this polynomial Will be continuously optimized as long as the tracked element is present, and is often very close to the ground truth, after having had the option to observe for a few frames. While this in and of itself is not able to do optical flow, it is useful for improving the accuracy of tracking, as well as optimizing runtime, as optical flow methods may narrow their search to a much smaller region than otherwise. It is worth mentioning, that as the Gunnar Farnebâck method relies on polynomials, it can model very complex movement patterns if need be.

5.4.3 CNN Based Methods

CNN-based optical flow models are A step up from traditional methods, utilizing neural networks to predict motion between frames. They utilize handcrafted features, trained features or a mixture of the two, to learn hierarchical features, attempting to learn complex patterns in the data. Many methods, such as FlowNet [53] and RAFT [54] utilize this technology, as it has achieved higher accuracy and efficiency compared to traditional methods. It is important to note that many models require a ground truth to effectively train optical flow, to avoid the model learning something else.

5.4.4 Summary

All three methods provide potential gains and consequences. Lucas Kanade benefits from its simplicity and is unlikely to suffer much from concept drift. Despite this, it is expected to underperform in comparison to others. Gunnar Farnebäck is expected to perform better overall, due to the prediction of movement, but is expected to fall short of CNN-based methods, by a noticeable margin. CNN-based methods are expected to outperform all other methods but may require specific training data, which may not always be feasible to obtain.

5.5 Object Detection Models

This section will cover the object detection model architectures Faster R-CNN and YOLOv5 which were used in the paper Seasons in Drift: A Long-Term Thermal Imaging Dataset for Studying Concept Drift [6] that published the baseline dataset *LTD*.

5.5.1 Faster R-CNN

Faster R-CNN [55] is a two-stage model object detector and the architecture is built with its own *RPN* and uses the classification head from Fast R-CNN [56].

The authors for Faster R-CNN [55] explored utilizing the Zeiler and Fergus model [57], in conjunction with VGG-16 [19] for the *RPN*. Since VGG-16 was the best-performing model the explanation of the *RPN* will be based on that. To generate the region proposals a small network with a sliding window with a spatial resolution of $n \times n$ (n=3) is slid over the convolutional feature map output from the last convolution layer in VGG-16. Each sliding window is mapped to a lower-dimensional feature which is 512 dimensions for VGG-16 and then activated with *ReLU*. The feature obtained from this is then fed into two fully connected layers which are the box-regression layer and the box-classification layer which can be seen in figure 5.10

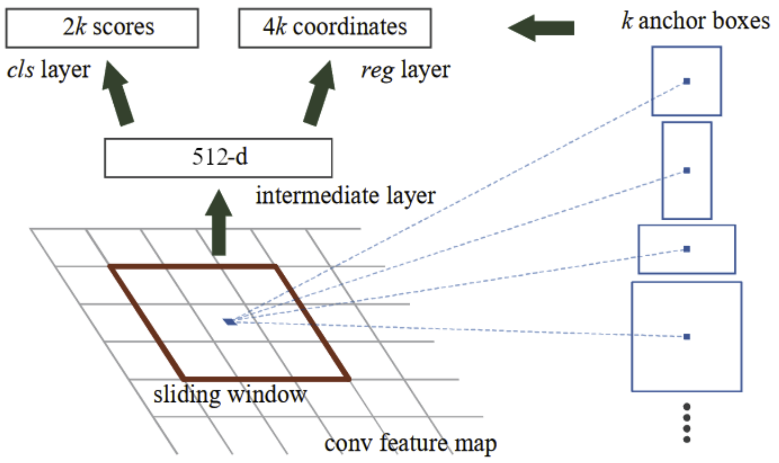


Figure 5.10: The region proposal network for Faster R-CNN. Edited to show that VGG-16 has 512 dimensions [55]

As seen in figure 5.10 at each sliding-windows location the *RPN* simultaneously predicts multiple region proposals, where the maximum number of proposals for each location is denoted as k. The k proposals are parameterized relative to k reference boxes called anchors, where an anchor is centred at the sliding window and associated with a scale and aspect ratio. By default, the 3 scales and 3 aspect ratios are used resulting in $k = 9$ anchors at each sliding position. The box-regression layer encodes the coordinates for k anchor boxes and outputs $4k$ coordinates, and the classification layer estimates the probability of each proposal having an object or no object with its output being $2k$ scores. The *RPN* has WHk anchors in total for a feature map size of $W \times H$.

With the proposed regions from *RPN* the Fast R-CNN [56] detector uses those proposed regions as an attention mechanism. The full architecture of Faster R-CNN [55] can be seen in figure 5.11

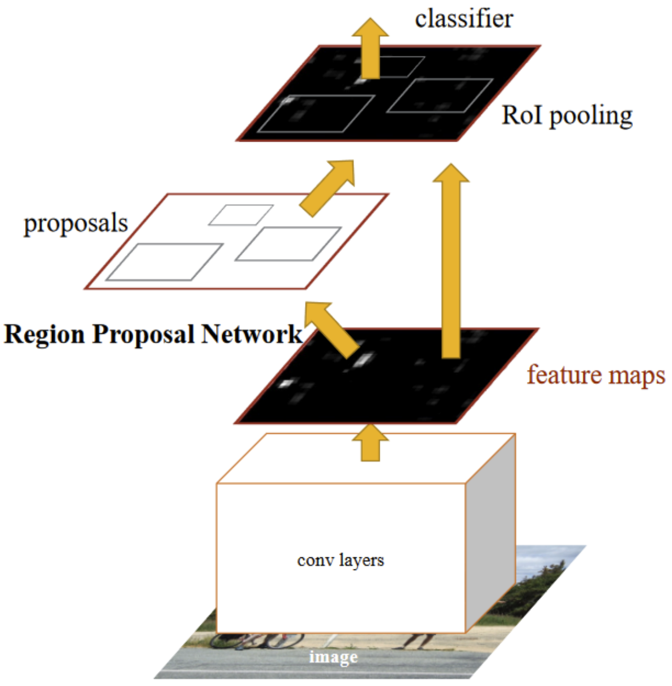


Figure 5.11: Faster R-CNN unified network with its *RPN* and Fast R-CNN classifier head [55]

The Fast R-CNN [56] classification head works by taking the proposed regions from the *RPN* into a *Region Of Interest* (*ROI*) pooling layer. The pooling layer uses max pooling to get features inside the *ROI* converted into a small feature map. The output from the pooling layer gets fed into a sequence of fully connected layers that branch into an object classifier that uses softmax and a bounding box regressor that outputs the location of the bounding box. The architecture for the Fast R-CNN can be seen in figure 5.12, where the classification head used by Faster R-CNN is inside the grey box call *ROI* feature vector.

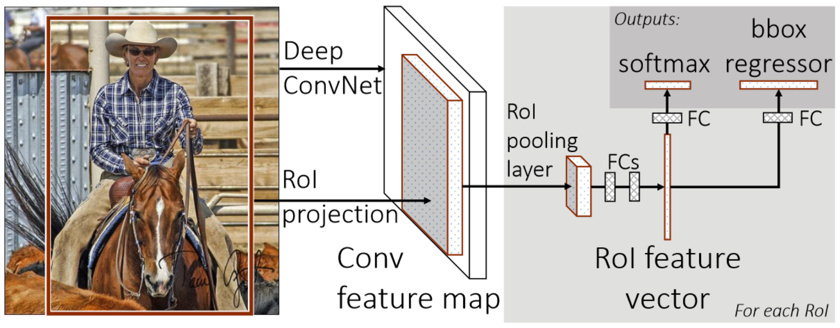


Figure 5.12: Full architecture for the Fast R-CNN network. The grey box on the right shows the part being used for the classifier head for Faster R-CNN [56]

5.5.2 You Only Look Once

YOLOv5 [3] is the other model used in Seasons in Drift: A Long-Term Thermal Imaging Dataset for Studying Concept Drift [6] which was compared against Faster R-CNN. There are no official papers for YOLOv5 [3] so this section will cover the original YOLO [58] architecture and the improvements upon it until YOLOv5.

5.5.2.1 YOLO

YOLO was developed by Redmon et al. [58] and is a one-stage detector, unlike Faster R-CNN [55], a two-stage detector. This is done by framing object detection as a regression problem instead of a classification problem, and as a consequence makes the model faster than Faster R-CNN with a small reduction in performance. This also makes it possible to use YOLO [58] for semantic segmentation. YOLO uses features from the entire image to predict each bounding box and does this also across all classes for an image simultaneously. It divides the input image into a grid with size $S \times S$ seen in figure 5.13, where a cell takes responsibility for an object if the centre falls within it. Each grid cell then predicts B bounding boxes and confidence scores, where the confidence reflects how accurate the bounding box may be and if it contains an object. Each bounding box has 5 predictions: x, y, w, h, and confidence, where (x, y) are coordinates that represent the centre of the bounding box relative to the bounds of the grid cell. w and h represent the width and height relative to the whole image, and confidence prediction represents the IoU between the predicted box and ground truth box. Each grid cell also predicts only 1 class of C conditional class probabilities regardless of the number of boxes B. Where the class is conditioned on the grid cell containing an object.

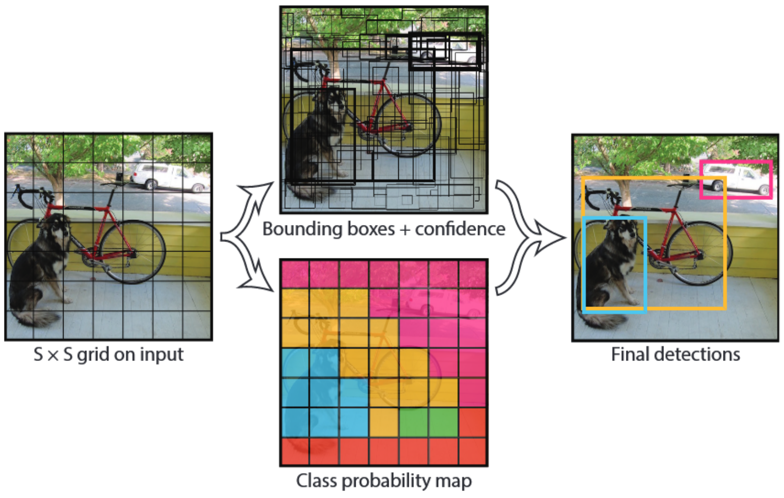


Figure 5.13: The full model for YOLO, where it divides the image into an $S \times S$ grid and for each cell predicts B bounding boxes and C class probabilities that are then used together for the final prediction [58]. These predictions are encoded as an $S \times S \times (B * 5 + C)$ tensor

The network architecture is inspired by GoogLeNet [59], and has 24 convolutional layers followed by 2 fully connected layers. The first layer is a 7×7 convolutional layer followed by a max pool layer, where then this is repeated with 3×3 convolutional layer followed by a max pool layer. The change from GoogLeNet is that instead of using the inception modules YOLO uses 1×1 convolutional layer followed by a 3×3 convolutional layer which can be seen in figure 5.14. The final output of the architecture is a $S \times S \times (B * 5 + C)$ tensor with predictions.

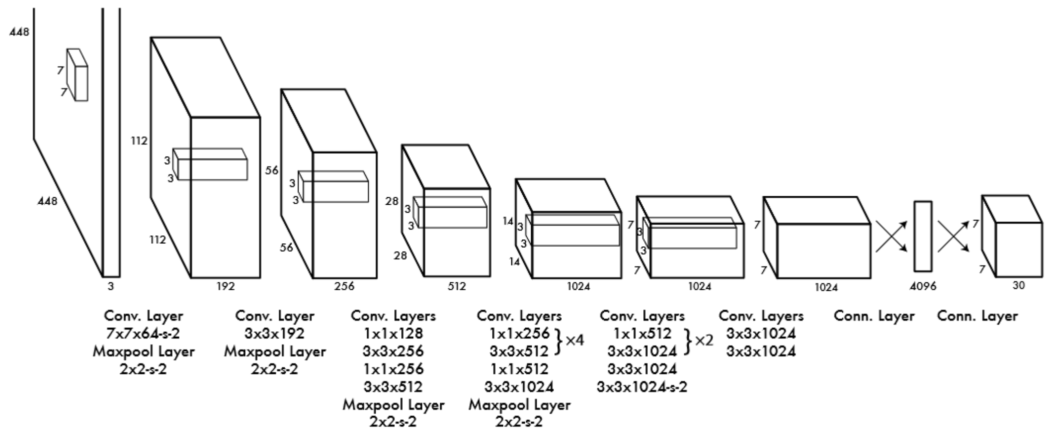
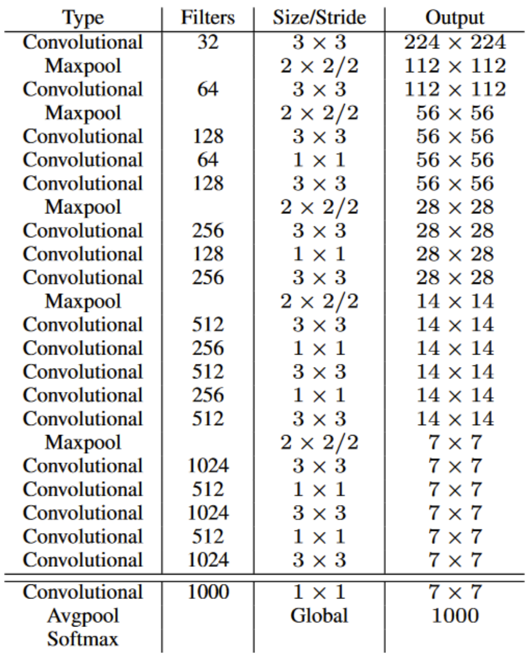


Figure 5.14: The architecture used for YOLO with a total of 24 convolutional layers [58]

5.5.2.2 YOLOv2

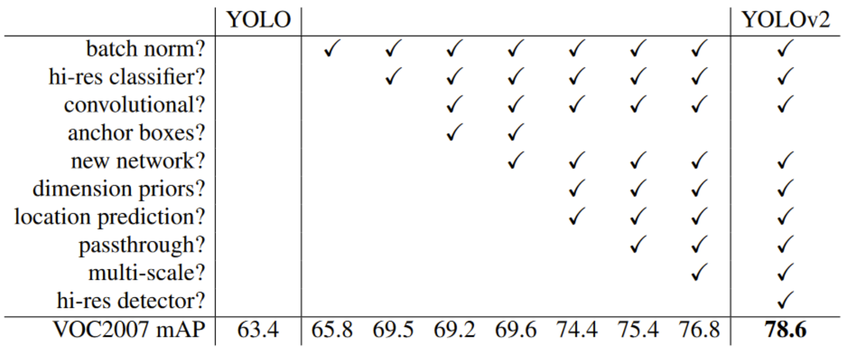
YOLO9000, also known as YOLOv2 [60], made some significant improvements to the YOLO architecture which resulted in increasing the *mean Average Precision* (*mAP*) while also increasing speed. It introduced batch normalization on all convolutional layers, which improved convergence and acted as a regularizer that combatted overfitting. The fully connected layers were removed and instead had a fully convolutional architecture called Darknet-19 which can be seen in figure 5.15. With the removal of fully connected layers, YOLOv2 instead uses the idea of anchor boxes to predict the bounding boxes like in Faster R-CNN [55]. Although this decreases the *mAP* slightly from 69.5 to 69.2 it increases the number of bounding box predictions from 98 to over a thousand and increases recall from 81% to 88%. Instead of the anchor boxes being handpicked YOLOv2 uses dimension clusters, where the priors are picked automatically using k-means clustering on the training set bounding boxes.



Type	Filters	Size/Stride	Output
Convolutional	32	3×3	224×224
Maxpool		$2 \times 2/2$	112×112
Convolutional	64	3×3	112×112
Maxpool		$2 \times 2/2$	56×56
Convolutional	128	3×3	56×56
Convolutional	64	1×1	56×56
Convolutional	128	3×3	56×56
Maxpool		$2 \times 2/2$	28×28
Convolutional	256	3×3	28×28
Convolutional	128	1×1	28×28
Convolutional	256	3×3	28×28
Maxpool		$2 \times 2/2$	14×14
Convolutional	512	3×3	14×14
Convolutional	256	1×1	14×14
Convolutional	512	3×3	14×14
Convolutional	256	1×1	14×14
Convolutional	512	3×3	14×14
Maxpool		$2 \times 2/2$	7×7
Convolutional	1024	3×3	7×7
Convolutional	512	1×1	7×7
Convolutional	1024	3×3	7×7
Convolutional	512	1×1	7×7
Convolutional	1024	3×3	7×7
Convolutional	1000	1×1	7×7
Avgpool		Global	1000
Softmax			

Figure 5.15: The improved backbone used in YOLOv2 Darknet-19, where the last layers with fully connected layers were removed for a full convolutional network [60]

YOLOv2, like YOLO, also predicts the location of the bounding box directly, instead of predicting offsets like other anchor box solutions it predicts the coordinates relative to the location of the grid cell. All the improvement for YOLOv2 is summarized in figure 5.16

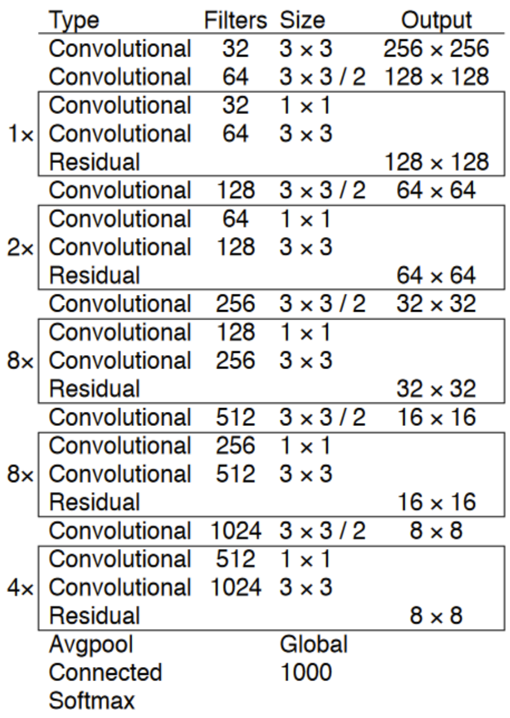


	YOLO								YOLOv2
batch norm?		✓	✓	✓	✓	✓	✓	✓	✓
hi-res classifier?			✓	✓	✓	✓	✓	✓	✓
convolutional?				✓	✓	✓	✓	✓	✓
anchor boxes?				✓	✓				
new network?					✓	✓	✓	✓	✓
dimension priors?						✓	✓	✓	✓
location prediction?						✓	✓	✓	✓
passthrough?							✓	✓	✓
multi-scale?								✓	✓
hi-res detector?									✓
VOC2007 mAP	63.4	65.8	69.5	69.2	69.6	74.4	75.4	76.8	**78.6**

Figure 5.16: The different improvements made in YOLOv2 and the impact they had on increasing mAP [60]

5.5.2.3 YOLOv3

YOLOv3 [61] made incremental improvement upon YOLOv2 [60], increasing the network size and sacrificing speed, but still being faster than other solutions. It was improved by predicting an objectness score, which only assigns one bounding box for each ground truth object using logistic regression. This is done by the output being 1 if it overlaps the ground truth object by more than any other bounding box prior, and for everything else, it is 0. Redmon and Farhadi also changed loss supervision so that if a bounding box prior is not assigned to a ground truth object then it does not incur loss for coordinate or class predictions, but only for objectness. The class predictions were also changed from softmax to using binary cross-entropy, which made the model multilabel so that overlapping labels like woman and person can be predicted in the same bounding box. Using the idea from Feature Pyramid Networks [62], YOLOv3 [61] also predicts boxes at three different scales, and changed the k-means clustering to use three prior boxes on all three scales. The backbone of the network was also changed from darknet-19 to darknet-53 which instead of having 19 convolutions has 53, but also now included residual connections as seen in figure 5.17.



	Type	Filters	Size	Output
	Convolutional	32	3 × 3	256 × 256
	Convolutional	64	3 × 3 / 2	128 × 128
	Convolutional	32	1 × 1	
1×	Convolutional	64	3 × 3	
	Residual			128 × 128
	Convolutional	128	3 × 3 / 2	64 × 64
	Convolutional	64	1 × 1	
2×	Convolutional	128	3 × 3	
	Residual			64 × 64
	Convolutional	256	3 × 3 / 2	32 × 32
	Convolutional	128	1 × 1	
8×	Convolutional	256	3 × 3	
	Residual			32 × 32
	Convolutional	512	3 × 3 / 2	16 × 16
	Convolutional	256	1 × 1	
8×	Convolutional	512	3 × 3	
	Residual			16 × 16
	Convolutional	1024	3 × 3 / 2	8 × 8
	Convolutional	512	1 × 1	
4×	Convolutional	1024	3 × 3	
	Residual			8 × 8
	Avgpool		Global	
	Connected		1000	
	Softmax			

Figure 5.17: The illustration shows the improved backbone design of Darknet-53, which increased the convolutional layers from 19 to 53 and integrated residual layers [61]

5.5.2.4 YOLOv4

YOLOv4 was developed by Bochkovskiy et al.[63]. There were substantial improvements over YOLOv3 [61], two of which were integrating what was called "bag of specials" and "bag of freebies". Bag of specials are plugin modules and post-processing methods which increase the inference cost by a small amount but significantly improve the accuracy of object detection, and bag of freebies are offline training strategies such as data augmentation. The methods from bag of freebies and specials were integrated into the backbone and detector head. The model introduced two new bag of freebies, this being mosaic data augmentation and *Self-Adversarial Training* (*SAT*). Mosaic data augmentation mixes four training images, which allows the detection of objects outside their normal context by "stitching" a part of the four images into one image. *SAT* is also a data augmentation technique which operates in two stages. In the first stage, the neural network makes an adversarial attack on the input image to create a deception that there are no objects in the image. In the second stage, the network is trained to detect an object on that image in the normal way. Using *SAT* makes the model more robust. An additional improvement was also made by using genetic algorithms to select the optimal hyperparameters. The models' architecture can be seen in figure 5.18

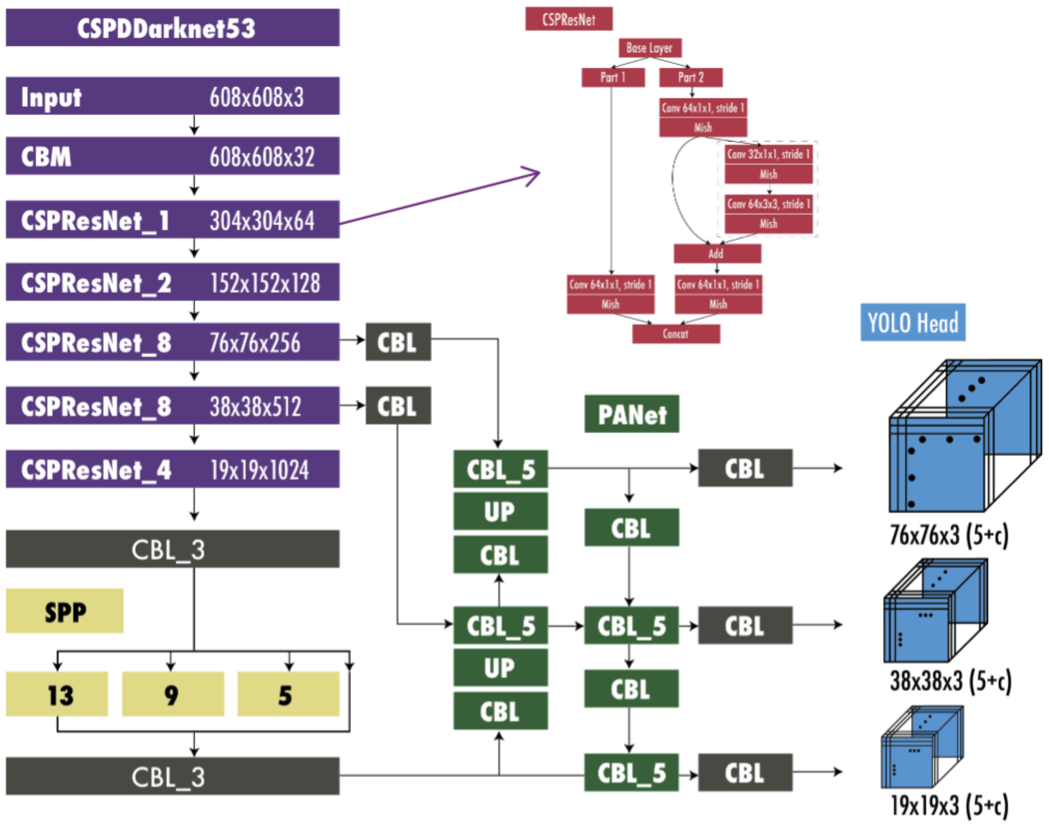


Figure 5.18: The full YOLOv4 architecture. It shows the different modules used as bag of freebies and bag of specials, these include CSP: cross-stage partial connection, CMB: Convolution + Batch Normalization + Mish activation, CBL: Convolution + Batch Normalization + Leaky ReLU, UP: upsampling, SPP: Spatial Pyramid Pooling, and PANet: Path Aggregation Network [64]

5.5.2.5 YOLOv5

Lastly, YOLOv5 [3] was released by Ultralytics a couple of months after YOLOv4 [63]. There wasn't much of a difference in the architecture, but instead, it was developed using Pytorch instead of Darknet. YOLOv5 is also open source, actively maintained, and easy to use, train, and deploy. It has 5 models with varying sizes so that it can be deployed in mobile to cloud devices. It also incorporates an algorithm called AutoAnchor, which is a pre-training tool that checks and adjusts anchor boxes to see if they fit the dataset and training settings. The AutoAnchor applies a k-means function to the dataset labels and generates initial conditions for a genetic evolution algorithm which evolves over generations to get the best-fit [64]. The architecture for YOLOv5L can be seen in figure 5.19

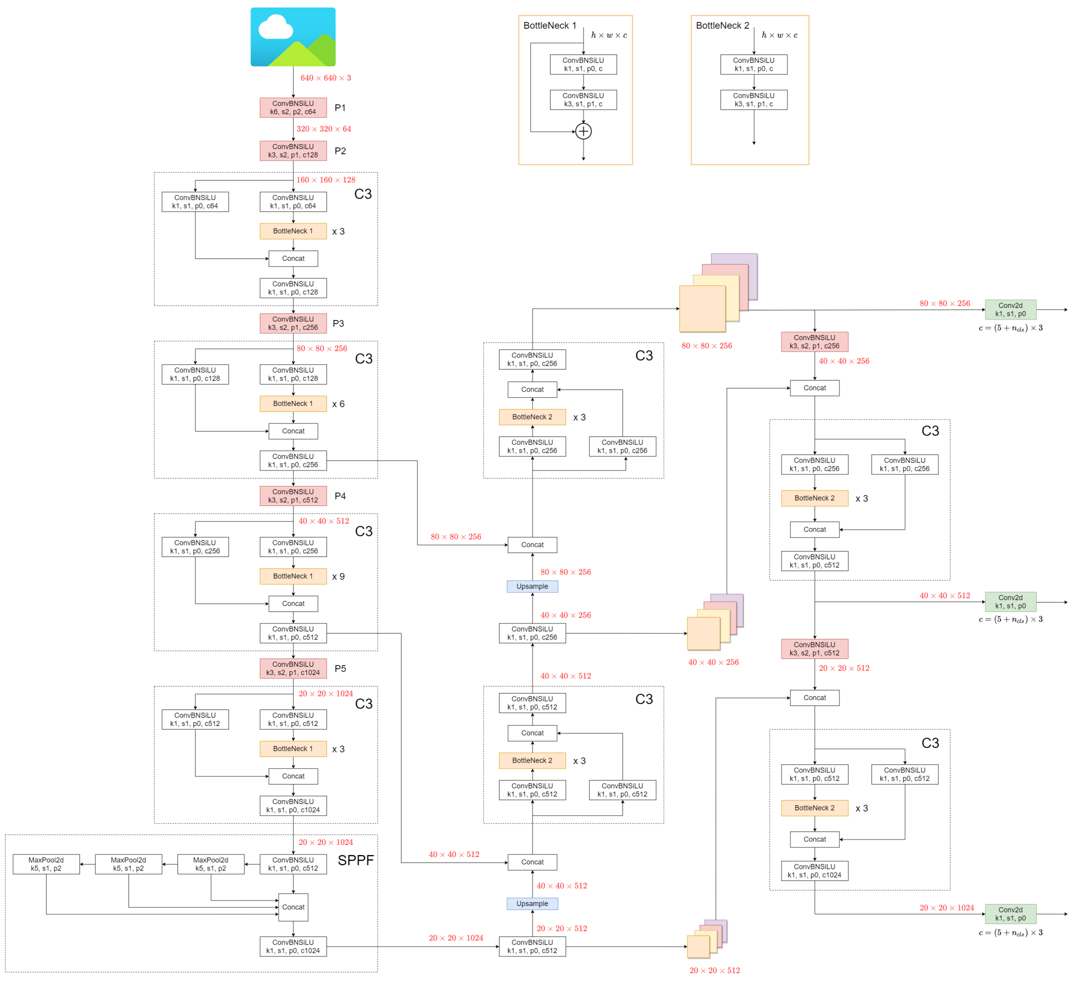


Figure 5.19: Shows the structure of the YOLOv5L. The structure of YOLOv5 consists of three main parts. This is the backbone CSP-Darknet53 like in YOLOv4 [63], the neck that connects the backbone and head with Spatial Pyramid Pooling Fusion (SPPF) and Cross-Stage Partial connection and Path Aggregation Network (CSP-PAN), and the head which is the one used in YOLOv3 [3], [61]

5.5.3 Performance Evaluation

The baseline performance of the models for the *LTD* [6] dataset can be seen in figure 5.3. While there is merit in using a two-stage detector as the conventional thought is that they are more accurate than a one-stage detector. However, since YOLO [58] has been well-maintained and updated it has over time become better than Faster R-CNN. The evaluation metrics for table 5.3 is mAP_{50}, and as can be seen from the table YOLOv5 is more precise than Faster R-CNN on all accounts. Therefore the object detection model used for the downstream task will be YOLOv5.

Method	Train	Test		
		Jan.	Apr.	Aug.
YOLOv5	Feb 100	0.7930	0.4860	0.4830
	Feb. 100 + Mar. 100	**0.8690**	**0.6640**	**0.6110**
Faster R-CNN	Feb 100	0.6400	0.2560	0.3180
	Feb. 100 + Mar. 100	**0.6990**	**0.3910**	**0.3380**

Table 5.3: Results showing mAP_{50} (↑) across every frame in the Seasons in Drift: A Long-Term Thermal Imaging Dataset for Studying Concept Drift paper. The method is the object detection model used, train is data from which month and how many frames, and test results for different months [6].

5.6 Evaluation Metrics

This section will cover the evaluation metrics used to measure object detection performance. Commonly the metric used for this is mAP, but to calculate this, precision, recall, and IoU need to be explained first.

5.6.1 Intersection Over Union

The IoU measures how much a predicted bounding box overlaps the ground truth with a value between 0 and 1. If the prediction is perfect and the bounding box overlaps it completely the value is one and if the prediction doesn't overlap at all the value is 0. The value is calculated by taking the ratio for the area where the bounding boxes intersect and the area of the union for the two bounding boxes. The IoU can be used as a threshold together with the class labels for object detection which determines what can be a *True Positive* (TP), *False Positive* (FP), *False Negative* (FN), or *True Negative* (TN) [65]. A prediction would be a TP when the IoU is above thresholds and it was the correct class label, it would be FN when it is the correct label but the IoU is less than the threshold. A TN would then be not doing a prediction when there is nothing, and a FP would be having an IoU above the threshold with the incorrect label.

5.6.2 Precision And Recall

Recall, also known as sensitivity, is the same as the true positive rate, which measures the proportion of real positives that are predicted as positives as seen in eq. 5.1 [66]

$$Recall = \frac{TP}{TP + FN} \tag{5.1}$$

In itself, recall isn't highly regarded for object detection as the focus is more on how precise the classifier is [66]. To get how precise the detection is precision can be used, where precision measures the proportion of predicted positives that are positives as seen in eq. 5.2

$$Precision = \frac{TP}{TP + FP} \tag{5.2}$$

While precision might be more desirable recall is still an important aspect so *Average Precision* (AP) can be used instead. As the name might suggest AP is not the average of precision but is the area under the precision and recall curve [30], [65].

5.6.3 Mean Average Precision

The precision and recall curve can be made by plotting the precision against recall for different IoU thresholds [65]. The negative relationship between precision and recall, where a higher FN is a lower FP and vice versa, makes it so that the curve ideally has a 90° angle where precision and recall is 1. The average precision can be calculated by the eq. 5.3

$$AP = \int_0^1 P(R)dR \approx \frac{1}{n}\sum_{0.0}^{1} P(R) \tag{5.3}$$

where $P(R)$ is the measured precision and recall at a certain IoU threshold and n is how many increment steps there are from 0.0 to 1 which determine the IoU threshold.

The mAP can then be calculated for each class by taking the average of AP across all relevant classes as seen in eq. 5.4

$$mAP = \frac{1}{k}\sum_{i}^{k} AP_i \tag{5.4}$$

where k is the number of classes that are in consideration and i is the index. To calculate mAP_{50} would mean to get the AP where the increment steps start at 0.5 IoU and using eq. 5.4 if there is multiple classes. More recent papers use mAP in the COCO context, this is defined as $mAP_{0.50:0.95}$ which means to take the average of all mAP starting from 0.5 to 0.95 with increments of 0.05 [67].

Chapter 6

System Design

6.1 Design

The design is inspired by the approach of Zoetgnande et al. [36]. Their experiments with combining various representations of the data are the inspiration point for this work's design. They successfully created a generalized representation between RGB and *TIR*. Based on this, it may be feasible to utilize this technique to generalize across seasonal drift. To be able to test the efficacy of many representations, the design is made with the intention for the representations to be easily swapped. Beyond the representations, the design should be static, for the purpose of consistency to give fair testing conditions. The design was decided to be a YOLOv5-based model [3], where the input consists of three gray scale representations, merged into a 3-channel image, imitating RGB. This approach was chosen based on the great performance of YOLOv5 and advice from its developers, discouraging any attempts to alter the number of channels in YOLOv5 from the default 3.

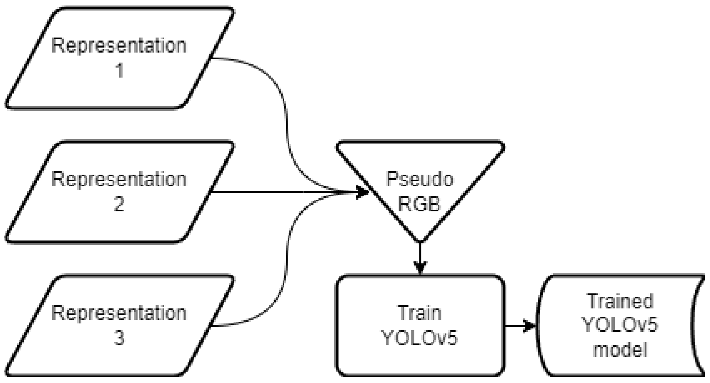


Figure 6.1: An illustration of the fundamental pipeline. Takes 3 representations and merges them to a pseudo-RGB image for training YOLOv5.

The fundamental pipeline shows the simplest form of the design. It simplifies the representations to a data input. These representations can take many forms, from a blurred image to the output of an AI model. The only requirement for a representation is for it to contain features relevant to the original image. This is achieved by building the representations from it, and evaluating their contribution later. From the extensive list of options, this project will focus on U^{2}Net for its superior performance mentioned in section 5.2.4 and Gunnar Farnebäck Optical Flow for its simplicity. As such, an extended pipeline can be seen in figure 6.2. To maintain the original information, it was decided that a minimum of one representation should always be the source data, to maintain the original image data and contextual information.

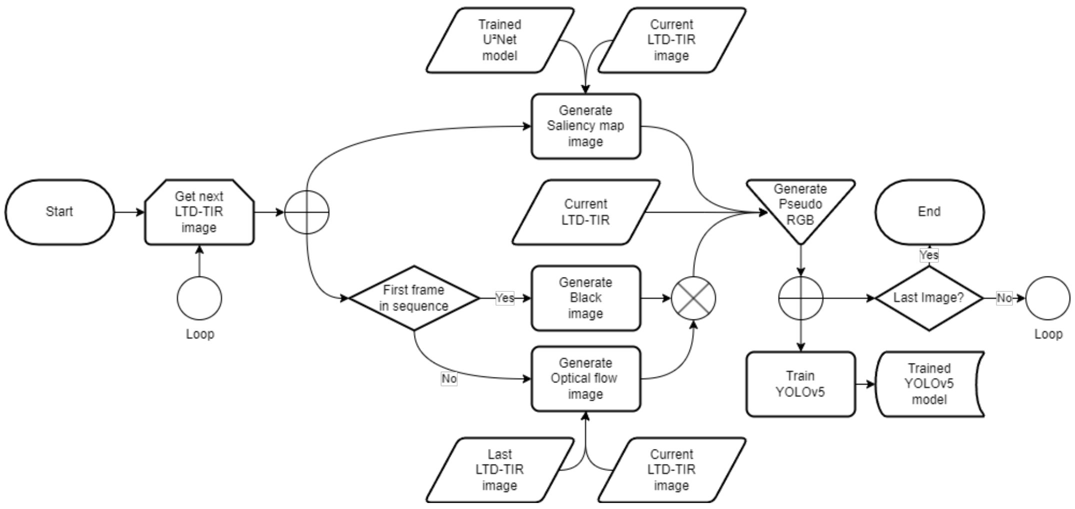


Figure 6.2: An extended pipeline example, utilizing U^{2}Net, Gunnar Farnebäck optical flow and the LTD-TIR dataset to provide 3 representations for YOLOv5.

The extended pipeline shows what a pipeline could look like. For the actual implementation, relevant representations will follow a pipeline that can be inserted for the implementation, as an input for one of the representations. In figure 6.3 the main pipeline can be seen in its final form.

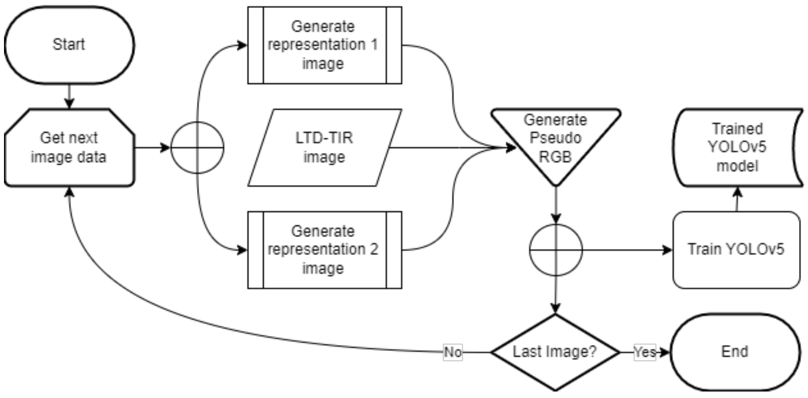


Figure 6.3: Complete flowchart of the modular design. The flowchart loops repeatedly over all images, feeding them to the YOLOv5 training.

Chapter 7

Implementation

7.1 Optical Flow Tests

A preliminary experiment was done to evaluate which optical flow model would be most beneficial to achieve the best performance. To evaluate this, a selection of models were selected and evaluated, based on visual inspection. The tested models were Lucas Kanade [51], [52], Gunnar Farnebäck [2], [51] and RAFT [54]. Lucas Kanade was selected for its ease of use, and simplicity. Gunnar Farnebäck for its expected great performance as an entirely algorithmic model, and RAFT which is considered to be one of the better CNN-based optical flow implementations.

When observing the output of Lucas Kanade, we see little to no useful information. It is simply not able to find relevant features when tested on some of our data.

As can be seen, the detected flow looks like noise and is not expected to be useful for the project.

When testing on Gunnar Farnebäck, the flow seems to follow objects in motion correctly. The result is rather good, despite a substantial halo being present.

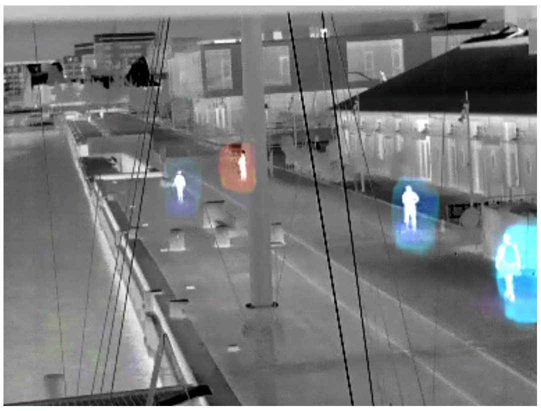


Figure 7.1: An example of a Gunnar Farnebäck model output, generated from our data

It can be seen that all humans in the sample image are detected. While this isn't consistent between all tested images, it is more likely to see them than not.

The RAFT implementation was run on a pre-trained model, as it is supervised, and would require the ground truth to train. Despite this, good performance was expected. The outcome was that RAFT either performs astoundingly well or returns what can only be described as a noise texture.

Figure 7.2: An example of a Raft model output, generated from our data

As can be seen, in one image, the flow effectively marks the pedestrian perfectly, for those detected. In images 2 and 3 however, we can see raft locating everything and nothing respectively.

Considering the inconsistency of RAFT in this case, Gunnar Farnebäck was chosen as the preferred option.

7.2 Data Split

The extended *LTD* data is already split in a train, test, and validation set. For this work, these were filtered into their respective months complying with set splits. This will make it easier to train and test the data more granularly for the U^{2}Net [1] and YOLOv5 [3]. As this work will focus on seasonal drift all the data for September is used as a test as there is a small number of samples for September and no samples for October and November. The number of images for the data split can be seen in table 7.1.

	Extended LTD-TIR		
	Train	Test	Val
Total	749195	49430	46013
Jan	84905	5798	4699
Feb	98414	6479	5658
Mar	124902	8966	7487
Apr	125306	7949	8561
May	75952	4913	4720
Jun	106081	7036	6707
Jul	33708	2163	2180
Aug	99388	6006	6001
Sep	0	659	0
Oct	0	0	0
Nov	0	0	0
Dec	0	0	0

Table 7.1: The total number of images split in train, test, and validation set and their respective months

7.3 Hardware And Software

All training and inference were done on **AI Cloud** where the hardware and software used can be seen below:

Hardware:

- RAM: 64GB
- GPU: NVIDIA L40S
- Rest: Depends on AI Cloud allocation. See **AI Cloud documentation**

Software:

- Ubuntu 20.04.6 LTS
- **Containr** build from an Anaconda 3 .yaml file:
  - Python 3.11.0
  - Anaconda 3 24.1.2
    - Pytorch 2.2.2
    - OpenCV 4.9.0.80
    - Numpy 1.24.3

∗ See .yaml file in appendix A.1

7.4 Data Preprocessing And Training

This section describes the data preprocessing, using the designed pipeline, needed to make the pseudo-RGB image and how it is obtained for the training process of U^{2}Net [1] and YOLOv5 [3] models.

7.4.1 Training U^{2}Net

Since U^{2}Net [1] is a model that needs masks for an image and the *LTD* dataset has no masks that can be used for ground truth. This work instead will procure the masks using the bounding boxes from the annotated classes including pedestrians. Only the bounding boxes for pedestrians were used to make the masks and the masks were also filled as can be seen in figure 7.3 with the original image from May can be seen in figure 7.4

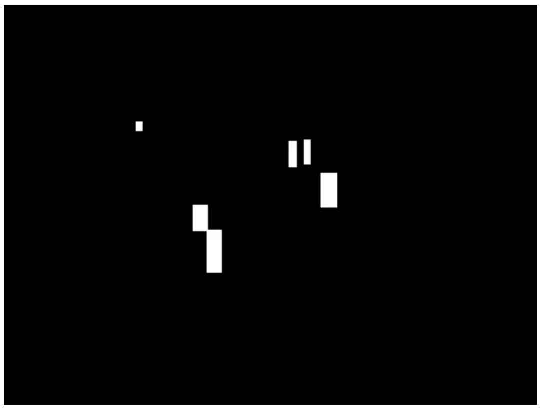


Figure 7.3: The procured pedestrian masks from the bounding boxes which are used as ground truth for training U^{2}Net.

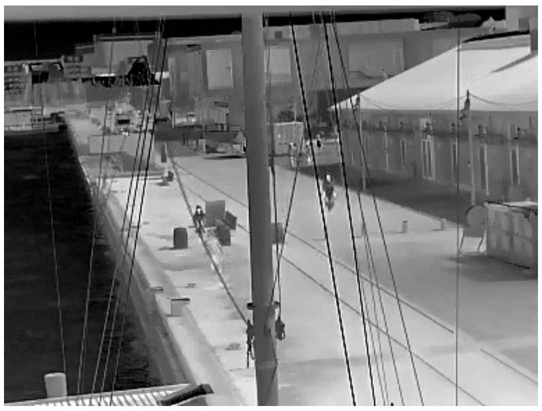


Figure 7.4: The original image from the extended LTD dataset. The image is from May.

Three versions of a U^{2}Net model were trained, referred to as the: winter, spring, and summer models. Each model is retrained following the original implementation on the data specific to its respective season for 10 epochs with a learning rate of 1e-4 and a batch size of 64. The final output after 10 epochs for the spring model can be seen in figure 7.5

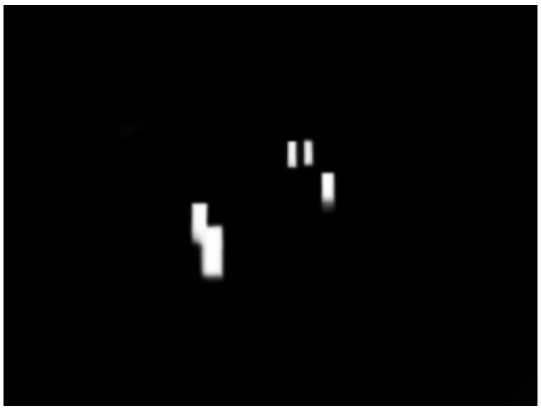


Figure 7.5: The final output saliency map for the spring model

As shown in figure 7.5 the saliency map obtained from the trained model resembles bounding boxes. The difference between the outputs from the winter, spring, and summer models and what they highlight in the original image can be seen in figure 7.6. The expected output is that the models highlight the pedestrians that were annotated in the masks seen in figure 7.3. The output instead shows that all three models missed the pedestrian that was occluded at the top left of the image. It can also be seen that the winter model only highlights two pedestrians from the ground truth and the spring and summer models highlight all pedestrians in the centre of the image. However, the summer model has some *FP*. The underperformance of the winter model can be caused by its overfitting for winter samples.

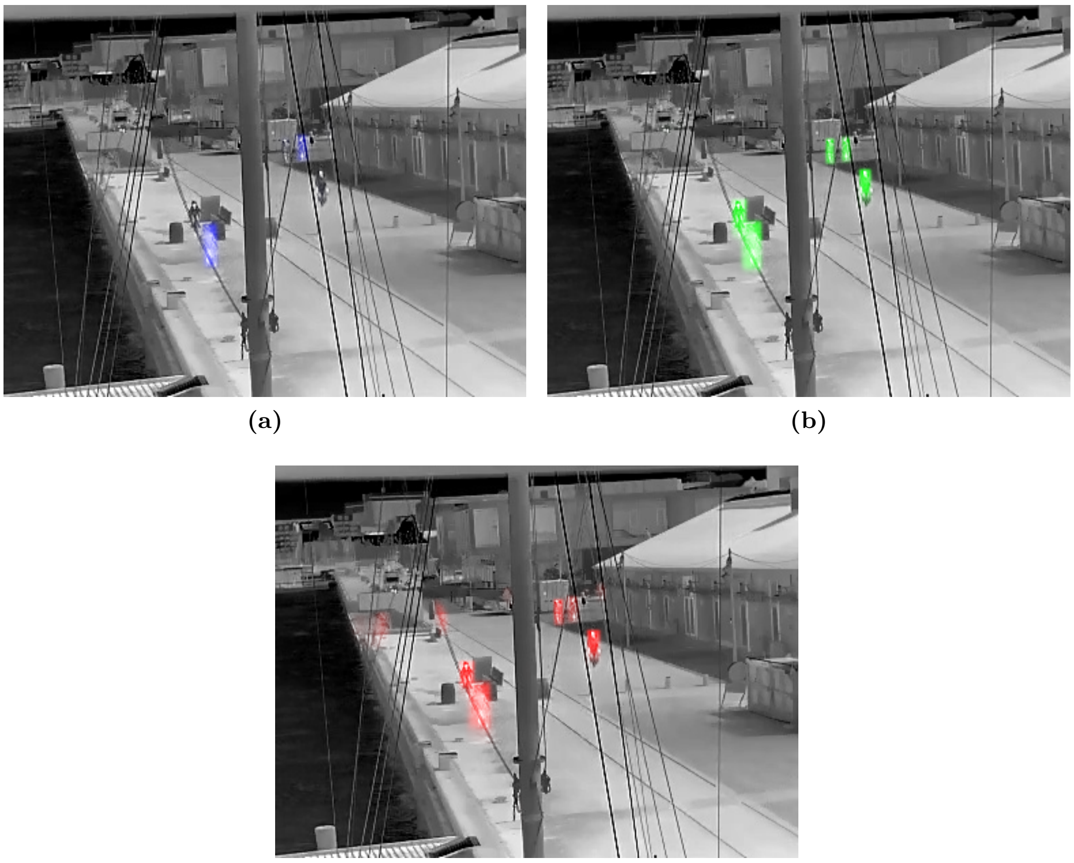

(a) (b) (c)

Figure 7.6: The saliency maps obtained from the U^{2}Net models superposed on top of the original image. (a) The output from the winter model. (b) The output from the spring model. (c) The output from the summer model

7.4.2 Training YOLOv5

As described in section 6.1 the model presented in this work makes a pseudo RGB image. This is done by preprocessing the data using the pipeline shown in figure 6.2, where the blue channel is the output from U^{2}Net used as a mask for the original image, the green channel is the source, and the red channel is the output from optical flow used as a mask. An example of an image using the summer U^{2}Net model can be seen in figure 7.7

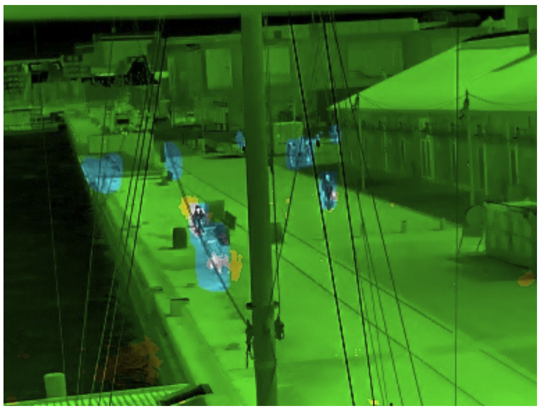


Figure 7.7: The preprocessed image made using the pipeline in this work for the summer U^{2}Net model.

This preprocessing is done for all 3 U^{2}Net models so that there is a total of 4 datasets with different variations, these being the original, preprocessed winter U^{2}Net, preprocessed spring U^{2}Net, and preprocessed summer U^{2}Net. The extended LTD dataset also has a different annotation format whereas YOLO uses the COCO annotation format. The change in annotation can be seen here:

- id, class, top$_x$, top$_y$, bottom$_x$, bottom$_y$, is crowded?
  $\downarrow$
- class, normalized centre x, normalized centre y, normalized width, normalized height

Three baseline models were also trained each having the data for their respective seasons winter, spring, and summer which will be used to compare against the models using the preprocessed data.

To train the YOLOv5 model the train file was used from the Ultralytics YOLOv5 repository [3]. The pre-trained weights from yolov5x were used with the Adam [68] optimizer and a batch size of 187. The models were all trained for 50 epochs with the hyperparameters used being based on the 'hyp.scratch-high.yaml' file. The changes made for the hyperparameters are:

- lr0: 1e-4
- Box loss gain: 1e-2
- Objectness loss gain: 0.4
- Objectness binary cross entropy loss positive weight: 0.6
- Number of anchors: 9

- perspective: 1e-3
- mixup: 0.2

The hyperparameters were obtained using individual sweeps and the full .yaml file can be seen in appendix A.2. The image size was set to 384 with the "–img 384" argument and the "–rect" argument was also used. An example of the full terminal argument can be seen in listing 1

```
python train.py --rect --img 384 --epochs 50 --data Harborfront_summer.yaml
↪  --batch-size 187 --name Summer_baseline --optimizer Adam --weights yolov5x.pt
↪  --hyp data/hyps/hyp.mine-scratch_v3.yaml --device 0
```

Listing 1: Linux terminal argument to run train.py

An example of images with annotations is saved by default by YOLOv5. An example of this can be seen in figure 7.8

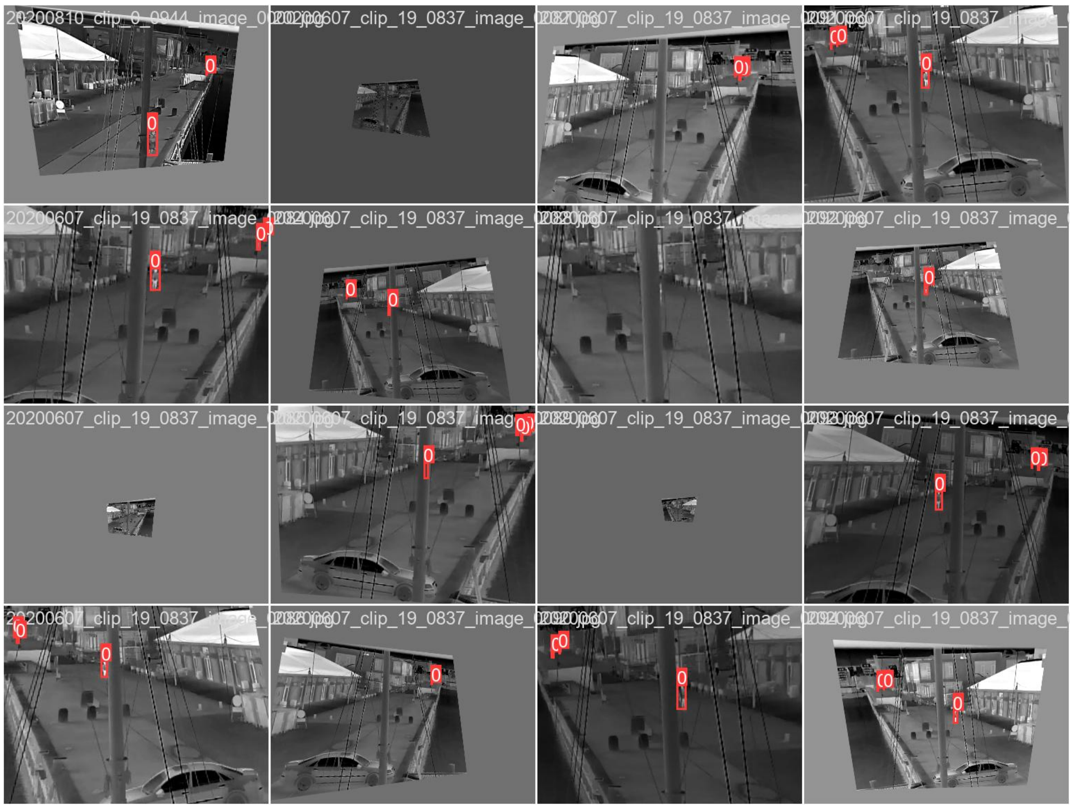


Figure 7.8: Train batch saved by YOLOv5 with annotations

YOLOv5 also saves different plots on the Weights and Biases website if the library is available, this was used to automatically plot the losses and metrics to a workspace which can be seen **here**. The naming scheme used for the models is "Season_model type". The preprocessed models have been called U^{2}OF, which stands for U^{2}Net Optical Flow.

The validation box loss gain can be seen in figure 7.9. With a closer inspection, it can be seen that the models made for this work start with a lower loss, but the baseline catches up and has a lower loss at the end.

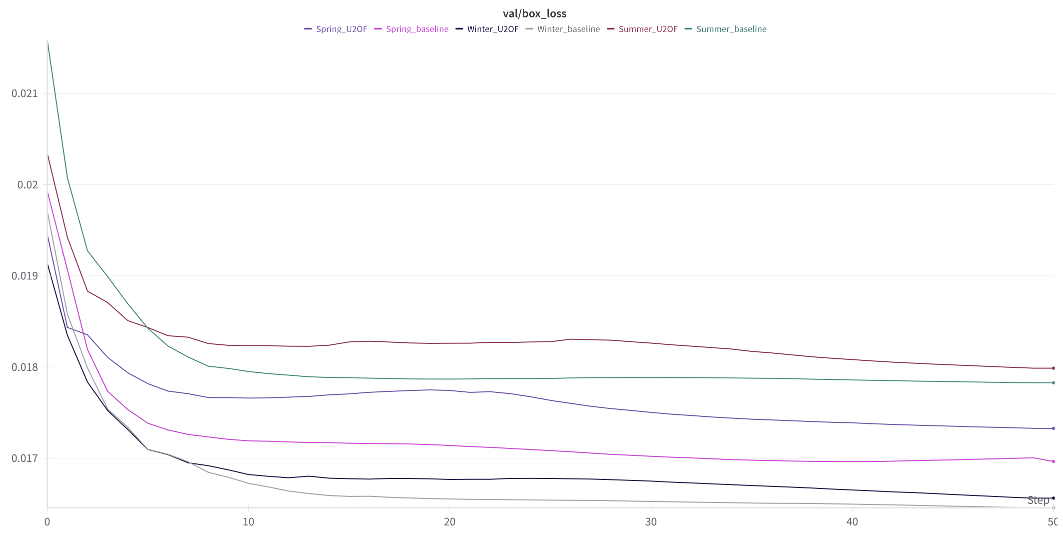


Figure 7.9: The validation box loss plot for the trained baseline and preprocessed models

The validation objectness loss can be seen in figure 7.10, which seems to not change much.

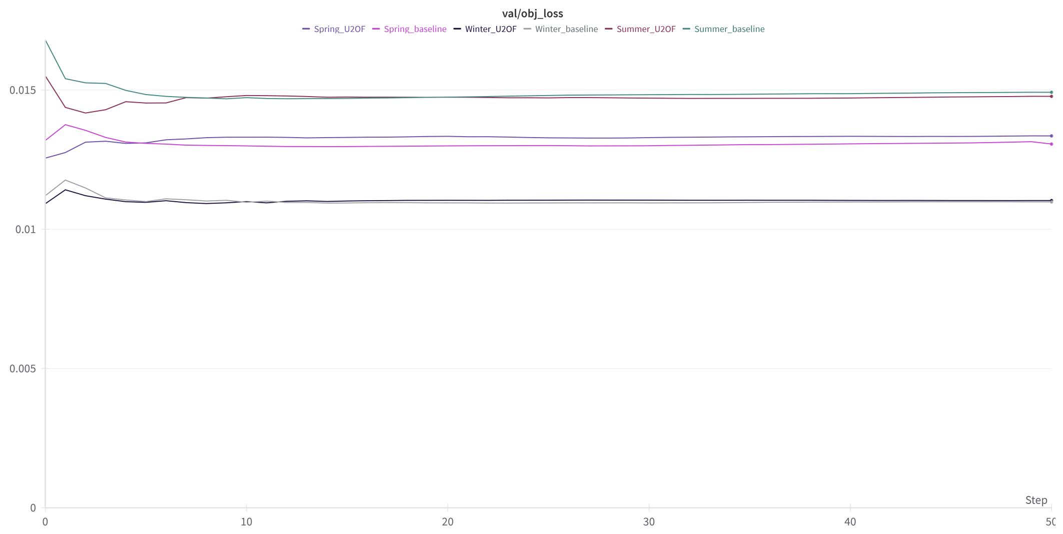


Figure 7.10: The objectness loss plot for the trained baseline and preprocessed models

The precision plot of the models can be seen in figure 7.11, where the models with the preprocessed data seem to be lower. However, this change seems to be because of the tradeoff being from an increase in recall seen in the recall plot in figure 7.12.

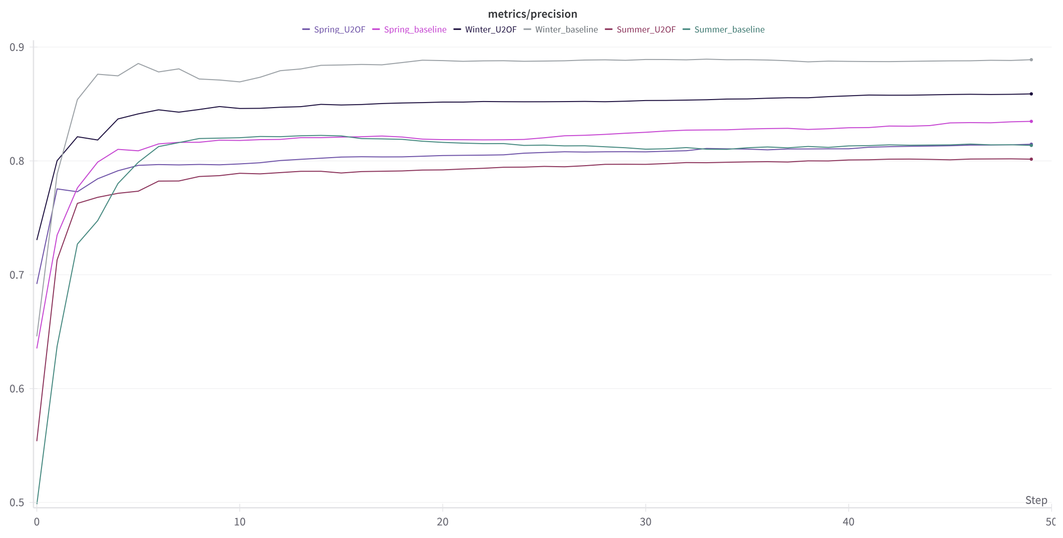


Figure 7.11: The precision plot for the trained baseline and preprocessed models

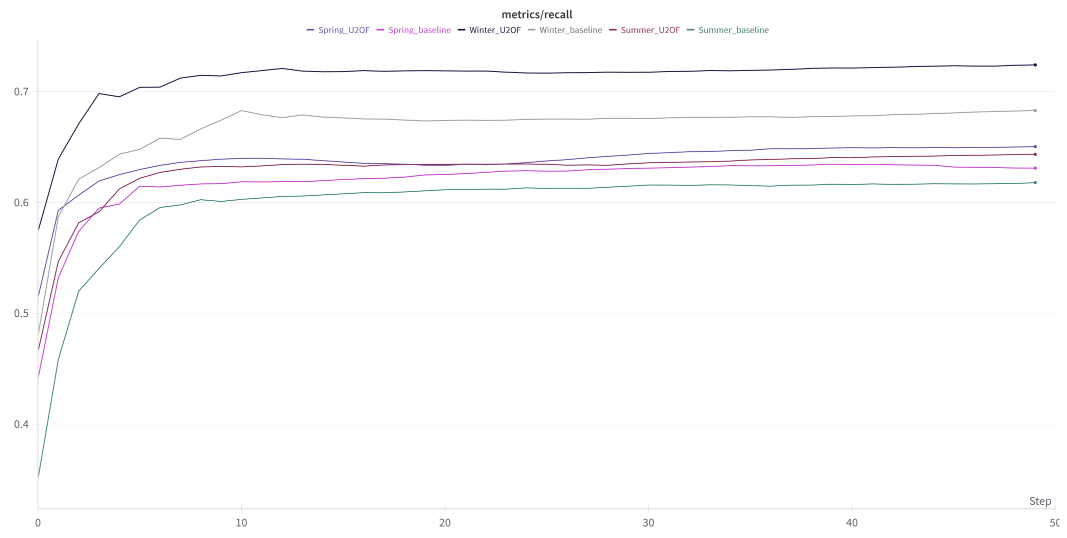


Figure 7.12: The recall plot for the trained baseline and preprocessed models

After 50 epochs the precision and recall values for each model's own validation set can be seen in table 7.2

	Winter baseline	Winter U^{2}OF	Spring baseline	Spring U^{2}OF	Summer baseline	Summer U^{2}OF
Precision	0.889	0.859	0.835	0.814	0.815	0.802
Recall	0.683	0.724	0.631	0.651	0.617	0.644

Table 7.2: Precision and recall values for each models own validation set after 50 epochs

Table 7.2 shows that all the preprocessed models seem to have around a 3% increase in recall with the tradeoff being a 3% decrease in precision for all except the summer U^{2}OF, which is on par with the baseline.

Chapter 8

Expirement procedure

This chapter describes the experiments and the procedure for how the experiments were conducted using the trained models mentioned in section 7.4.2.

Six main experiments were conducted to find the performance between the different seasons. Three of these were experiments on the baseline models for each season, and the other three were experiments using the proposed U^{2}OF model for each season. Each model was tested on all months for a more granular measurement, which was done using the original implementation of "val.py" in the Ultralytics YOLOv5 [3] repository. The argument "–task test" was used to use the test set for each month, and all images were set to a size of 384 using the "–img 384" argument. A batch size of 187 was also used to make it consistent with how the model was trained, but this shouldn't change the results for inference. An example of the full terminal argument for the U^{2}OF spring can be seen in listing 2

```
val.py --task test --img 384 --weights runs/train/Spring_U2OF/weights/last.pt
↪  --batch-size 187 --device 0 --project runs/test/spring_U2OF --data
↪  Harborfront_Januar_spring_U2OF.yaml --name Januar
```

Listing 2: Linux terminal argument to run val.py

To change the models the "–weights" argument can be changed to the path of the desired model checkpoint. The project folder it is saved under can be changed with the "–project" argument and the name of the test can be changed with the "–name" argument (This work used "Januar" which is Danish for January).

The expectation for these experiments is that U^{2}OF show an improved performance in the seasons where the model isn't trained against the baseline. This means that ie. the U^{2}OF winter model performs better than the baseline between March and September.

Chapter 9

Results

9.1 Evaluation Of Experiments

9.1.1 Baselines

The baselines serve as proof of seasonal concept drift, as well as a comparison for subsequent tests.

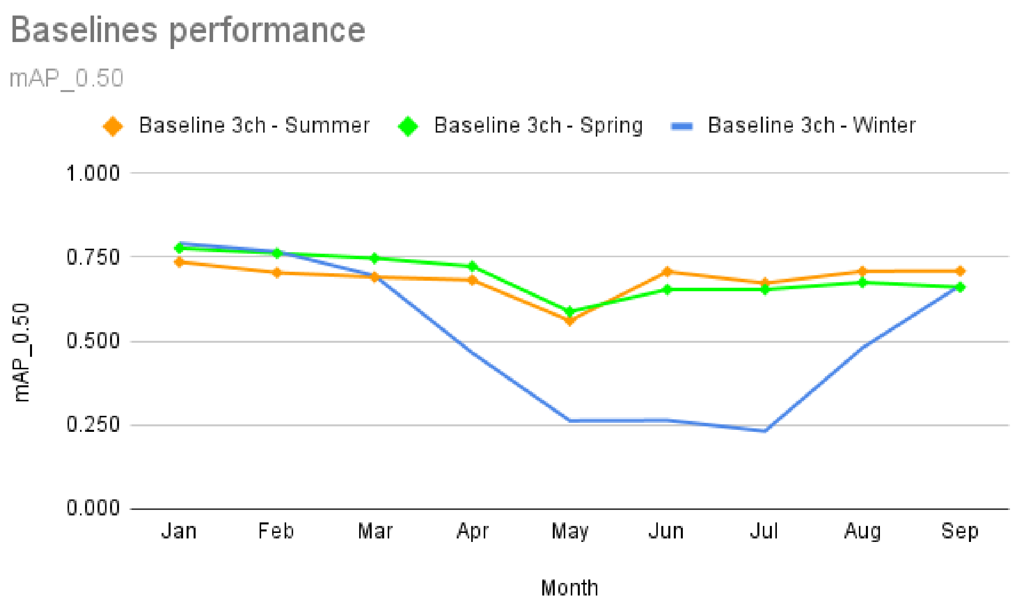


Figure 9.1: mAP$_{0.50}$ distribution of baselines as a function of time of year.

	mAP$_{0.50}$								
	Jan	Feb	Mar	Apr	May	Jun	Jul	Aug	Sep
Baseline 3ch - Summer	0.736	0.704	0.691	0.682	0.561	0.707	0.673	0.708	0.709
Baseline 3ch - Spring	0.777	0.762	0.747	0.723	0.588	0.654	0.654	0.675	0.661
Baseline 3ch - Winter	0.791	0.767	0.695	0.465	0.263	0.264	0.232	0.480	0.666

Table 9.1: mAP$_{0.50}$ results for baseline models on the test set

The table 9.1 shows the performance of the three baselines across the dataset. The difference in performance can also be seen in figure A.1. As expected, the winter baseline outperforms the others during the winter months, the spring baseline during the spring months, and the summer baseline during the summer months. The winter baseline significantly underperforms during the summer months, showing the clear presence of concept drift. Overall, the spring baseline is the best performing, though the summer baseline is close in comparison. It can be observed that a major performance dip happens in May, which is a recurring trend and will be addressed later.

	$mAP_{0.50:0.95}$								
	Jan	Feb	Mar	Apr	May	Jun	Jul	Aug	Sep
Baseline 3ch - Summer	0.399	0.344	0.330	0.318	0.238	0.323	0.303	0.343	0.309
Baseline 3ch - Spring	0.441	0.387	0.371	0.347	0.254	0.304	0.290	0.322	0.306
Baseline 3ch - Winter	0.462	0.403	0.351	0.228	0.123	0.126	0.102	0.244	0.286

Table 9.2: COCO mAP results for the baseline models

The table 9.2 contains the COCO mAP for the baselines. In combination with Table 9.1, it can be seen that the winter baseline performs better than the other baselines relative to their performance in the COCO mAP. This suggests that the IoU scores used to threshold what is considered a detection are higher in January for the winter baseline compared to the other baselines.

9.1.2 U^{2}OF Test Results

Our implementation is compared to the relevant baseline, for the different tested models.

9.1.2.1 U^{2}OF Summer Performance

	$mAP_{0.50}$								
	Jan	Feb	Mar	Apr	May	Jun	Jul	Aug	Sep
Baseline 3ch - Summer	0.736	0.704	0.691	0.682	0.561	0.707	0.673	0.708	0.709
U^{2}OF - Summer	0.728	0.698	0.682	0.641	0.537	0.691	0.676	0.702	0.700

Table 9.3: $mAP_{0.50}$ results for U^{2}OF trained on summer data compared to the baseline model

Table 9.3 shows that the U^{2}OF underperforms with as much as 0.041, in April, and to a lesser degree for the rest of the months. It outperforms by 0.003 in July but is otherwise unremarkable.

	$mAP_{0.50:0.95}$								
	Jan	Feb	Mar	Apr	May	Jun	Jul	Aug	Sep
Baseline 3ch - Summer	0.399	0.344	0.330	0.318	0.238	0.323	0.303	0.343	0.309
U^{2}OF - Summer	0.410	0.353	0.334	0.307	0.225	0.323	0.296	0.341	0.298

Table 9.4: COCO mAP results for U^{2}OF trained on summer data compared to the baseline model

The table 9.4 shows that the U^{2}OF summer measurements have better COCO mAP scores. This suggests that the predictions made in these months have a better IoU score.

9.1.2.2 U^{2}OF Spring Performance

	mAP$_{0.50}$								
	Jan	Feb	Mar	Apr	May	Jun	Jul	Aug	Sep
Baseline 3ch - Spring	0.777	0.762	0.747	0.723	0.588	0.654	0.654	0.675	0.661
U^{2}OF - Spring	0.757	0.752	0.753	0.716	0.593	0.641	0.632	0.651	0.492

Table 9.5: mAP$_{0.50}$ results for U^{2}OF trained on spring data compared to the baseline model

The U^{2}OF spring measurements found in table 9.5 show similar results to the summer variant. In this instance, the model outperforms the baseline during March and May, by 0.006 and 0.005 respectively. It is also seen that the model substantially underperforms during September.

	mAP$_{0.50:0.95}$								
	Jan	Feb	Mar	Apr	May	Jun	Jul	Aug	Sep
Baseline 3ch - Spring	0.441	0.387	0.371	0.347	0.254	0.304	0.290	0.322	0.306
U^{2}OF - Spring	0.444	0.394	0.387	0.360	0.258	0.305	0.282	0.321	0.230

Table 9.6: COCO mAP results for U^{2}OF trained on summer data compared to the baseline model

Observing table 9.6 it can be seen that the U^{2}OF spring model outperforms the baseline from January to June, suggesting that it has better IoU scores than the baseline for these months.

9.1.2.3 U^{2}OF Winter Performance

	mAP$_{0.50}$								
	Jan	Feb	Mar	Apr	May	Jun	Jul	Aug	Sep
Baseline 3ch - Winter	0.791	0.767	0.695	0.465	0.263	0.264	0.232	0.480	0.666
U^{2}OF - Winter	0.786	0.766	0.668	0.458	0.236	0.241	0.201	0.459	0.545

Table 9.7: mAP$_{0.50}$ results for U^{2}OF trained on winter data compared to the baseline model

In table 9.7 we see a unilateral decrease in performance, especially in September, reaching a difference of 0.169. It is expected that the winter performance to be lower than others, but not for all months, compared to the baseline.

	$mAP_{0.50:0.95}$								
	Jan	Feb	Mar	Apr	May	Jun	Jul	Aug	Sep
Baseline 3ch - Winter	0.462	0.403	0.351	0.228	0.123	0.126	0.102	0.244	0.286
U^2OF - Winter	0.460	0.404	0.332	0.223	0.106	0.112	0.086	0.228	0.270

Table 9.8: COCO mAP results for U^2OF trained on winter data compared to the baseline model

Table 9.8 that the gap between the U^2OF winter model is smaller compared to mAP_{50}.

9.2 Ablation Studies

Based on the shortcomings of the original test results, additional ablation studies were conducted to better understand them.

9.2.1 Baseline Channel Ablation

Baseline channel ablations were done to test the change in performance using less than 3 channels. This was done by changing the line 836 inside the "dataloaders.py" from the code shown in listing 3 to the code shown in listing 4

```
img = img.transpose((2, 0, 1))[::-1]
```

Listing 3: Line of code inside the dataloaders.py from the ultralytics YOLOv5 repository [3]

```
(B, G, R) = cv2.split(img)
zeroes = np.zeros(img.shape[:2], dtype="uint8")
img = cv2.merge([zeroes, G, zeroes])
img = img.transpose(2, 0, 1)
```

Listing 4: Changed code in dataloaders.py from the ultralytics YOLOv5 repository [3]

The continuous overlap of our U^2OF models with the baseline suggests our U^2Net and optical flow being comparable to 2 channels which are a duplicate of the source. To quantify what this amounts to, ablation studies were conducted, in which the baseline was stripped on one and two channels, by inputting a zero matrix in their place, effectively disabling them.

	mAP$_{0.50}$								
	Jan	Feb	Mar	Apr	May	Jun	Jul	Aug	Sep
Baseline 3ch - Summer	0.736	0.704	0.691	0.682	0.561	0.707	0.673	0.708	0.709
Baseline 2ch - Summer	0.736	0.701	0.685	0.685	0.556	0.709	0.678	0.705	0.679
Baseline 1ch - Summer	0.735	0.706	0.688	0.680	0.550	0.713	0.669	0.710	0.697

Table 9.9: mAP$_{0.50}$ results for the reducing channel numbers compared to 3 channel baseline model for baseline models trained on summer data

As can be seen in table 9.9 the difference between one, two and three channels of the source image is negligible. This suggests additional source images provide little to no benefit. The result for COCO *mAP* can be seen in table 9.10.

	mAP$_{0.50:0.95}$								
	Jan	Feb	Mar	Apr	May	Jun	Jul	Aug	Sep
Baseline 3ch - Summer	0.399	0.344	0.330	0.318	0.238	0.323	0.303	0.343	0.309
Baseline 2ch - Summer	0.398	0.344	0.331	0.323	0.239	0.326	0.306	0.343	0.311
Baseline 1ch - Summer	0.389	0.337	0.321	0.309	0.230	0.320	0.296	0.336	0.309

Table 9.10: COCO mAP results for the reducing channel numbers compared to 3 channel baseline model for baseline models trained on summer data

Observing table 9.10 we again see little tangible benefit, with a slight decrease in 1 channel performance.

	mAP$_{0.50}$								
	Jan	Feb	Mar	Apr	May	Jun	Jul	Aug	Sep
Baseline 3ch - Winter	0.791	0.767	0.695	0.465	0.263	0.264	0.232	0.480	0.666
Baseline 2ch - Winter	0.787	0.765	0.694	0.488	0.281	0.280	0.236	0.494	0.709
Baseline 1ch - Winter	0.789	0.764	0.688	0.472	0.285	0.274	0.234	0.465	0.687

Table 9.11: mAP$_{0.50}$ results for the reducing channel numbers compared to 3 channel baseline model for baseline models trained on winter data

When observing the one, two, and three-channel variants of the winter baseline in table 9.11, no substantial difference is found in January and February. It is noteworthy that the 3-channel baseline appears to be overfitting slightly and therefore has a lower performance outside the months it was trained on. The result for COCO *mAP* can be seen in table 9.12.

	$mAP_{0.50:0.95}$								
	Jan	Feb	Mar	Apr	May	Jun	Jul	Aug	Sep
Baseline 3ch - Winter	0.462	0.403	0.351	0.228	0.123	0.126	0.102	0.244	0.286
Baseline 2ch - Winter	0.460	0.401	0.347	0.236	0.130	0.130	0.102	0.251	0.316
Baseline 1ch - Winter	0.462	0.401	0.346	0.230	0.129	0.127	0.101	0.234	0.300

Table 9.12: COCO mAP results for the reducing channel numbers compared to 3 channel baseline model for baseline models trained on winter data

Table 9.12 shows that the 3-channel winter baseline has a slight edge for the first three months and worse performance outside these supporting previous findings.

9.2.2 Alternative baseline ablation

An ablation was done for the baseline with a more varying amount of data. Here the training set for February, May, and July was used to train the model. These months were chosen based on them having the worst performance in their seasons. The expected result of this study is that the performance of the model in the months it is trained on is increased while also mitigating performance lost to concept drift. The result between all baselines and the model with February, May, and July data can be seen in table 9.13

	$mAP_{0.50}$								
	Jan	Feb	Mar	Apr	May	Jun	Jul	Aug	Sep
Baseline 3ch - Winter	0.791	0.767	0.695	0.465	0.263	0.264	0.232	0.480	0.666
Baseline 3ch - Spring	0.777	0.762	0.747	0.723	0.588	0.654	0.654	0.675	0.661
Baseline 3ch - Summer	0.736	0.704	0.691	0.682	0.561	0.707	0.673	0.708	0.709
Baseline 3ch - Feb+May+Jul	0.774	0.762	0.737	0.690	0.574	0.653	0.654	0.670	0.681

Table 9.13: $mAP_{0.50}$ results for using varied data

In table 9.13 the performance for the varied baseline is mediocre, never being worst or best for any month. It can also be seen that performance for February, May, and July didn't increase as hypothesized. This might suggest that the data was too varied and the model couldn't get a correct fit. It might also suggest that the problems from May were amplified. The result for COCO *mAP* can be seen in table 9.14.

	$mAP_{0.50:0.95}$								
	Jan	Feb	Mar	Apr	May	Jun	Jul	Aug	Sep
Baseline 3ch - Winter	0.462	0.403	0.351	0.228	0.123	0.126	0.102	0.244	0.286
Baseline 3ch - Spring	0.441	0.387	0.371	0.347	0.254	0.304	0.290	0.322	0.306
Baseline 3ch - Summer	0.399	0.344	0.330	0.318	0.238	0.323	0.303	0.343	0.309
Baseline 3ch - Feb+May+Jul	0.430	0.384	0.353	0.321	0.243	0.293	0.278	0.317	0.308

Table 9.14: COCO mAP results for using varied data

As seen in table 9.14 the performance for the varied baseline can again be observed to yield mediocre performance across the board, supporting previous findings.

9.2.3 U^{2}Net Ablation

To understand the performance impact of U^{2}Net, various configurations were tested to see performance yields.

The measurements in table 9.15 show the result of using only the U^{2}Net output as the secondary channel. This was then compared to the 2-channel baseline.

	$mAP_{0.50}$								
	Jan	Feb	Mar	Apr	May	Jun	Jul	Aug	Sep
Baseline 2ch - Summer	0.736	0.701	0.685	0.685	0.556	0.709	0.678	0.705	0.679
U^2 - Summer	0.724	0.697	0.677	0.641	0.527	0.688	0.672	0.703	0.681

Table 9.15: $mAP_{0.50}$ results for U^2 trained on summer data compared to the baseline model

As seen in table 9.15 the overall performance is worse, only beating the baseline for September, by 0.002, and being outperformed in April by 0.044. This might suggest that U^{2}Net does not impact the performance of pedestrian detection positively. The result for COCO mAP can be seen in table 9.16.

	$mAP_{0.50:0.95}$								
	Jan	Feb	Mar	Apr	May	Jun	Jul	Aug	Sep
Baseline 2ch - Summer	0.398	0.344	0.331	0.323	0.239	0.326	0.306	0.343	0.311
U^2 - Summer	0.404	0.350	0.330	0.303	0.221	0.319	0.295	0.338	0.299

Table 9.16: COCO mAP results for U^2 trained on summer data compared to the baseline model

In table 9.16 it can be seen that the U^{2}Net model outperforms the baseline in winter months, suggesting better IoU scores for those months. For some months the gap has widened, proving that the performance of the model has shifted away from those months, focusing the high IoU scores on the winter months.

An additional experiment was also conducted where a copy of the U^{2}Net channel was used as a third channel. The $mAP_{0.50}$ results can be seen in table 9.17

	mAP$_{0.50}$								
	Jan	Feb	Mar	Apr	May	Jun	Jul	Aug	Sep
Baseline 3ch - Summer	0.736	0.704	0.691	0.682	0.561	0.707	0.673	0.708	0.709
U^{2}x2 - Summer	0.726	0.701	0.680	0.644	0.536	0.694	0.669	0.704	0.676

Table 9.17: mAP$_{0.50}$ results for U^{2}x2 trained on summer data compared to the baseline model

As seen in table 9.17 there is a minor increase in performance for the first half of the data, when compared to 9.15, and a subsequent drop in some later months. When compared to the baseline, the performance is inferior for all months. The result for COCO mAP can be seen in table 9.18.

	mAP$_{0.50:0.95}$								
	Jan	Feb	Mar	Apr	May	Jun	Jul	Aug	Sep
Baseline 3ch - Summer	0.399	0.344	0.330	0.318	0.238	0.323	0.303	0.343	0.309
U^{2}x2 - Summer	0.413	0.358	0.338	0.309	0.229	0.325	0.297	0.344	0.299

Table 9.18: COCO mAP results for U^{2}x2 trained on summer data compared to the baseline model

In table 9.18 it can be observed that the model outperforms the baseline in January, February, March, June and August, showing better IoU scores for most months.

9.2.4 Optical Flow Ablation

Additional ablation studies were done using optical flow as the only channel to show the impact it has on performance. The $mAP_{0.50}$ results can be seen in 9.19.

	mAP$_{0.50}$								
	Jan	Feb	Mar	Apr	May	Jun	Jul	Aug	Sep
Baseline 2ch - Summer	0.736	0.701	0.685	0.685	0.556	0.709	0.678	0.705	0.679
OF - Summer	0.749	0.725	0.706	0.685	0.561	0.705	0.672	0.706	0.712

Table 9.19: mAP$_{0.50}$ results for OF trained on summer data compared to the baseline model

In table 9.19 it can be seen that the model meets or exceeds the performance of the baseline for all months but June and July. This improvement is of substantial size. The result for COCO mAP can be seen in table 9.20

	$mAP_{0.50:0.95}$								
	Jan	Feb	Mar	Apr	May	Jun	Jul	Aug	Sep
Baseline 2ch - Summer	0.398	0.344	0.331	0.323	0.239	0.326	0.306	0.343	0.311
OF - Summer	0.410	0.356	0.338	0.320	0.234	0.322	0.294	0.339	0.319

Table 9.20: COCO mAP results for OF trained on summer data compared to the baseline model

when looking at table 9.20 it can be seen that the performance is slightly shifted in favor of winter months, being surpassed in summer months.

Using 2 channels for optical flow was also studied the results for $mAP_{0.50}$ can be seen in table 9.21

	$mAP_{0.50}$								
	Jan	Feb	Mar	Apr	May	Jun	Jul	Aug	Sep
Baseline 3ch - Summer	0.736	0.704	0.691	0.682	0.561	0.707	0.673	0.708	0.709
OFx2 - Summer	0.757	0.732	0.714	0.688	0.558	0.709	0.672	0.708	0.705

Table 9.21: $mAP_{0.50}$ results for OFx2 trained on summer data compared to the baseline model

When comparing table 9.21 with table 9.19, we see a bigger improvement, for most months, but a drop in May. When looking at the baseline it outperforms the model in May, July and September but by the slightest margin.

	$mAP_{0.50:0.95}$								
	Jan	Feb	Mar	Apr	May	Jun	Jul	Aug	Sep
Baseline 3ch - Summer	0.399	0.344	0.330	0.318	0.238	0.323	0.303	0.343	0.309
OFx2 - Summer	0.405	0.354	0.337	0.318	0.230	0.322	0.292	0.338	0.307

Table 9.22: COCO mAP results for OFx2 trained on summer data compared to the baseline model

When observing the COCO mAP in table 9.22 we again see the winter-shifted distribution from the optical flow model, seen previously.

9.2.5 YOLOv5 Confidence Ablation

The confidence parameter of YOLOv5 has a default value of 0.001. This is a very low value, resulting in anything detected in the slightest is considered a find. This parameter was swept to see its influence on the results.

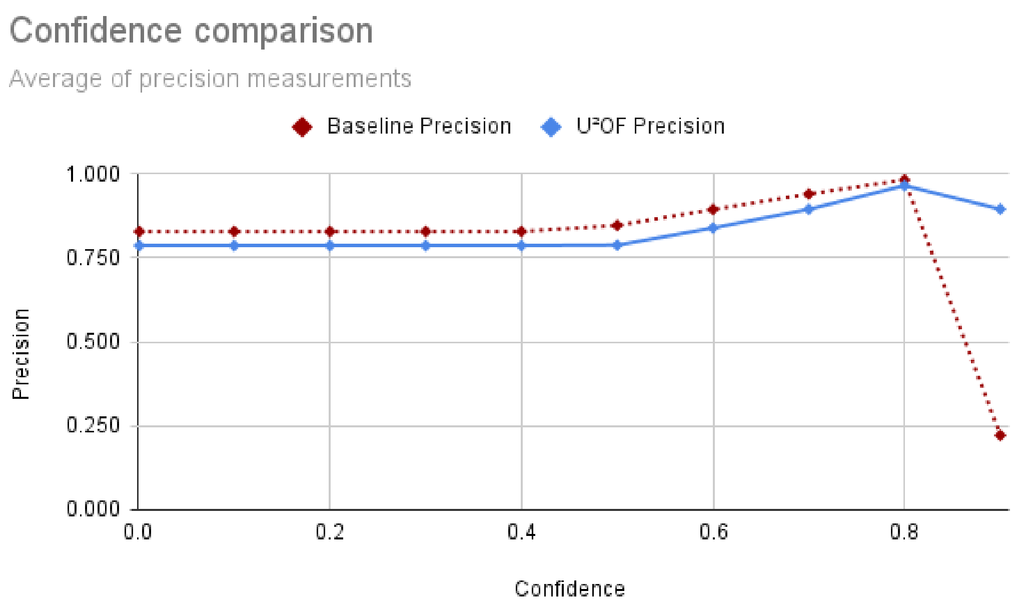


Figure 9.2: Graph showing the confidence to precision relation of the baseline and U^{2}OF

	Baseline Confidence parameter - mAP$_{0.50}$								
	Jan	Feb	Mar	Apr	May	Jun	Jul	Aug	Sep
Baseline Conf 0.001	0.736	0.704	0.691	0.682	0.561	0.707	0.673	0.708	0.709
Baseline Conf 0.1	0.766	0.738	0.731	0.716	0.599	0.736	0.702	0.730	0.736
Baseline Conf 0.2	0.776	0.748	0.742	0.726	0.613	0.746	0.713	0.740	0.745
Baseline Conf 0.3	0.781	0.754	0.748	0.733	0.623	0.753	0.721	0.747	0.749
Baseline Conf 0.4	0.787	0.760	0.755	0.739	0.633	0.757	0.726	0.752	0.749
Baseline Conf 0.5	0.789	0.765	0.759	0.739	0.637	0.751	0.724	0.752	0.743
Baseline Conf 0.6	0.771	0.750	0.744	0.723	0.619	0.719	0.699	0.726	0.743
Baseline Conf 0.7	0.728	0.712	0.701	0.673	0.583	0.654	0.640	0.670	0.707
Baseline Conf 0.8	0.626	0.624	0.602	0.568	0.514	0.558	0.540	0.576	0.625
Baseline Conf 0.9	0.500	0.000	0.000	0.500	0.000	0.000	0.000	0.000	0.000

Table 9.23: Baseline mAP$_{0.50}$ as a function of confidence parameter and time of year

When observing table 9.23 it can be seen that the best baseline performance is achieved with a confidence parameter of 0.4 or 0.5, depending on whether winter or summer months are prioritized. This can be used to optimize further for performance.

	Baseline Confidence parameter - $mAP_{0.50}$								
	Jan	Feb	Mar	Apr	May	Jun	Jul	Aug	Sep
U^2OF Conf 0.001	0.728	0.698	0.682	0.641	0.537	0.691	0.676	0.702	0.700
U^2OF Conf 0.1	0.757	0.730	0.724	0.683	0.575	0.720	0.702	0.725	0.736
U^2OF Conf 0.2	0.766	0.740	0.735	0.695	0.589	0.729	0.713	0.734	0.744
U^2OF Conf 0.3	0.772	0.747	0.743	0.704	0.600	0.736	0.719	0.739	0.750
U^2OF Conf 0.4	0.778	0.755	0.752	0.716	0.612	0.743	0.725	0.746	0.751
U^2OF Conf 0.5	0.783	0.762	0.759	0.721	0.617	0.742	0.724	0.744	0.746
U^2OF Conf 0.6	0.769	0.753	0.747	0.712	0.607	0.720	0.708	0.724	0.743
U^2OF Conf 0.7	0.736	0.725	0.712	0.681	0.578	0.674	0.670	0.678	0.735
U^2OF Conf 0.8	0.649	0.646	0.622	0.591	0.526	0.575	0.572	0.590	0.665
U^2OF Conf 0.9	0.451	0.500	0.455	0.500	0.125	0.500	0.500	0.500	0.500

Table 9.24: U^2OF $mAP_{0.50}$ as a function of confidence parameter and time of year

When comparing table 9.24 to table 9.23, we see that the difference is mostly consistent with the performance difference observed in the original U^2OF models, showing this may not be useful for mitigating concept drift.

	U^2OF Confidence parameter - Precision								
	Jan	Feb	Mar	Apr	May	Jun	Jul	Aug	Sep
U^2OF Conf 0.001	0.832	0.825	0.844	0.794	0.690	0.808	0.775	0.795	0.716
U^2OF Conf 0.1	0.832	0.825	0.844	0.794	0.690	0.808	0.775	0.795	0.716
U^2OF Conf 0.2	0.832	0.825	0.844	0.794	0.690	0.808	0.775	0.795	0.716
U^2OF Conf 0.3	0.832	0.825	0.844	0.794	0.690	0.808	0.775	0.795	0.716
U^2OF Conf 0.4	0.832	0.825	0.844	0.794	0.690	0.808	0.775	0.795	0.716
U^2OF Conf 0.5	0.820	0.805	0.837	0.793	0.698	0.825	0.794	0.804	0.716
U^2OF Conf 0.6	0.872	0.865	0.886	0.848	0.750	0.876	0.862	0.862	0.732
U^2OF Conf 0.7	0.928	0.923	0.926	0.909	0.803	0.922	0.929	0.917	0.798
U^2OF Conf 0.8	0.982	0.983	0.976	0.966	0.892	0.963	0.980	0.965	0.982
U^2OF Conf 0.9	0.900	1.000	0.909	1.000	0.250	1.000	1.000	1.000	1.000

Table 9.25: U^2OF precision as a function of confidence parameter and time of year

Observing the table 9.25, an anomaly can be observed for 0.9 confidence in May.

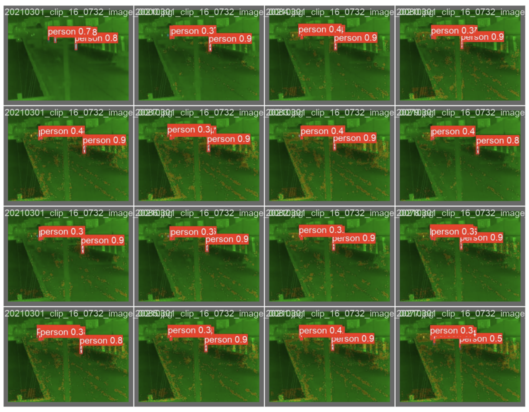
9.2.6 May clip ablation

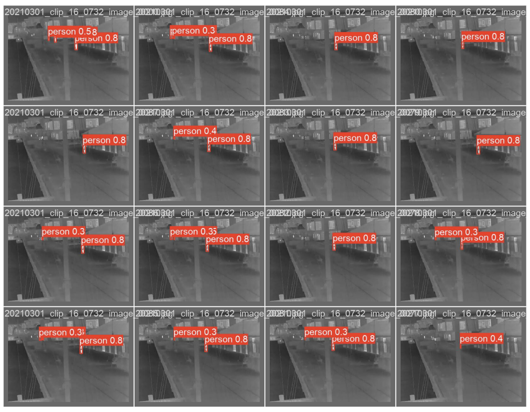
An additional ablation was done after the discovered results from the confidence ablation. This ablation study focuses on a single clip in May because the confidence result shown in table 9.25 was outside the expectation. With a confidence of 0.9, the expected precision should be either 0 or 0.9 and above, but unexpectedly the results show that the precision is 0.250 which means that the model only got 1 out of 4 detections as *TP*. The ablation study was done on clip 16 for May 1st and the results can be seen in table 9.26

	Precision	Recall	$\text{mAP}_{0.50}$	$\text{mAP}_{0.50:0.95}$
Baseline	0.885	0.790	0.875	0.423
U^2OF	0.756	0.811	0.833	0.448

Table 9.26: The result metrics for May 1st, clip 16

The results from table 9.26 show that the precision and $\text{mAP}_{0.50}$ is significantly worse in the proposed model whereas the recall and $\text{mAP}_{0.50:0.95}$ is slightly better. Saved predictions and the corresponding labels can be seen in figure 9.3 these can be seen in higher resolution in A.4


(a)


(b)

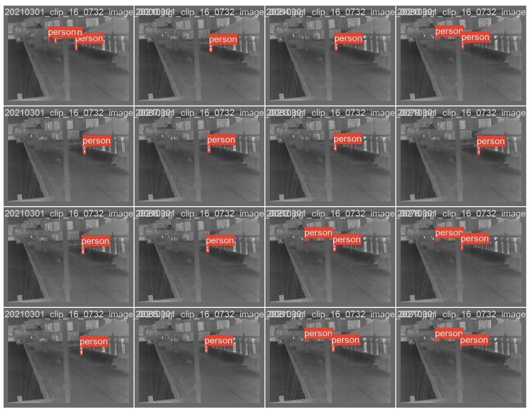

(c)

Figure 9.3: The saved batches by the YOLOv5 model for May 1st clip 16. (a) The batch prediction saved by YOLOv5 for the proposed models. (b) The batch predictions saved for the baseline. (c) The ground truth labels for the image batches

Figure 9.3 shows that the baseline and the proposed model find pedestrians who are not labelled, but it also shows that the proposed model finds pedestrians who are not labelled in higher quantity and also finds pedestrians with higher confidence. This disproportionally affects the results for the proposed model negatively, which might suggest that the actual performance of the proposed model exceeds the baseline performance.

Figure 9.4 depicts the discrepancy between the baseline detection and the U^{2}OF detection. It can be seen that the U^{2}OF detects multiple people in the back, which the baseline does not. These people are not annotated, and as such are considered errors in the metric, despite superior detection.

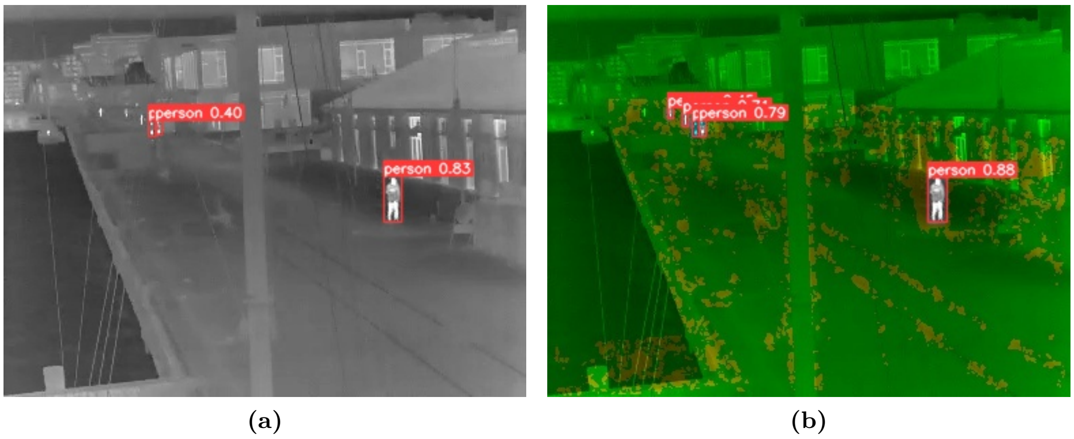


Figure 9.4: Detection comparison for (a) baseline (b) U^{2}OF

Chapter 10

Discussion

This study aimed to determine whether seasonal concept drift in thermal images can be mitigated by adapting the original domain to a complex representation for pedestrian detection. The results of this work have found that while the method demonstrated minimal overall performance improvement in initial tests, further examinations revealed that the dataset used for testing contained significant annotation errors. Specifically, the proposed implementation successfully detected pedestrians that were not annotated in the dataset, highlighting the dataset's shortcomings rather than the ineffectiveness of our approach. The reliance on a faulty dataset has impacted the validity of our results. The minimal improvement may be a consequence of this annotation error, but could also be a result of negligible improvement. It is at this time not possible to evaluate quantitatively the performance of the solution, without correcting or replacing the dataset. It is however possible to partially evaluate the solution, based on trends. We have the continuous trend of optical flow models having improved performance in winter on COCO mAP measurements, when compared to $\text{mAP}_{0.50}$, showing that optical flow has improved the IoU scores for winter measurement, and has generally improved other measurements as well. This does eliminate some concept drift, as the models for the ablation studies were run on summer-based models. A similar result was seen in the COCO mAP related to the U^2Net ablation studies, albeit without the overall performance provided by optical flow. Overall our research suggests that this approach may be viable, as the results show relatively good performance, despite being penalized for better performance.

This work ultimately proves that the approach to mitigating concept drift is viable, as it has shown the ability to detect people, which the baseline cannot in the same frames. The upper limitations of our method cannot be evaluated without a completely annotated dataset but are expected to be an overall improvement.

The errors in the dataset could pose problems for existing research based on it, as their results may in some cases be invalidated, and their findings along with it.

Future research should be acutely aware of the shortcomings of the dataset, and attempt to rectify or mitigate them in their research. The method has room for further testing, with various representations, to figure out if others may yield better performance overall. It would also be of interest to experiment with other models than YOLOv5, to see how the methodology affects them differently.

Chapter 11

Conclusion

The goal of this work was to mitigate concept drift in thermal images, utilizing complex representations. Our work has proven to be able to mitigate some concept drift, when going from summer to winter, but ultimately cannot show its full potential due to missing annotations in the dataset. The results highlight the shortcomings of the dataset and some of the nuances of the implementation. It shows the importance of quality and consistency in datasets. Future work should focus on acquiring a more complete dataset, before revisiting this approach and extending the list of tested representations, to find potentially better representations.

Bibliography

[1] X. Qin, Z. Zhang, C. Huang, M. Dehghan, O. R. Zaiane, and M. Jagersand, “U-2-net: Going deeper with nested u-structure for salient object detection,” eng, *Pattern recognition*, vol. 106, 2020, ISSN: 0031-3203.

[2] G. Farnebäck, “Two-frame motion estimation based on polynomial expansion,” 2003, pp. 363–370, ISBN: 978-3-540-45103-7.

[3] G. Jocher, A. Chaurasia, A. Stoken, *et al.*, *ultralytics/yolov5: v7.0 - YOLOv5 SOTA Realtime Instance Segmentation*, version v7.0, Nov. 2022. DOI: `10.5281/zenodo.7347926`. [Online]. Available: `https://doi.org/10.5281/zenodo.7347926`.

[4] M. Guan, C. Wen, M. Shan, C.-L. Ng, and Y. Zou, “Real-time event-triggered object tracking in the presence of model drift and occlusion,” *IEEE Transactions on Industrial Electronics*, vol. 66, no. 3, pp. 2054–2065, 2019. DOI: `10.1109/TIE.2018.2835390`.

[5] I. Žliobaitė, M. Pechenizkiy, and J. Gama, “An overview of concept drift applications,” in *Big Data Analysis: New Algorithms for a New Society*, N. Japkowicz and J. Stefanowski, Eds. Cham: Springer International Publishing, 2016, pp. 91–114, ISBN: 978-3-319-26989-4. DOI: `10.1007/978-3-319-26989-4_4`. [Online]. Available: `https://doi.org/10.1007/978-3-319-26989-4_4`.

[6] I. Nikolov, M. Philipsen, J. Liu, *et al.*, “Seasons in drift: A long-term thermal imaging dataset for studying concept drift,” English, in *Thirty-fifth Conference on Neural Information Processing Systems 2021*, Thirty-fifth Conference on Neural Information Processing Systems, NeurIPS 2021 ; Conference date: 06-12-2021 Through 14-12-2021, Neural Information Processing Systems Foundation, 2021. [Online]. Available: `https://neurips.cc/`.

[7] A. S. Johansen, K. Nasrollahi, S. Escalera, and T. B. Moeslund, “Who cares about the weather? inferring weather conditions for weather-aware object detection in thermal images,” *Applied Sciences*, vol. 13, no. 18, 2023, ISSN: 2076-3417. DOI: `10.3390/app131810295`. [Online]. Available: `https://www.mdpi.com/2076-3417/13/18/10295`.

[8] M. Kieu, A. D. Bagdanov, M. Bertini, and A. del Bimbo, “Task-conditioned domain adaptation for pedestrian detection in thermal imagery,” in *Computer Vision – ECCV 2020*, A. Vedaldi, H. Bischof, T. Brox, and J.-M. Frahm, Eds., Cham: Springer International Publishing, 2020, pp. 546–562, ISBN: 978-3-030-58542-6.

[9] K. My, A. Bagdanov, M. Bertini, and A. Bimbo, “Domain adaptation for privacy-preserving pedestrian detection in thermal imagery,” in Sep. 2019, pp. 203–213, ISBN: 978-3-030-30644-1. DOI: `10.1007/978-3-030-30645-8_19`.

[10] D. König, M. Adam, C. Jarvers, G. Layher, H. Neumann, and M. Teutsch, “Fully convolutional region proposal networks for multispectral person detection,” in *2017 IEEE Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, 2017, pp. 243–250. DOI: `10.1109/CVPRW.2017.36`.

[11] R. Gade and T. B. Moeslund, “Thermal cameras and applications: A survey,” eng, *Machine vision and applications*, vol. 25, no. 1, pp. 245–262, 2014, ISSN: 0932-8092.

[12] H. Budzier and G. Gerlach, *Thermal Infrared Sensors: Theory, Optimisation and Practice*, eng, 1. Aufl. Newark: Wiley, 2011, ISBN: 9780470871928.

[13] J. S. Byrnes, *Unexploded ordnance detection and mitigation* (NATO Science for Peace and Security Series B: Physics and Biophysics), eng, 1st ed. 2009. Dordrecht, Netherlands: Springer, 2009, ISBN: 1-281-95474-8.

[14] M. Vollmer, “Infrared thermal imaging,” eng, in *Computer Vision*, Cham: Springer International Publishing, 2021, pp. 666–670, ISBN: 3030634159.

[15] S. Lee and S. H. Park, “Concept drift modeling for robust autonomous vehicle control systems in time-varying traffic environments,” eng, *Expert systems with applications*, vol. 190, pp. 116 206–, 2022, ISSN: 0957-4174.

[16] T. Arsen, *Domain adaptation*, 2024. [Online]. Available: `https://sbert.net/examples/domain_adaptation/README.html`.

[17] R. Kundu, *Domain adaptation in computer vision: Everything you need to know*, 2022. [Online]. Available: `https://www.v7labs.com/blog/domain-adaptation-guide`.

[18] D. Konig, M. Adam, C. Jarvers, G. Layher, H. Neumann, and M. Teutsch, “Fully convolutional region proposal networks for multispectral person detection,” eng, in *2017 IEEE Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, vol. 2017-, IEEE, 2017, pp. 243–250, ISBN: 1538607336.

[19] K. Simonyan and A. Zisserman, “Very deep convolutional networks for large-scale image recognition,” eng, *arXiv.org*, 2015, ISSN: 2331-8422.

[20] D. Guan, Y. Cao, J. Yang, Y. Cao, and M. Y. Yang, "Fusion of multispectral data through illumination-aware deep neural networks for pedestrian detection," eng, *Information fusion*, vol. 50, pp. 148–157, 2019, ISSN: 1566-2535.

[21] C. Li, D. Song, R. Tong, and M. Tang, "Illumination-aware faster r-cnn for robust multispectral pedestrian detection," eng, *Pattern recognition*, vol. 85, pp. 161–171, 2019, ISSN: 0031-3203.

[22] C. Li, D. Song, R. Tong, and M. Tang, "Multispectral pedestrian detection via simultaneous detection and segmentation," eng, *arXiv.org*, 2018, ISSN: 2331-8422.

[23] R. Li, J. Xiang, F. Sun, Y. Yuan, L. Yuan, and S. Gou, "Multiscale cross-modal homogeneity enhancement and confidence-aware fusion for multispectral pedestrian detection," eng, *IEEE transactions on multimedia*, vol. 26, pp. 1–13, 2024, ISSN: 1520-9210.

[24] M. A. Galarza-Bravo and M. J. Flores-Calero, "Pedestrian detection at night based on faster r-cnn and far infrared images," eng, in *Intelligent Robotics and Applications*, vol. 10985, Cham: Springer International Publishing, 2018, pp. 335–345, ISBN: 9783319975887.

[25] Y. Chen and H. Shin, "Pedestrian detection at night in infrared images using an attention-guided encoder-decoder convolutional neural network," *Applied Sciences*, vol. 10, no. 3, 2020, ISSN: 2076-3417. DOI: `10.3390/app10030809`. [Online]. Available: `https://www.mdpi.com/2076-3417/10/3/809`.

[26] X. Dai, X. Yuan, and X. Wei, "Tirnet: Object detection in thermal infrared images for autonomous driving," eng, *Applied intelligence (Dordrecht, Netherlands)*, vol. 51, no. 3, pp. 1244–1261, 2021, ISSN: 0924-669X.

[27] Y. Cao, T. Zhou, X. Zhu, and Y. Su, "Every feature counts: An improved one-stage detector in thermal imagery," in *2019 IEEE 5th International Conference on Computer and Communications (ICCC)*, 2019, pp. 1965–1969. DOI: `10.1109/ICCC47050.2019.9064036`.

[28] H. Patel, K. Prajapati, A. Sarvaiya, *et al.*, "Depthwise convolution for compact object detector in nighttime images," eng, in *2022 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, Piscataway: IEEE, 2022, pp. 378–388, ISBN: 1665487399.

[29] D. Heo, E. Lee, and B. C. Ko, "Pedestrian detection at night using deep neural networks and saliency maps," eng, *Electronic Imaging*, no. 17, pp. 060 403–1–060403–9, 2017, ISSN: 2470-1173.

[30] D. Ghose, S. M. Desai, S. Bhattacharya, D. Chakraborty, M. Fiterau, and T. Rahman, "Pedestrian detection in thermal images using saliency maps," eng, in *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, vol. 2019-, Piscataway: IEEE, 2019, pp. 988–997, ISBN: 1728125065.

[31] K. My, A. Bagdanov, M. Bertini, and A. Bimbo, "Domain adaptation for privacy-preserving pedestrian detection in thermal imagery," in Sep. 2019, pp. 203–213, ISBN: 978-3-030-30644-1. DOI: 10.1007/978-3-030-30645-8_19.

[32] K. My, A. Bagdanov, M. Bertini, and A. Bimbo, "Task-conditioned domain adaptation for pedestrian detection in thermal imagery," in Nov. 2020, pp. 546–562, ISBN: 978-3-030-58541-9. DOI: 10.1007/978-3-030-58542-6_33.

[33] K. My, L. Berlincioni, L. Galteri, M. Bertini, A. Bagdanov, and A. Bimbo, "Robust pedestrian detection in thermal imagery using synthesized images," Jan. 2021, pp. 8804–8811. DOI: 10.1109/ICPR48806.2021.9412764.

[34] F. Munir, S. Azam, M. A. Rafique, A. M. Sheri, M. Jeon, and W. Pedrycz, *Exploring thermal images for object detection in underexposure regions for autonomous driving*, 2021. arXiv: 2006.00821 [cs.CV].

[35] I. B. Akkaya, F. Altinel, and U. Halici, "Self-training guided adversarial domain adaptation for thermal imagery," in *2021 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, 2021, pp. 4317–4326. DOI: 10.1109/CVPRW53098.2021.00488.

[36] Y. Zoetgnande and J.-L. Dillenseger, "Domain generalization for activity recognition: Learn from visible, infer with thermal," in *11th International Conference on Pattern Recognition Applications and Methods*, Online Streaming, France: SCITEPRESS - Science and Technology Publications, Feb. 2022, pp. 722–729. DOI: 10.5220/0010906300003122. [Online]. Available: https://hal.science/hal-03588563.

[37] C. Luo, Y. Zhang, J. Guo, *et al.*, "Sar-cdss: A semi-supervised cross-domain object detection from optical to sar domain," *Remote Sensing*, vol. 16, p. 940, Mar. 2024. DOI: 10.3390/rs16060940.

[38] H. Li, S. J. Pan, S. Wang, and A. C. Kot, "Domain generalization with adversarial feature learning," in *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2018, pp. 5400–5409. DOI: 10.1109/CVPR.2018.00566.

[39] Z. Murez, S. Kolouri, D. Kriegman, R. Ramamoorthi, and K. Kim, "Image to image translation for domain adaptation," in *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, Los Alamitos, CA, USA: IEEE Computer Society, 2018, pp. 4500–4509. DOI: 10.1109/CVPR.2018.00473. [Online]. Available: https://doi.ieeecomputersociety.org/10.1109/CVPR.2018.00473.

[40] H. Zhou, F. Jiang, and H. Lu, “Ssda-yolo: Semi-supervised domain adaptive yolo for cross-domain object detection,” eng, *Computer vision and image understanding*, vol. 229, pp. 103 649–, 2023, ISSN: 1077-3142.

[41] J. Li and W. Gao, *Visual Saliency Computation: A Machine Learning Perspective* (Lecture Notes in Computer Science), eng. Cham: Springer International Publishing, vol. 8408, ISBN: 3319056417.

[42] J. Zhang, F. Malmberg, and S. Sclaroff, *Visual Saliency: From Pixel-Level to Object-Level Analysis*, eng, Elektronisk udgave. Cham: Springer International Publishing, 2019, ISBN: 9783030048310.

[43] C. Yang, L. Zhang, H. Lu, X. Ruan, and M.-H. Yang, “Saliency detection via graph-based manifold ranking,” eng, in *2013 IEEE Conference on Computer Vision and Pattern Recognition*, IEEE, 2013, pp. 3166–3173, ISBN: 9781538656723.

[44] A. Garcia-Diaz, X. R. Fdez-Vidal, X. M. Pardo, and R. Dosil, “Saliency from hierarchical adaptation through decorrelation and variance normalization,” eng, *Image and vision computing*, vol. 30, no. 1, pp. 51–64, 2012, ISSN: 0262-8856.

[45] B. Jiang, L. Zhang, H. Lu, C. Yang, and M.-H. Yang, “Saliency detection via absorbing markov chain,” eng, in *2013 IEEE International Conference on Computer Vision*, IEEE, 2013, pp. 1665–1672, ISBN: 1479928402.

[46] N. Liu, J. Han, and M.-H. Yang, “Picanet: Learning pixel-wise contextual attention for saliency detection,” eng, in *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, IEEE, 2018, pp. 3089–3098, ISBN: 9781538664209.

[47] F. Visin, K. Kastner, K. Cho, M. Matteucci, A. Courville, and Y. Bengio, “Renet: A recurrent neural network based alternative to convolutional networks,” eng, *arXiv.org*, 2015, ISSN: 2331-8422.

[48] O. Ronneberger, P. Fischer, and T. Brox, “U-net: Convolutional networks for biomedical image segmentation,” eng, in *Medical Image Computing and Computer-Assisted Intervention – MICCAI 2015*, ser. Lecture Notes in Computer Science, Cham: Springer International Publishing, pp. 234–241, ISBN: 9783319245737.

[49] Z. Deng, X. Hu, L. Zhu, *et al.*, “R^3net: Recurrent residual refinement network for saliency detection,” in *Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence, IJCAI-18*, International Joint Conferences on Artificial Intelligence Organization, Jul. 2018, pp. 684–690. DOI: `10.24963/ijcai.2018/95`. [Online]. Available: `https://doi.org/10.24963/ijcai.2018/95`.

[50] S. Xie, R. Girshick, P. Dollar, Z. Tu, and K. He, “Aggregated residual transformations for deep neural networks,” eng, in *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, IEEE, 2017, pp. 5987–5995, ISBN: 1538604574.

[51] OpenCV, *Optical flow*, 2024. [Online]. Available: `https://docs.opencv.org/4.x/db/d7f/tutorial_js_lucas_kanade.html`.

[52] G. Boesch, *Optical flow – everything you need to know in 2024*, 2024. [Online]. Available: `https://viso.ai/deep-learning/optical-flow/`.

[53] P. Fischer, A. Dosovitskiy, E. Ilg, *et al.*, *Flownet: Learning optical flow with convolutional networks*, 2015. arXiv: `1504.06852 [cs.CV]`.

[54] Z. Teed and J. Deng, *Raft: Recurrent all-pairs field transforms for optical flow*, 2020. arXiv: `2003.12039 [cs.CV]`.

[55] S. Ren, K. He, R. Girshick, and J. Sun, "Faster r-cnn: Towards real-time object detection with region proposal networks," eng, *IEEE transactions on pattern analysis and machine intelligence*, vol. 39, no. 6, pp. 1137–1149, 2017, ISSN: 0162-8828.

[56] R. Girshick, "Fast r-cnn," eng, in *2015 IEEE International Conference on Computer Vision (ICCV)*, IEEE, 2015, pp. 1440–1448, ISBN: 1467383910.

[57] M. D. Zeiler and R. Fergus, "Visualizing and understanding convolutional networks," eng, in *Computer Vision – ECCV 2014*, ser. Lecture Notes in Computer Science, Cham: Springer International Publishing, pp. 818–833, ISBN: 3319105892.

[58] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You only look once: Unified, real-time object detection," eng, in *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, IEEE, 2016, pp. 779–788, ISBN: 9781467388511.

[59] C. Szegedy, W. Liu, Y. Jia, *et al.*, "Going deeper with convolutions," eng, in *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, vol. 7-12-, IEEE, 2015, pp. 1–9, ISBN: 1467369640.

[60] J. Redmon and A. Farhadi, "Yolo9000: Better, faster, stronger," eng, in *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, IEEE, 2017, pp. 6517–6525, ISBN: 1538604574.

[61] J. Redmon and A. Farhadi, "Yolov3: An incremental improvement," eng, *arXiv.org*, 2018, ISSN: 2331-8422.

[62] T.-Y. Lin, P. Dollar, R. Girshick, K. He, B. Hariharan, and S. Belongie, "Feature pyramid networks for object detection," eng, in *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, IEEE, 2017, pp. 936–944, ISBN: 1538604574.

[63] A. Bochkovskiy, W. Chien-Yao, and M. L. Hong-Yuan, "Yolov4: Optimal speed and accuracy of object detection," eng, *arXiv.org*, 2020, ISSN: 2331-8422.

[64] M. Hussain, "Yolov1 to v8: Unveiling each variant - a comprehensive review of yolo," eng, *IEEE access*, vol. 12, pp. 1–1, 2024, ISSN: 2169-3536.

[65] A. Anwar, *What is average precision in object detection localization algorithms and how to calculate it?* 2022. [Online]. Available: `https://towardsdatascience.com/what-is-average-precision-in-object-detection-localization-algorithms-and-how-to-calculate-it-3f330efe697b`.

[66] D. M. W. Powers, "Evaluation: From precision, recall and f-measure to roc, informedness, markedness and correlation," eng, *arXiv (Cornell University)*, 2020, ISSN: 2331-8422.

[67] J. Hui, *Map (mean average precision) for object detection*, 2019. [Online]. Available: `https://jonathan-hui.medium.com/map-mean-average-precision-for-object-detection-45c121a31173`.

[68] D. P. Kingma and J. Ba, *Adam: A method for stochastic optimization*, 2017. arXiv: `1412.6980 [cs.LG]`.

Appendix A

Appendix

A.1 Anaconda environment

```
name: Name
channels:
  - pytorch
  - nvidia
  - conda-forge
  - defaults
dependencies:
  - _libgcc_mutex=0.1=conda_forge
  - _openmp_mutex=4.5=2_gnu
  - alsa-lib=1.2.8=h166bdaf_0
  - aom=3.5.0=h27087fc_0
  - appdirs=1.4.4=pyhd3eb1b0_0
  - attr=2.5.1=h166bdaf_1
  - blas=1.0=mkl
  - blosc=1.21.3=h6a678d5_0
  - bottleneck=1.3.7=py311hf4808d0_0
  - brotli=1.0.9=h5eee18b_7
  - brotli-bin=1.0.9=h5eee18b_7
  - brotli-python=1.0.9=py311h6a678d5_7
  - brunsli=0.1=h2531618_0
  - bzip2=1.0.8=h5eee18b_5
  - c-ares=1.19.1=h5eee18b_0
  - ca-certificates=2024.3.11=h06a4308_0
  - cairo=1.16.0=ha61ee94_1014
  - certifi=2024.2.2=py311h06a4308_0
  - cfitsio=3.470=h5893167_7
  - charls=2.2.0=h2531618_0
  - charset-normalizer=2.0.4=pyhd3eb1b0_0
  - click=8.1.7=py311h06a4308_0
  - contourpy=1.2.0=py311hdb19cb5_0
  - cuda-cudart=12.1.105=0
  - cuda-cupti=12.1.105=0
  - cuda-libraries=12.1.0=0
  - cuda-nvrtc=12.1.105=0
  - cuda-nvtx=12.1.105=0
  - cuda-opencl=12.4.127=0
  - cuda-runtime=12.1.0=0
  - cycler=0.11.0=pyhd3eb1b0_0
  - dav1d=1.2.1=h5eee18b_0
  - dbus=1.13.18=hb2f20db_0
  - docker-pycreds=0.4.0=pyhd3eb1b0_0
  - expat=2.5.0=h6a678d5_0
  - ffmpeg=5.1.2=gpl_h8dda1f0_106
  - fftw=3.3.10=nompi_hc118613_108
  - filelock=3.13.1=py311h06a4308_0
  - font-ttf-dejavu-sans-mono=2.37=hd3eb1b0_0
  - font-ttf-inconsolata=2.001=hcb22688_0
  - font-ttf-source-code-pro=2.030=hd3eb1b0_0
  - font-ttf-ubuntu=0.83=h8b1ccd4_0
  - fontconfig=2.14.2=h14ed4e7_0
  - fonts-anaconda=1=h8fa9717_0
  - fonts-conda-ecosystem=1=hd3eb1b0_0
  - fonttools=4.25.0=pyhd3eb1b0_0
  - freeglut=3.2.2=h9c3ff4c_1
  - freetype=2.12.1=h4a9f257_0
  - fsspec=2023.10.0=py311h06a4308_0
  - gettext=0.22.5=h59595ed_2
  - gettext-tools=0.22.5=h59595ed_2
  - giflib=5.2.1=h5eee18b_3
  - gitdb=4.0.7=pyhd3eb1b0_0
  - gitpython=3.1.37=py311h06a4308_0
  - glib=2.78.4=h6a678d5_0
  - glib-tools=2.78.4=h6a678d5_0
  - gmp=6.2.1=h295c915_3
  - gmpy2=2.1.2=py311hc9b5ff0_0
  - gnutls=3.7.9=hb077bed_0
  - graphite2=1.3.14=h295c915_1
  - gst-plugins-base=1.22.0=h4243ec0_2
  - gstreamer=1.22.0=h25f0c4b_2
  - gstreamer-orc=0.4.38=hd590300_0
  - harfbuzz=6.0.0=h8e241bc_0
  - hdf5=1.14.0=nompi_hb72d44e_103
  - icu=70.1=h27087fc_0
  - idna=3.4=py311h06a4308_0
  - imagecodecs=2023.1.23=py311h8105a5c_0
  - imageio=2.33.1=py311h06a4308_0
  - intel-openmp=2023.1.0=hdb19cb5_46306
  - jack=1.9.22=h11f4161_0
  - jasper=2.0.33=h0ff4b12_1
  - jinja2=3.1.3=py311h06a4308_0
  - joblib=1.2.0=py311h06a4308_0
  - jpeg=9e=h5eee18b_1
  - jxrlib=1.1=h7b6447c_2
  - kiwisolver=1.4.4=py311h6a678d5_0
  - krb5=1.20.1=h143b758_1
  - lame=3.100=h7b6447c_0
  - lazy_loader=0.3=py311h06a4308_0
  - lcms2=2.12=h3be6417_0
  - ld_impl_linux-64=2.38=h1181459_1
  - lerc=3.0=h295c915_0
  - libaec=1.1.3=h59595ed_0
  - libasprintf=0.22.5=h661eb56_2
  - libasprintf-devel=0.22.5=h661eb56_2
  - libavif=0.11.1=h8182462_2
  - libblas=3.9.0=1_h86c2bf4_netlib
  - libbrotlicommon=1.0.9=h5eee18b_7
  - libbrotlidec=1.0.9=h5eee18b_7
  - libbrotlienc=1.0.9=h5eee18b_7
  - libcap=2.66=ha37c62d_0
  - libcblas=3.9.0=5_h92ddd45_netlib
  - libclang=15.0.7=default_h127d8a8_5
  - libclang13=15.0.7=default_h5d6823c_5
  - libcublas=12.1.0.26=0
  - libcufft=11.0.2.4=0
  - libcufile=1.9.1.3=0
  - libcups=2.3.3=h36d4200_3
  - libcurand=10.3.5.147=0
  - libcurl=8.5.0=h251f7ec_0
  - libcusolver=11.4.4.55=0
  - libcusparse=12.0.2.55=0
  - libdb=6.2.32=h6a678d5_1
  - libdeflate=1.17=h5eee18b_1
  - libdrm=2.4.120=hd590300_0
  - libedit=3.1.20230828=h5eee18b_0
  - libev=4.33=h7f8727e_1
  - libevent=2.1.10=h28343ad_4
  - libffi=3.4.4=h6a678d5_0
  - libflac=1.4.3=h59595ed_0
  - libgcc-ng=13.2.0=h807b86a_5
  - libgcrypt=1.10.3=hd590300_0
  - libgettextpo=0.22.5=h59595ed_2
  - libgettextpo-devel=0.22.5=h59595ed_2
  - libgfortran-ng=13.2.0=h69a702a_5
  - libgfortran5=13.2.0=ha4646dd_5
  - libglib=2.78.4=hdc74915_0
  - libglu=9.0.0=hf484d3e_1
  - libgomp=13.2.0=h807b86a_5
  - libgpg-error=1.48=h71f35ed_0
  - libiconv=1.17=hd590300_2
  - libidn2=2.3.4=h5eee18b_0
  - libjpeg-turbo=2.0.0=h9bf148f_0
  - liblapack=3.9.0=5_h92ddd45_netlib
  - liblapacke=3.9.0=5_h92ddd45_netlib
  - libllvm15=15.0.7=hadd5161_1
  - libnghttp2=1.57.0=h2d74bed_0
  - libnpp=12.0.2.50=0
  - libnsl=2.0.0=h5eee18b_0
  - libnvjitlink=12.1.105=0
  - libnvjpeg=12.1.1.14=0
  - libogg=1.3.5=h27cfd23_1
  - libopencv=4.7.0=py311h7a0761e_1
  - libopus=1.3.1=h7b6447c_0
  - libpciaccess=0.18=hd590300_0
  - libpng=1.6.39=h5eee18b_0
  - libpq=15.2=hb675445_0
  - libprotobuf=3.21.12=hfc55251_2
  - libsndfile=1.2.2=hc60ed4a_1
  - libsqlite=3.45.2=h2797004_0
  - libssh2=1.10.0=hdbd6064_2
  - libstdcxx-ng=13.2.0=h7e041cc_5
  - libsystemd0=252=h2a991cd_0
  - libtasn1=4.19.0=h5eee18b_0
  - libtiff=4.5.1=h6a678d5_0
  - libtool=2.4.7=h27087fc_0
  - libudev1=253=h0b41bf4_0
  - libunistring=0.9.10=h27cfd23_0
  - libuuid=2.38.1=h0b41bf4_0
  - libva=2.18.0=h0b41bf4_0
  - libvorbis=1.3.7=h7b6447c_0
  - libvpx=1.11.0=h295c915_0
  - libwebp-base=1.3.2=h5eee18b_0
  - libxcb=1.13=h1bed415_1
  - libxkbcommon=1.5.0=h79f4944_1
  - libxml2=2.10.3=hca2bb57_4
  - libzlib=1.2.13=hd590300_5
  - libzopfli=1.0.3=he6710b0_0
  - lightning-utilities=0.9.0=py311h06a4308_0
  - llvm-openmp=14.0.6=h9e868ea_0
  - lz4-c=1.9.4=h6a678d5_0
  - markupsafe=2.1.3=py311h5eee18b_0
  - matplotlib=3.8.4=py311h06a4308_0
  - matplotlib-base=3.8.4=py311ha02d727_0
  - mkl=2023.1.0=h213fc3f_46344
  - mkl-service=2.4.0=py311h5eee18b_1
  - mkl_fft=1.3.8=py311h5eee18b_0
  - mkl_random=1.2.4=py311hdb19cb5_0
  - mpc=1.1.0=h10f8cd9_1
  - mpfr=4.0.2=hb69a4c5_1
  - mpg123=1.32.6=h59595ed_0
  - mpmath=1.3.0=py311h06a4308_0
  - munkres=1.1.4=py_0
  - mysql-common=8.0.32=ha901b37_0
  - mysql-libs=8.0.32=hd7da12d_0
  - ncurses=6.4=h6a678d5_0
  - nettle=3.9.1=h7ab15ed_0
  - networkx=3.1=py311h06a4308_0
  - nspr=4.35=h6a678d5_0
  - nss=3.89.1=h6a678d5_0
  - numexpr=2.8.7=py311h65dcdc2_0
  - numpy=1.24.3=py311h08b1b3b_1
  - numpy-base=1.24.3=py311hf175353_1
  - openh264=2.3.1=hcb278e6_2
  - openjpeg=2.4.0=h3ad879b_0
  - openssl=3.0.13=h7f8727e_0
  - p11-kit=0.24.1=hc5aa10d_0
  - packaging=23.2=py311h06a4308_0
  - pandas=2.2.1=py311ha02d727_0
  - pathtools=0.1.2=pyhd3eb1b0_1
  - pcre2=10.42=hebb0a14_0
  - pillow=10.2.0=py311h5eee18b_0
  - pip=23.3.1=py311h06a4308_0
  - pixman=0.40.0=h7f8727e_1
  - ply=3.11=py311h06a4308_0
  - protobuf=4.21.12=py311hcafe171_0
  - psutil=5.9.0=py311h5eee18b_0
  - pulseaudio=16.1=ha8d29e2_1
  - py-opencv=4.7.0=py311h781c19f_1
  - pyparsing=3.0.9=py311h06a4308_0
  - pyqt=5.15.10=py311h6a678d5_0
  - pyqt5-sip=12.13.0=py311h5eee18b_0
  - pysocks=1.7.1=py311h06a4308_0
  - python=3.11.0=he550d4f_1_cpython
  - python-dateutil=2.8.2=pyhd3eb1b0_0
  - python-tzdata=2023.3=pyhd3eb1b0_0
  - python_abi=3.11=4_cp311
  - pytorch=2.2.2=py3.11_cuda12.1_cudnn8.9.2_0
  - pytorch-cuda=12.1=ha16c6d3_5
  - pytorch-lightning=2.0.3=py311h06a4308_0
```

```
- pytorch-mutex=1.0=cuda
- pytz=2023.3.post1=py311h06a4308_0
- pyyaml=6.0.1=py311h5eee18b_0
- qt-main=5.15.8=h5d23da1_6
- readline=8.2=h5eee18b_0
- requests=2.31.0=py311h06a4308_1
- scikit-image=0.22.0=py311ha02d727_0
- scikit-learn=1.3.0=py311ha02d727_1
- scipy=1.12.0=py311h08b1b3b_0
- sentry-sdk=1.9.0=py311h06a4308_0
- setproctitle=1.2.2=py311h5eee18b_0
- setuptools=68.2.2=py311h06a4308_0
- sip=6.7.12=py311h6a678d5_0
- six=1.16.0=pyhd3eb1b0_1
- smmap=4.0.0=pyhd3eb1b0_0
- snappy=1.1.10=h6a678d5_1
- sqlite=3.41.2=h5eee18b_0
- svt-av1=1.4.1=hcb278e6_0
- sympy=1.12=py311h06a4308_0
- tbb=2021.8.0=hdb19cb5_0
- tensorboardx=2.6.2.2=pyhd8ed1ab_0
- threadpoolctl=2.2.0=pyh0d69192_0
- tifffile=2023.4.12=py311h06a4308_0
- tk=8.6.12=h1ccaba5_0
- torchaudio=2.2.2=py311_cu121
- torchmetrics=1.1.2=py311h06a4308_0
- torchtriton=2.2.0=py311
- torchvision=0.17.2=py311_cu121
- tornado=6.3.3=py311h5eee18b_0
- tqdm=4.65.0=py311h92b7b1e_0
- typing-extensions=4.9.0=py311h06a4308_1
- typing_extensions=4.9.0=py311h06a4308_1
- tzdata=2024a=h04d1e81_0
- urllib3=2.1.0=py311h06a4308_1
- wandb=0.16.5=pyhd8ed1ab_0
- wheel=0.41.2=py311h06a4308_0
- x264=1!164.3095=h166bdaf_2
- x265=3.5=h924138e_3
- xcb-util=0.4.0=h516909a_0
- xcb-util-image=0.4.0=h166bdaf_0
- xcb-util-keysyms=0.4.0=h516909a_0
- xcb-util-renderutil=0.3.9=h166bdaf_0
- xcb-util-wm=0.4.1=h516909a_0
- xkeyboard-config=2.38=h0b41bf4_0
```

```
  - xorg-fixesproto=5.0=h7f98852_1002
  - xorg-inputproto=2.3.2=h7f98852_1002
  - xorg-kbproto=1.0.7=h7f98852_1002
  - xorg-libice=1.1.1=hd590300_0
  - xorg-libsm=1.2.4=h7391055_0
  - xorg-libx11=1.8.4=h0b41bf4_0
  - xorg-libxau=1.0.11=hd590300_0
  - xorg-libxext=1.3.4=h0b41bf4_2
  - xorg-libxfixes=5.0.3=h7f98852_1004
  - xorg-libxi=1.7.10=h7f98852_0
  - xorg-libxrender=0.9.10=h7f98852_1003
  - xorg-renderproto=0.11.1=h7f98852_1002
  - xorg-xextproto=7.3.0=h0b41bf4_1003
  - xorg-xproto=7.0.31=h27cfd23_1007
  - xz=5.4.6=h5eee18b_0
  - yaml=0.2.5=h7b6447c_0
  - zfp=1.0.0=h6a678d5_0
  - zlib=1.2.13=hd590300_5
  - zstd=1.5.5=hc292b87_0
prefix: path
```

Listing 5: Anaconda3 environment

A.2 YOLOv5 hyperparameters

```
# YOLOv5  by Ultralytics, AGPL-3.0 license
# Hyperparameters for high-augmentation COCO training from scratch
# python train.py --batch 32 --cfg yolov5m6.yaml --weights '' --data coco.yaml --img
↪  1280 --epochs 300
# See tutorials for hyperparameter evolution
↪  https://github.com/ultralytics/yolov5#tutorials

lr0: 0.0001 # initial learning rate (SGD=1E-2, Adam=1E-3)
lrf: 0.1 # final OneCycleLR learning rate (lr0 * lrf)
momentum: 0.937 # SGD momentum/Adam beta1
weight_decay: 0.0005 # optimizer weight decay 5e-4
warmup_epochs: 3.0 # warmup epochs (fractions ok)
warmup_momentum: 0.8 # warmup initial momentum
warmup_bias_lr: 0.1 # warmup initial bias lr
box: 0.01 # box loss gain
cls: 0.3 # cls loss gain
cls_pw: 1.0 # cls BCELoss positive_weight
obj: 0.7 # obj loss gain (scale with pixels)
obj_pw: 1.0 # obj BCELoss positive_weight
iou_t: 0.20 # IoU training threshold
anchor_t: 4.0 # anchor-multiple threshold
anchors: 9  # anchors per output layer (0 to ignore)
fl_gamma: 0.0 # focal loss gamma (efficientDet default gamma=1.5)
hsv_h: 0.015 # image HSV-Hue augmentation (fraction)
hsv_s: 0.7 # image HSV-Saturation augmentation (fraction)
hsv_v: 0.4 # image HSV-Value augmentation (fraction)
degrees: 0.0 # image rotation (+/- deg)
translate: 0.1 # image translation (+/- fraction)
scale: 0.9 # image scale (+/- gain)
shear: 0.0 # image shear (+/- deg)
perspective: 0.001 # image perspective (+/- fraction), range 0-0.001
flipud: 0 # image flip up-down (probability)
fliplr: 0.5 # image flip left-right (probability)
mosaic: 1.0 # image mosaic (probability)
mixup: 0.2 # image mixup (probability)
copy_paste: 0.1 # segment copy-paste (probability)
```

Listing 6: Anaconda3 environment

A.3 Result visualizations

A.3.1 Baselines

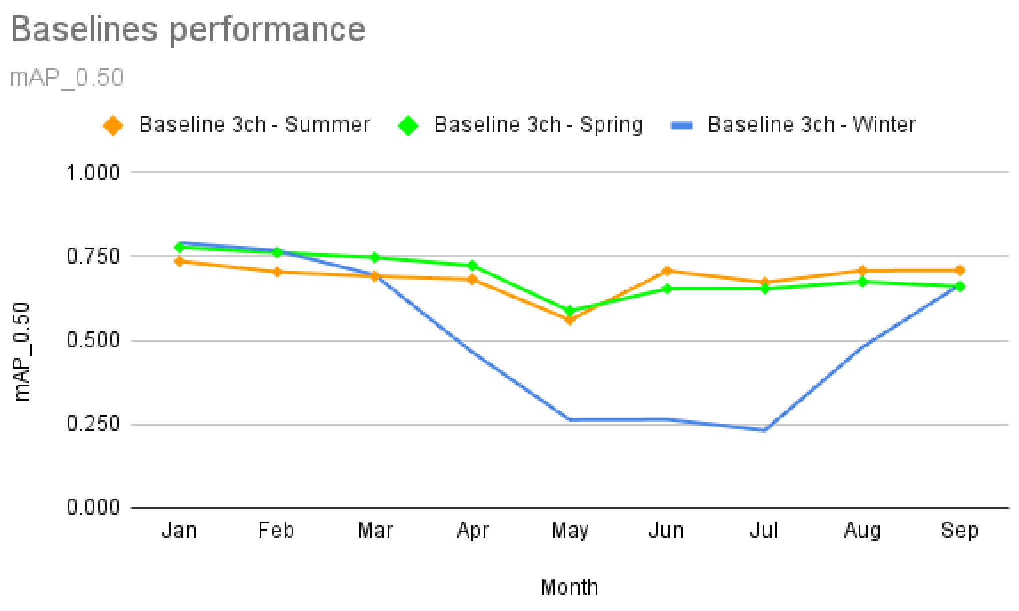


Figure A.1: $mAP_{0.50}$ distribution of baselines as a function of time of year.

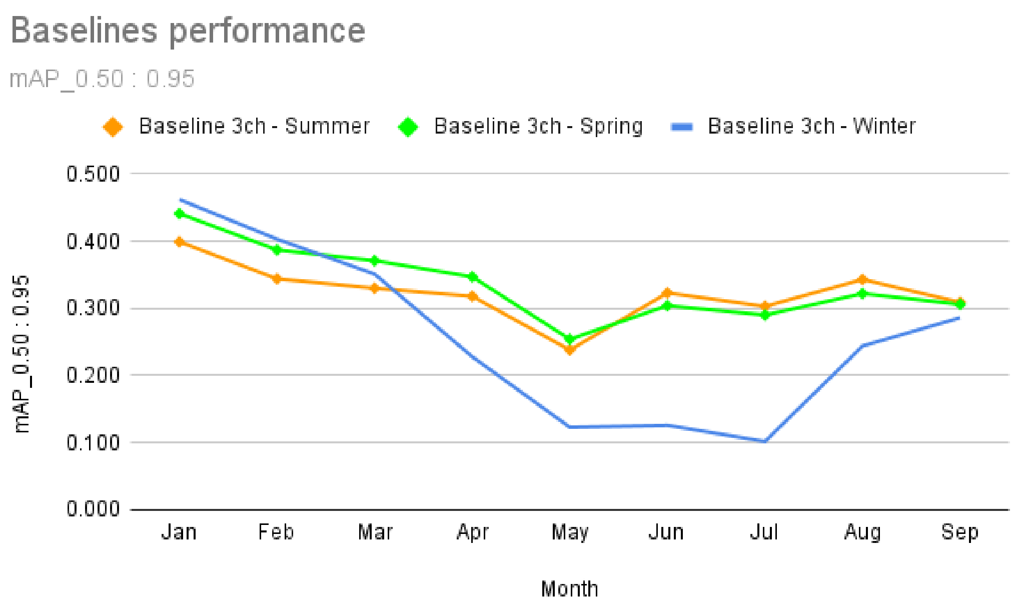


Figure A.2: mAP$_{0.50:0.95}$ distribution of baselines as a function of time of year.

A.3.2 U²OF

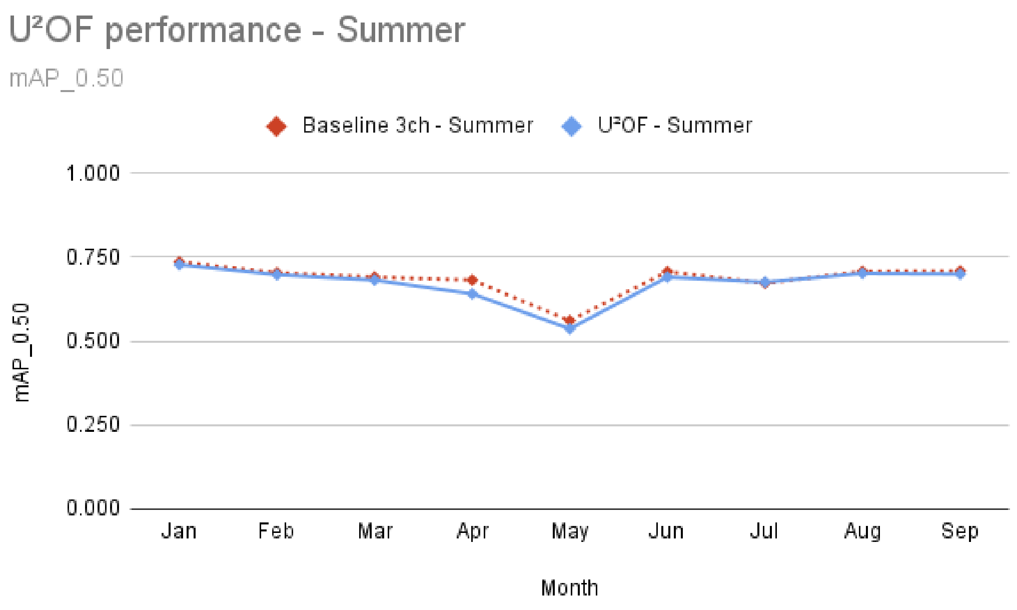


Figure A.3: TBD

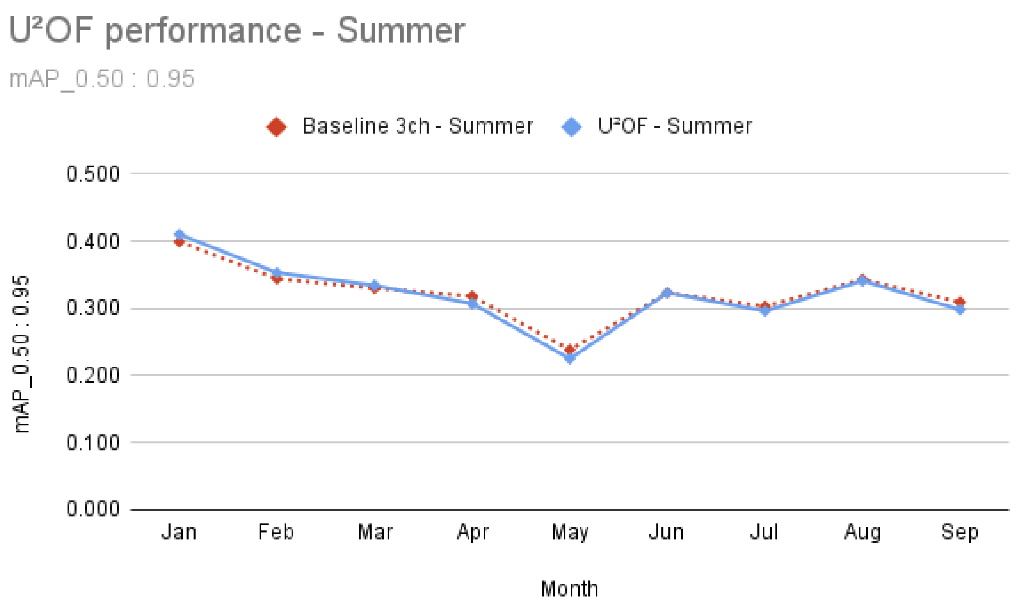


Figure A.4: TBD

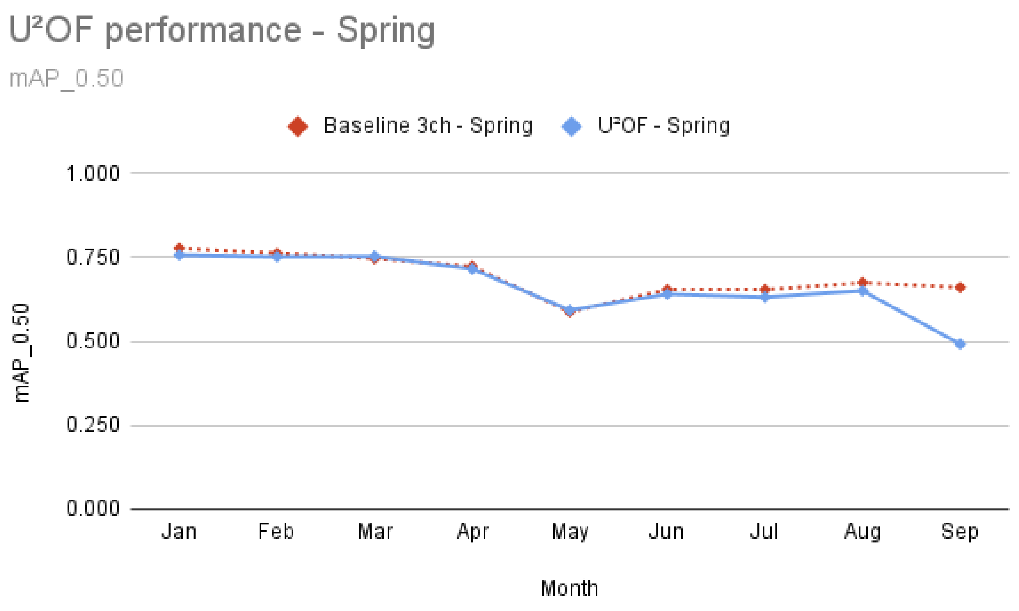


Figure A.5: TBD

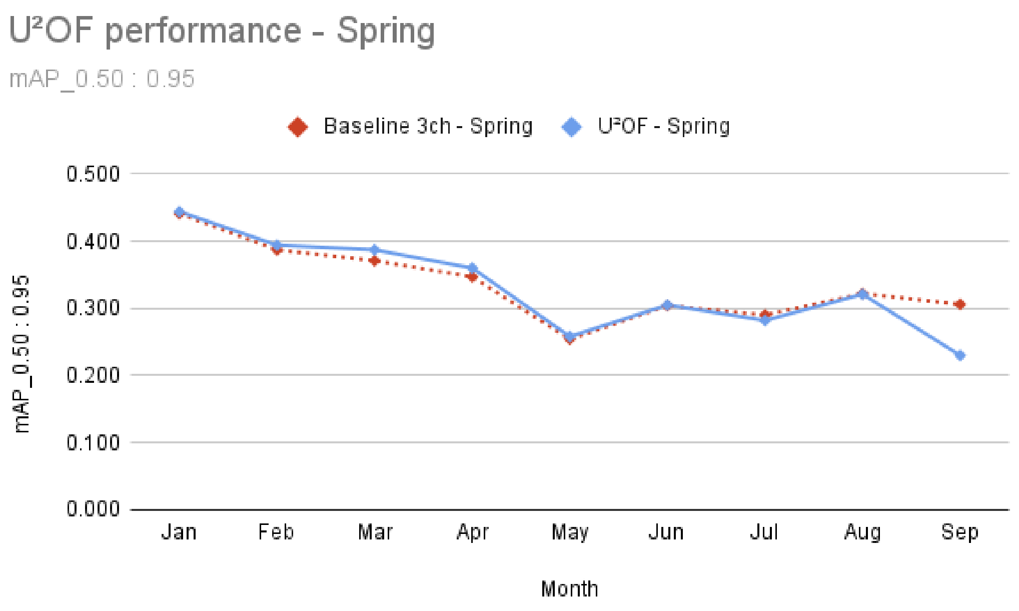


Figure A.6: TBD

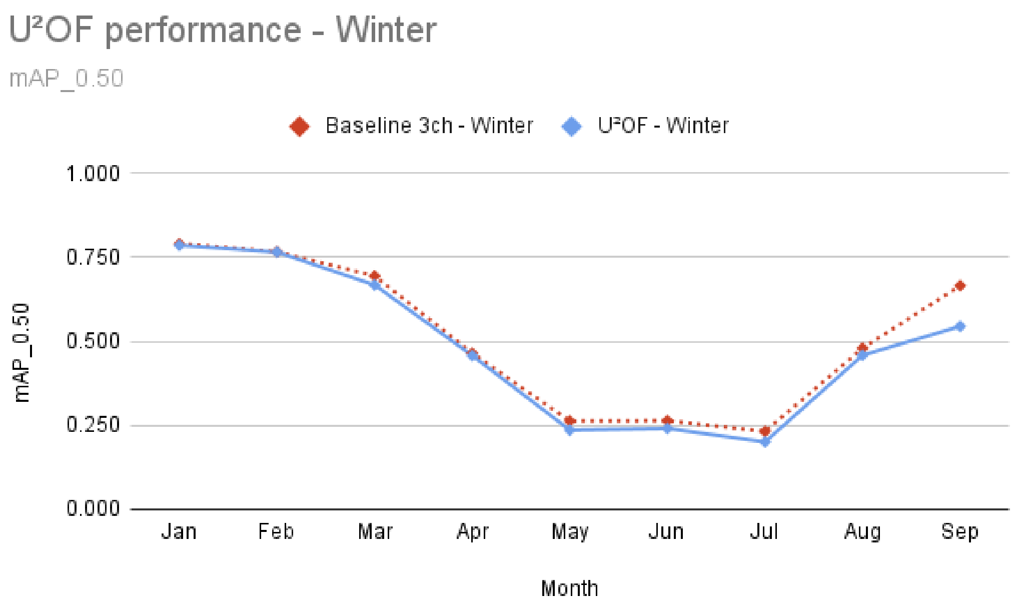


Figure A.7: TBD

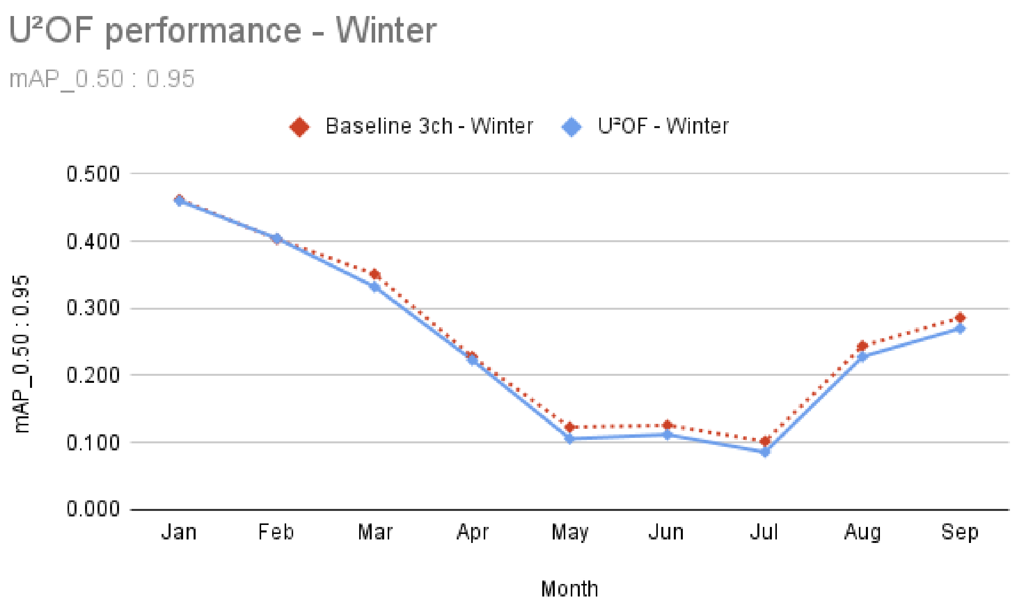


Figure A.8: TBD

A.3.3 Channels

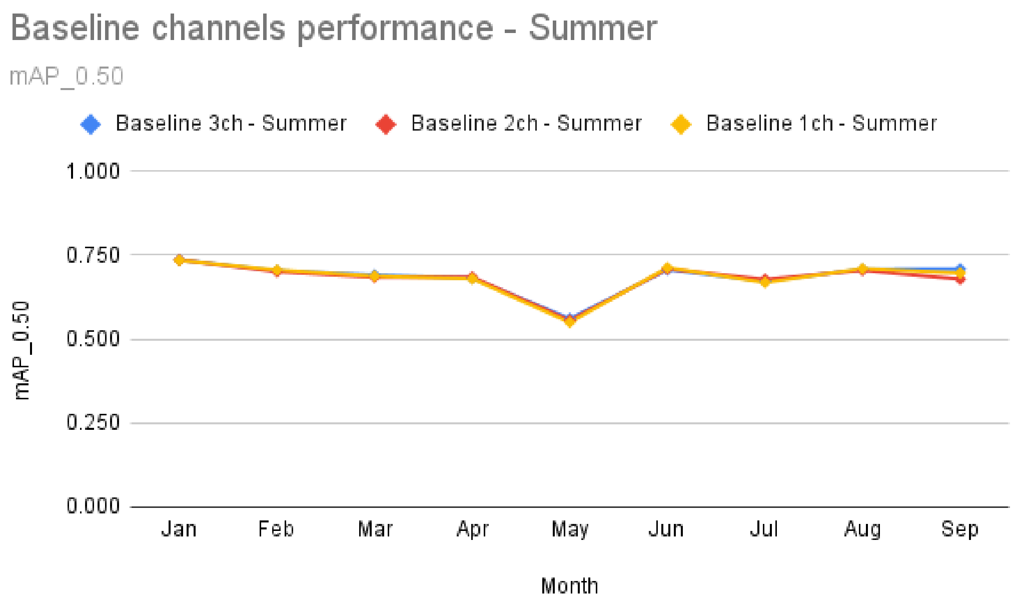


Figure A.9: TBD

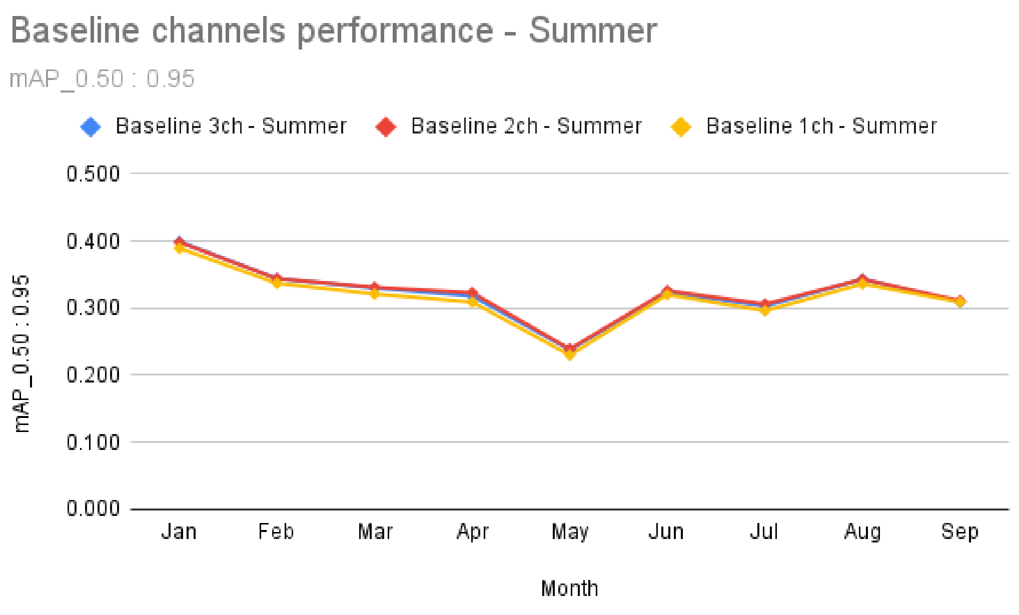


Figure A.10: TBD

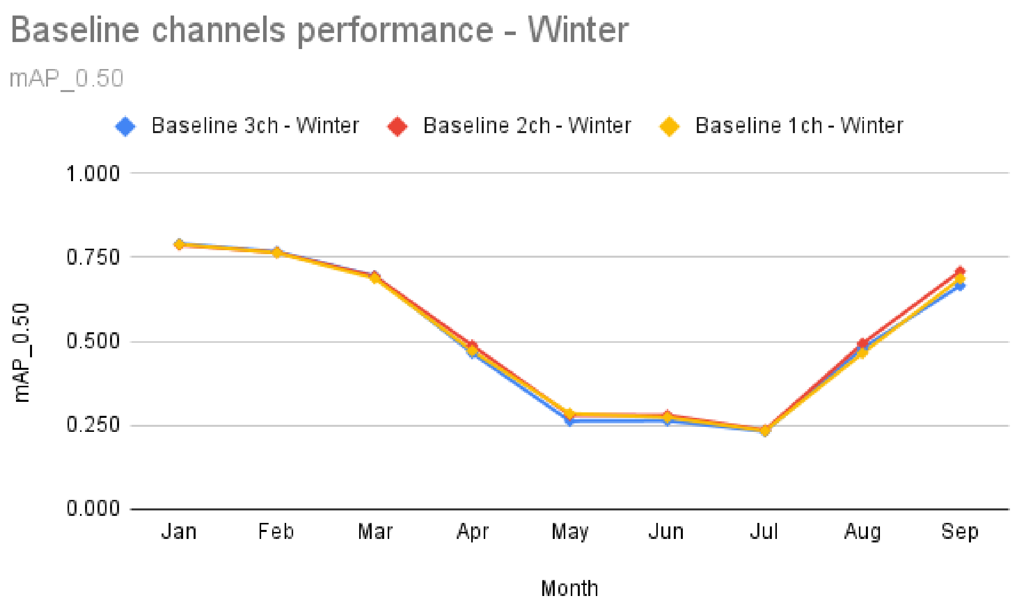


Figure A.11: TBD

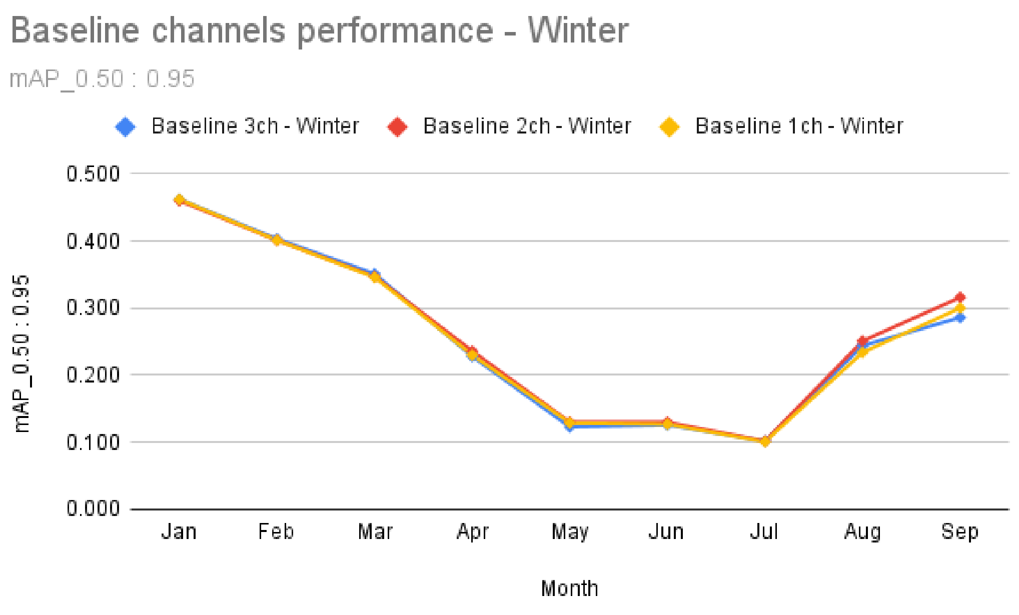


Figure A.12: TBD

A.3.4 U^{2}Net

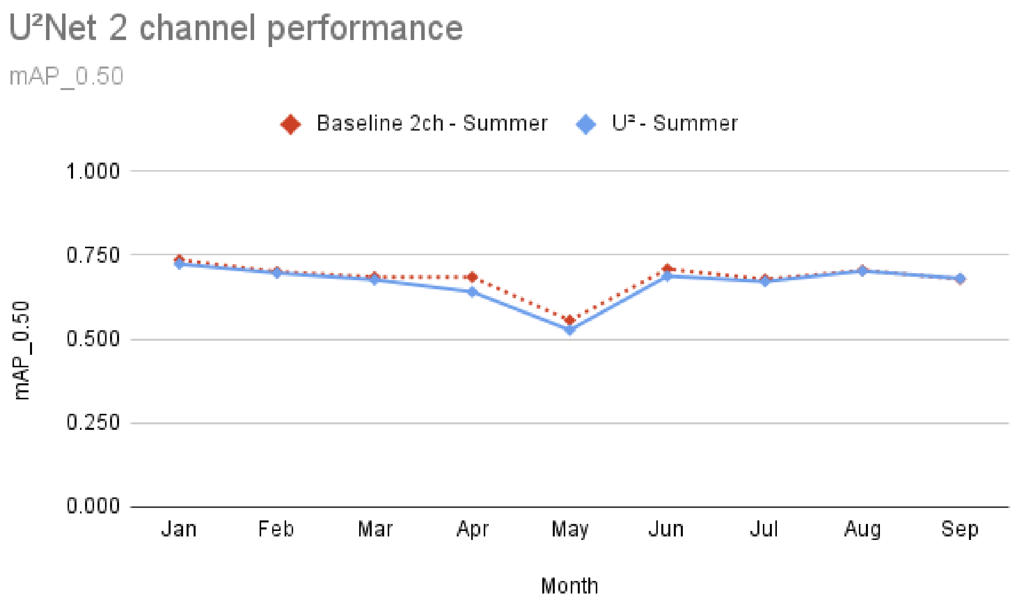


Figure A.13: TBD

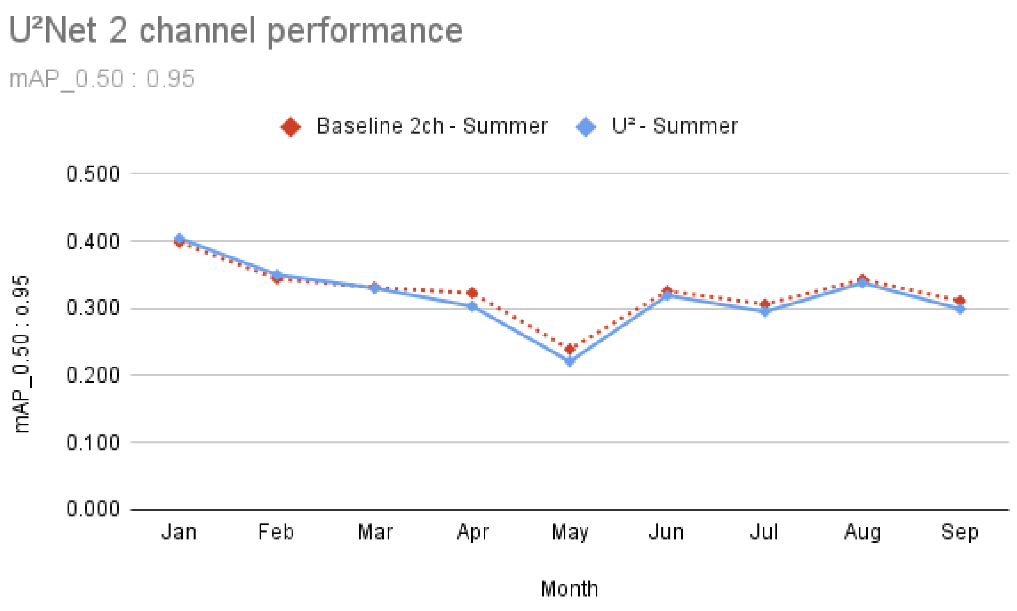


Figure A.14: TBD

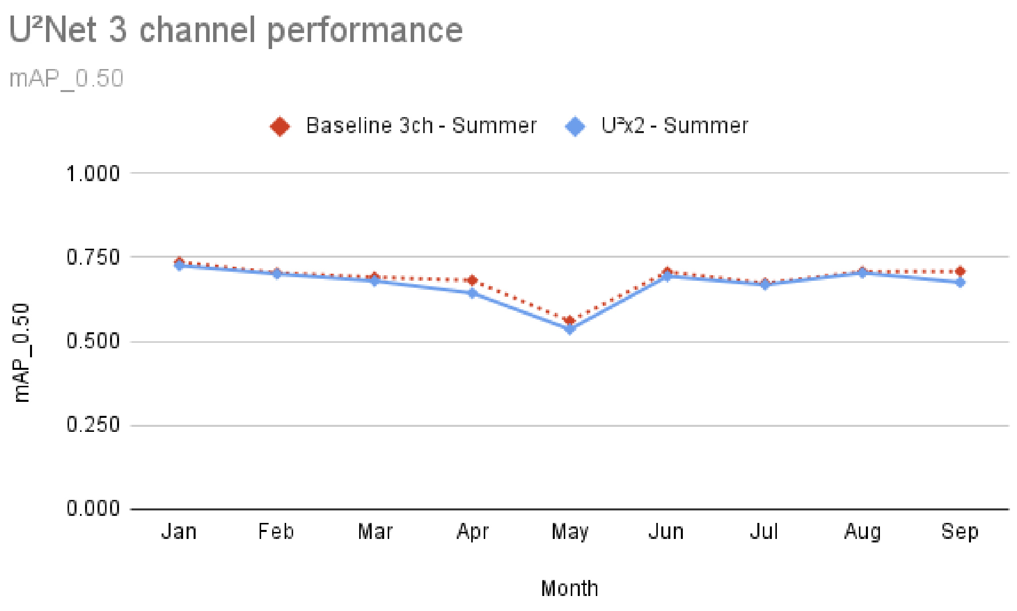


Figure A.15: TBD

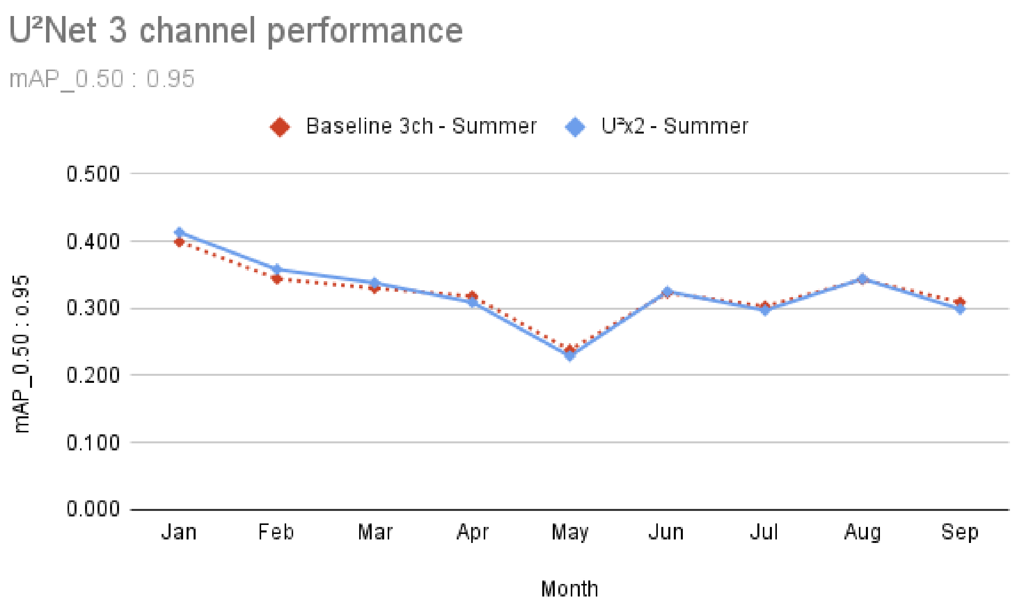


Figure A.16: TBD

A.3.5 Optical Flow

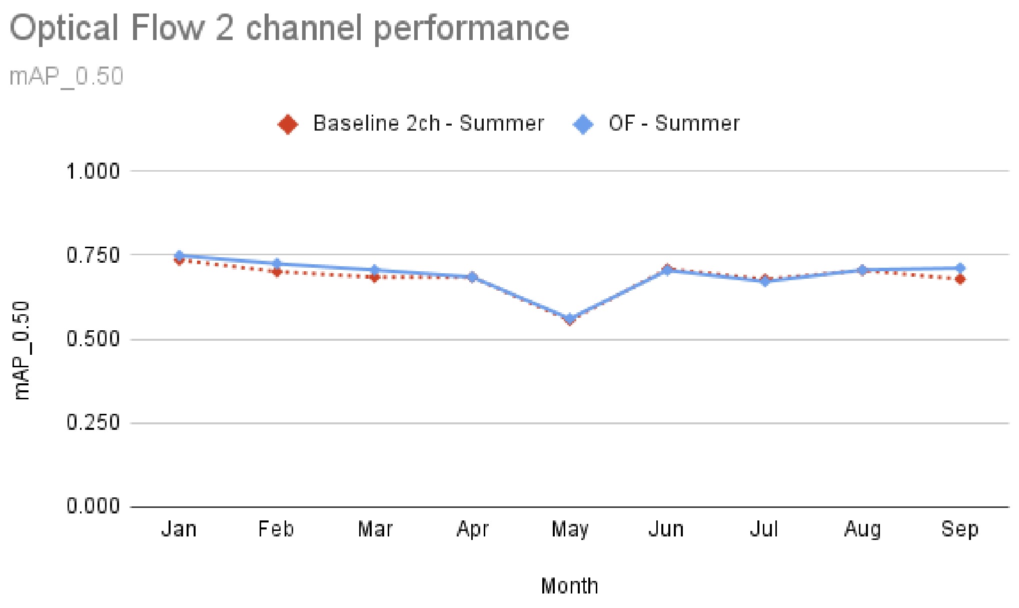


Figure A.17: TBD

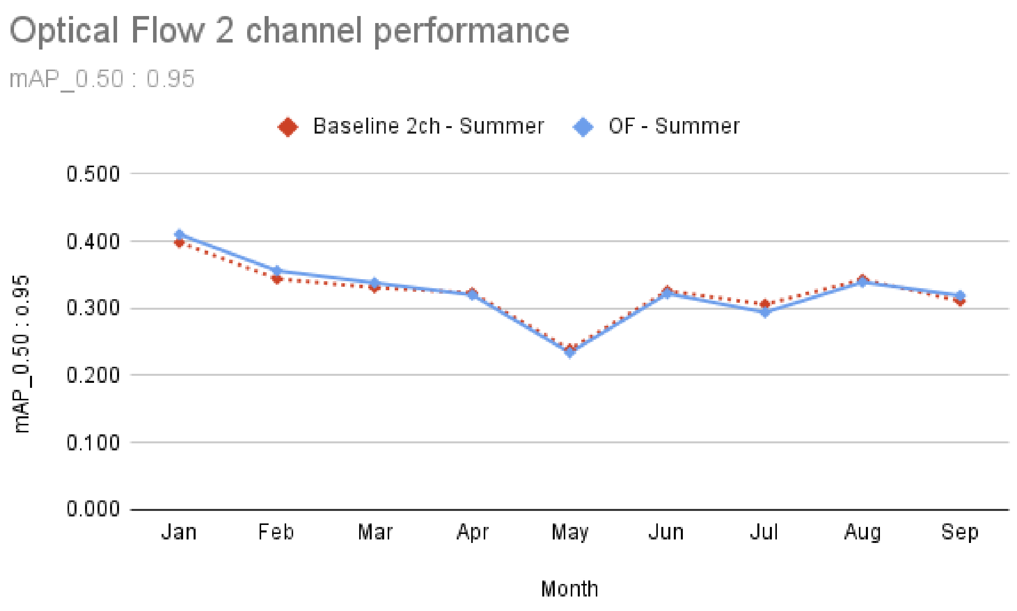


Figure A.18: TBD

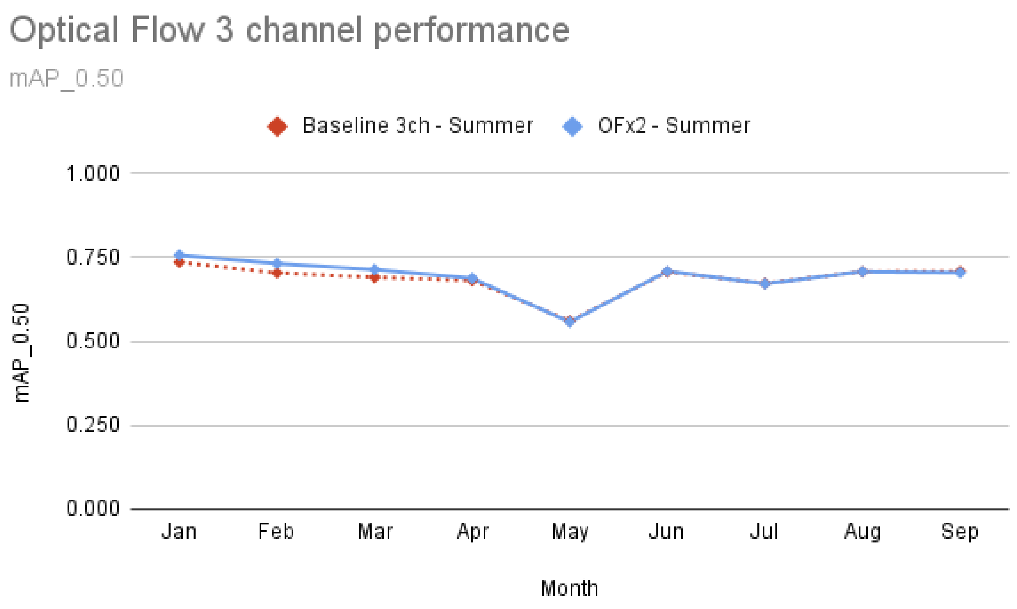


Figure A.19: TBD

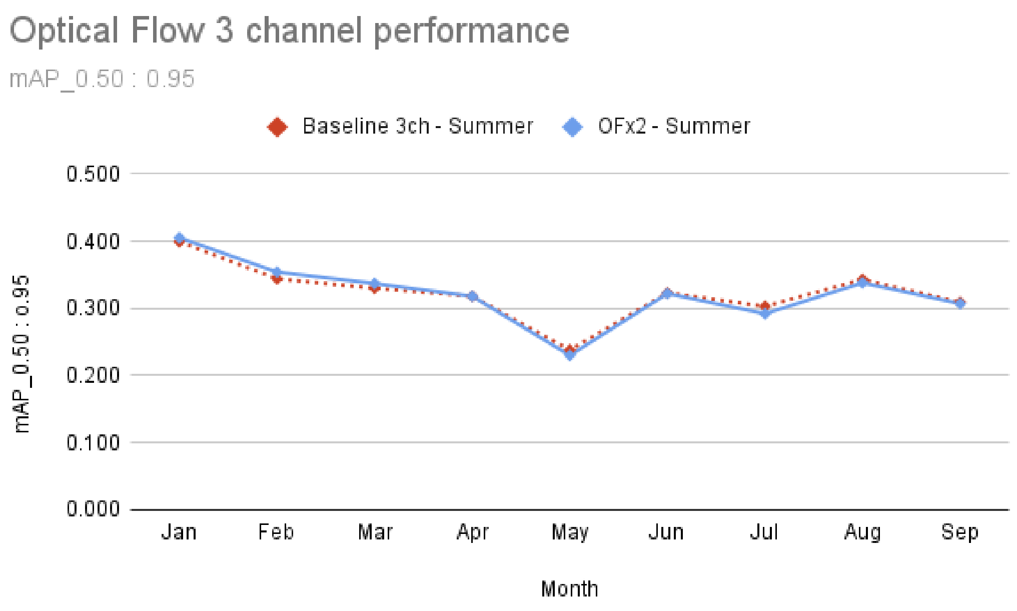


Figure A.20: TBD

A.4 May clip

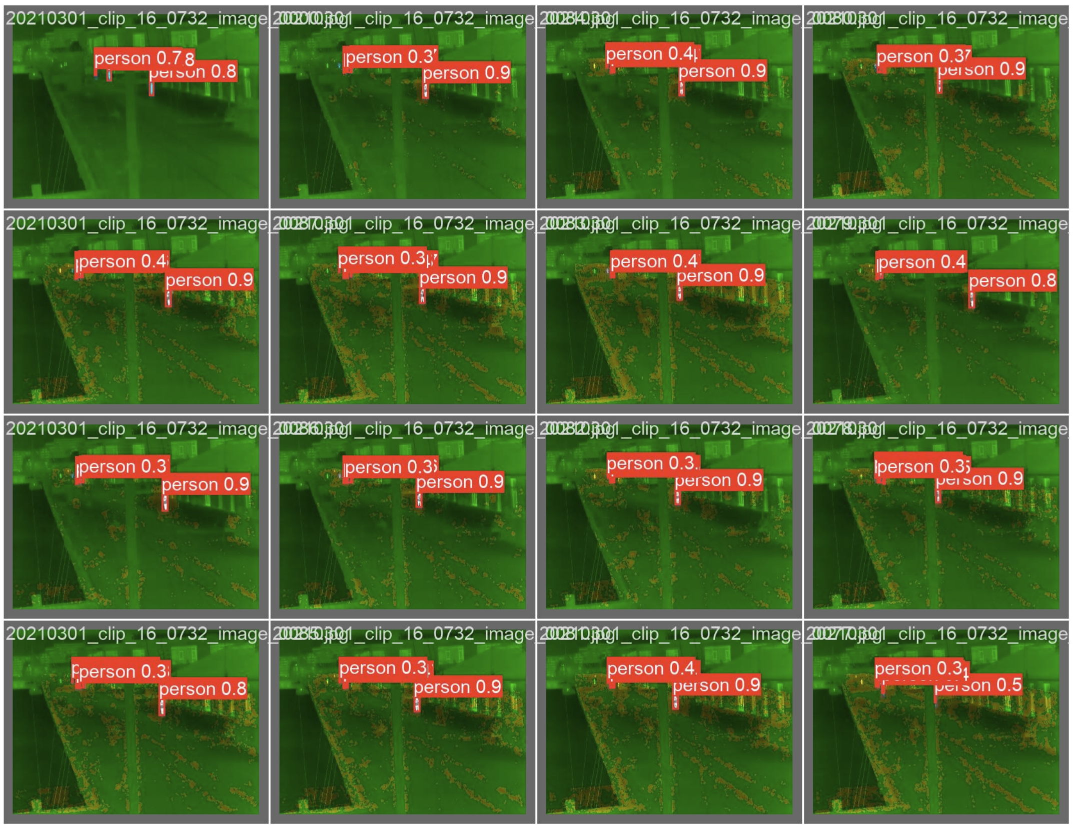


Figure A.21: Enter Caption

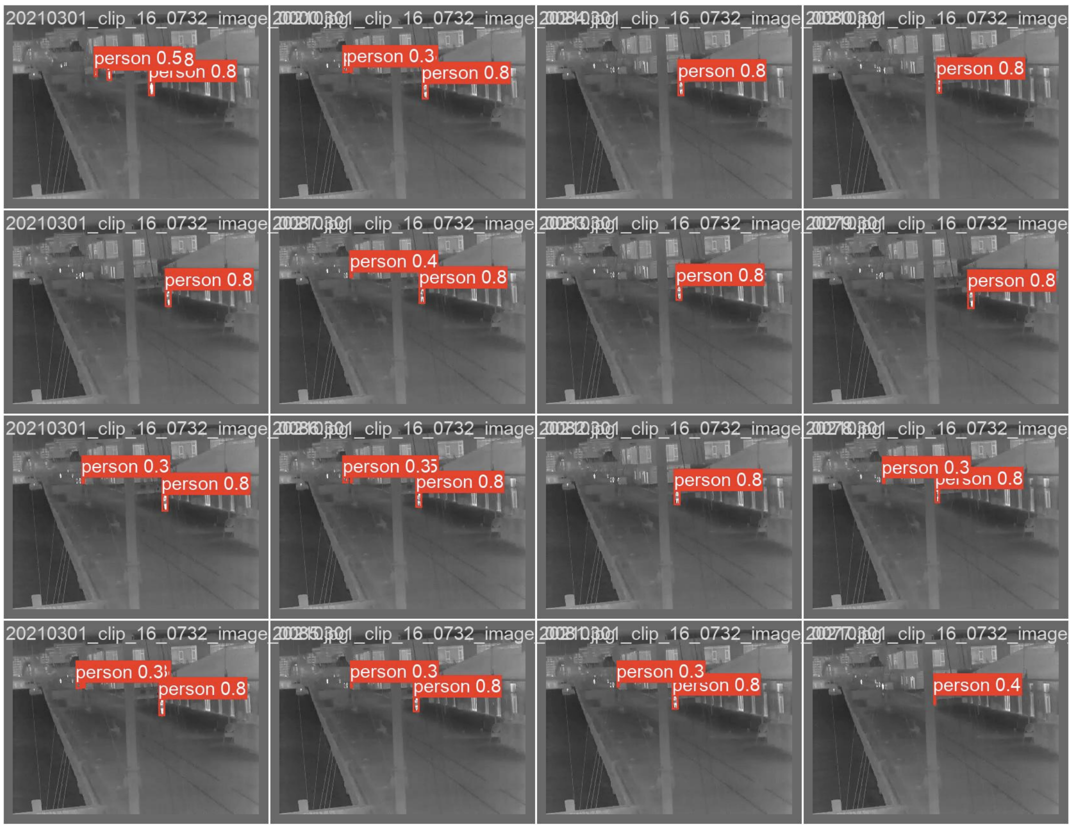


Figure A.22: Enter Caption

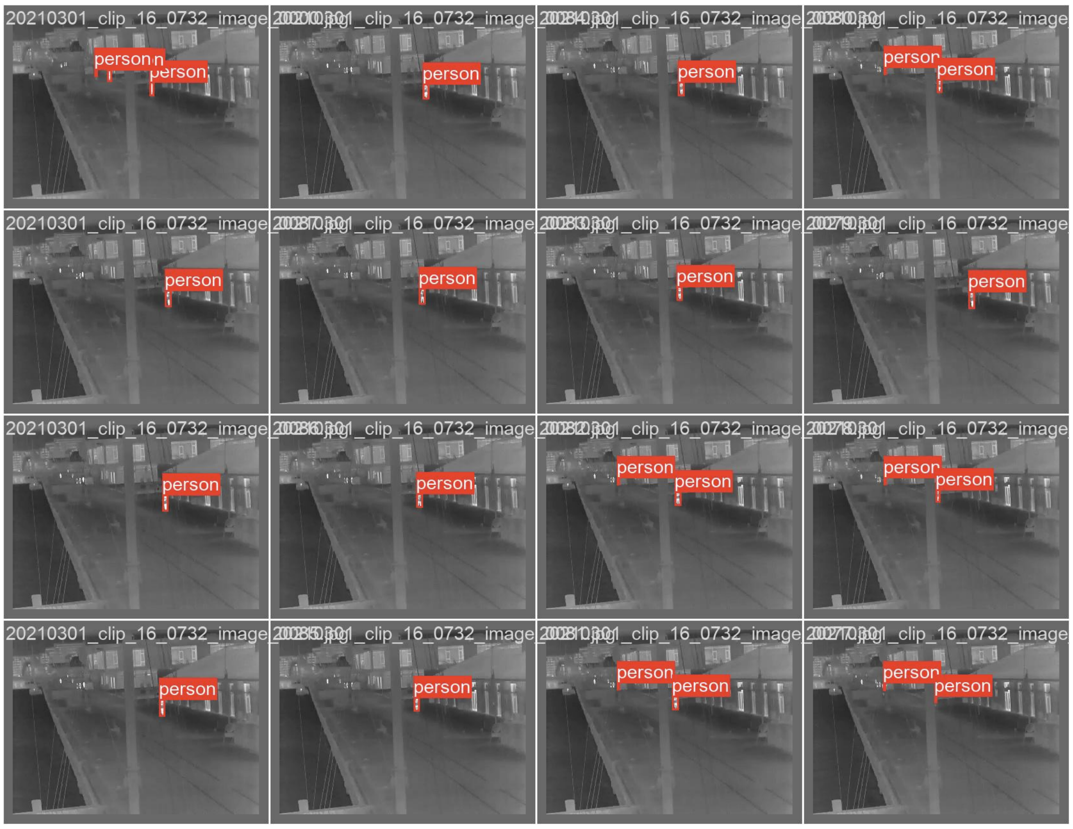


Figure A.23: Enter Caption