



BRNO UNIVERSITY OF TECHNOLOGY

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

FACULTY OF INFORMATION TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ

DEPARTMENT OF INTELLIGENT SYSTEMS

ÚSTAV INTELIGENTNÍCH SYSTÉMŮ

DETECTION AND RECOGNITION OF DRONE MOVEMENT IN VIDEO

DETEKCE A ROZPOZNÁNÍ POHYBU DRONU VE VIDEU

BACHELOR'S THESIS

BAKALÁŘSKÁ PRÁCE

AUTHOR

AUTOR PRÁCE

SIMON LAPŠANSKÝ

SUPERVISOR

VEDOUCÍ PRÁCE

Prof. Ing., Dipl.-Ing. MARTIN DRAHANSKÝ, Ph.D.

BRNO 2022

Bachelor's Thesis Specification



Student: **Lapšanský Simon**
Programme: Information Technology
Title: **Detection and Recognition of Drone Movement in Video**
Category: Image Processing

Assignment:

1. Get acquainted with video object detection and recognition literature.
2. Create a database of videos depicting drones of different sizes, shapes, and colors against various backgrounds.
3. Propose an algorithm for the detection and recognition of drone motion in captured video sequences from the 2nd point.
4. Implement the proposed algorithm from the 3rd point.
5. Evaluate the performance of the algorithm from the 4th point applied to the database from the 2nd point, summarize the achieved results.

Recommended literature:

- SEIDALIYEVA, Ulzhalgas, et al. Real-Time and accurate drone detection in a video with a static background. *Sensors*, 2020, 20.14: 3856.
- ALSOLIMAN, Anas, et al. COTS Drone Detection using Video Streaming Characteristics. In: *International Conference on Distributed Computing and Networking 2021*. 2021. p. 166-175.

Requirements for the first semester:

- Items 1 to 3 of the assignment.

Detailed formal requirements can be found at <https://www.fit.vut.cz/study/theses/>

Supervisor: **Drahanský Martin, prof. Ing., Dipl.-Ing., Ph.D.**
Head of Department: Hanáček Petr, doc. Dr. Ing.
Beginning of work: November 1, 2021
Submission deadline: May 11, 2022
Approval date: May 11, 2022

Abstract

With the increase in drone availability in recent years, the risk of using drones as a tool for attacks has also increased. Based on these risks, this paper proposes a method for their real-time detection, followed by classification. The proposed approach utilizes the background subtraction method for object detection while using deep learning to classify the detected object. The MOG2 utilizes the Gaussian mixture model method to provide background subtraction, while the YOLOv5 object detection model uses convolutional neural networks for classification. The approach implementation produces a way for drone detection and classification while utilizing the processor, reaching results sufficient for real-time drone detection and classification. The method evaluating 1080p recording using the Intel i5-7600K averaged 16 frames per second while detecting a single object within the frame.

Abstrakt

S nárastom dostupnosti dronov, narástlo aj riziko ich využívania na nelegálne aktivity. Na základe týchto rizík bola navrhnutá metóda na ich detekciu a následnú klasifikáciu aplikovateľnú v reálnom čase. Navrhovaný prístup využíva metódu odčítania pozadia, slúžiacu na detekciu objektov, zatiaľ čo klasifikácia je dosiahnutá pomocou hlbokého učenia. MOG2 využíva metódu zmiešaného Gaussovho modelu, ktorý slúži na odčítanie pozadia, za účelom detekcie objektov. YOLOv5 model pracujúci s neurónovými sieťami je využitý na následnú klasifikáciu detegovaných objektov. Implementácia vytvára spôsob detekcie a klasifikácie dronov s využitím procesora dosahujúca výsledky postačujúce na aplikovanie detekcie a klasifikácie dronov v reálnom čase. Metóda vyhodnocujúca záznam v rozlíšení 1080p, využívajúca procesor Intel i5-7600K dosahovala v priemere 16 snímiek za sekundu, počas detekcie jedného objektu v snímke.

Keywords

Drone detection, Drone classification, MOG2 method, Computer vision, Deep learning, YOLOv5 model

Klíčové slová

Detekcia dronov, Klasifikácia dronov, MOG2 metóda, Počítačové videnie, Hlboké učenie, YOLOv5 model

Reference

LAPŠANSKÝ, Simon. *Detection and Recognition of Drone Movement in Video*. Brno, 2022. Bachelor's thesis. Brno University of Technology, Faculty of Information Technology. Supervisor Prof. Ing., Dipl.-Ing. Martin Drahan-ský, Ph.D.

Rozšírený abstrakt

S nárastom dostupnosti dronov, narastá aj riziko ich využívania na nelegálne aktivity. Medzi najčastejšie prípady zneužitia dronov patrí sledovanie objektov či osôb, no drony môžu byť taktiež využité na teroristické či iné útoky. Táto práca poskytuje riešenie na detekciu a klasifikáciu dronov za účelom zabránenia prípadných útokov.

Na detekciu dronov z video záznamu bola použitá metóda odčítania pozadia založená na Gaussovom zmiešanom modeli, konkrétne MOG2. MOG2 pracuje s tromi farebnými kanálmi, ktoré sú reprezentované pomocou trojrozmerného Gaussovho zmiešaného modelu. Krivky obsiahnuté v tom modeli reprezentujú buď pozadie alebo prípadne zistené popredie. Pričom väčšie hodnoty kriviek reprezentujú pozadie, zatiaľ čo menšie hodnoty definujú popredie. Táto metóda je aplikovaná pri detekcii používajúcej statickú kameru.

Klasifikácia bola dosiahnutá pomocou hlbokého učenia. Klasifikácia využíva konvolučné neurónové siete na spracovanie vstupných obrázkov. V tomto prístupe bol využitý model YOLOv5s, ktorého architektúra zabezpečuje nadriadenú rýchlosť klasifikácie v porovnaní s ostatnými metódami. Model YOLOv5 bol trénovaný za pomoci datasetu zobrazujúceho drona typu DJI Mavic Air v čiernej farbe. Tento dataset pozostával z 4 videí pričom 3 z nich boli použité na vytvorenie datasetu na trénovanie a zvyšné video bolo použité na vyhodnotenie implementovaného algoritmu. Predtým ako boli tieto videá použité na trénovanie, musel byť vytvorený dataset anotovaných obrázkov. Tieto obrázky boli zachytené pomocou MOG2 algoritmu a následne anotované a ohraničené. Pomocou nástroja roboflow bol vytvorený dataset kompatibilný s modelom YOLOv5, určený na jeho trénovanie. Výsledný dataset tvorilo 7271 obrázkov dronov, vtákov a rôznych typov pozadií. Trénovanie modelu využívalo YOLOv5 skript určený na jeho trénovanie. Architektúra modelu a jeho konfigurácia bola popísaná v yamle súbore. Na trénovanie bolo použitých 130 epochov pričom obrázky boli zoskupené po skupinkách obsahujúcich 24 obrázkov. Celkové trénovanie bolo uskutočnené pomocou Google collab prostredia, ktoré sprístupnilo využitie grafickej karty Tesla K80, čo razantne zmenšilo čas tréovania na hodinu, 47 minút a 56 sekúnd.

Implementácia navrhovaného algoritmu bola dosiahnutá pomocou jazyka python o verzii 3.8.10, obsiahnutá v jednom skripte `main.py`, ktorý sa skladá z dvoch modulov. Prvý ma na starosti detekciu objektov pomocou MOG2 algoritmu dostupného v knižnici OpenCV, zatiaľ čo druhý modul využíva natrénovaný YOLOv5 model na následnú klasifikáciu výsledkov prvého modulu. Natrénovaný model bol načítaný do skriptu s využitím PyTorch knižnice, ktorá umožňuje jednoduché nasadenie vlastných modelov. Modul slúžiaci na detekciu objektov aplikuje morfologické operácie na výsledky MOG2 algoritmu, za účelom získania ucelených objektov. Tieto objekty sú následne orezané a ich rozlíšenie je upravené tak aby ho mohol YOLOv5 model následne vyhodnotiť. Po vyhodnotení modelom, je klasifikovaný objekt ohraničený štvorcem, nad ktorým je vypísaná klasifikovaná trieda spolu s hodnotou reprezentujúcou istotu, s akou bol objekt klasifikovaný.

Natrénovaný YOLOv5 model pri klasifikácii objektov, ktorých prekryv ohraničujúcich boxov tvoril 50% z celkovej oblasti tvorenej týmito boxami, dosahoval vysokú priemernú presnosť klasifikácie, ktorá sa blížila k bezchybnej presnosti. Priemerná presnosť pri prekryve o 95% bola podstatne nižšia a nekonzistentná. Hodnoty sa počas tréovania pohybovali v rozmedzí [25-35]%, bez toho aby sa hodnoty zlepšovali vzhľadom na čas strávený tréovaním.

Na zvýšenie efektívnosti detekcie dronov, boli aplikované morfologické operácie ako erózia a dilatácia. Erózia bola aplikovaná za účelom odstránenia šumu aby sa predišlo veľkému množstvu nesprávnych detekcií. Podľa záznamu a množstva šumu v snímke je

možné meniť hodnoty erózie v reálnom čase pomocou lišty nachádzajúcej sa pod grafickým výstupom. Dilatácia slúži na pospájanie neucelených výstupov z MOG2 algoritmu.

Následné vyhodnotenie klasifikácie bola vykonané na dvoch nahrávkach. Prvá zobrazovala v snímkach natrénovaného drona, zatiaľ čo druhá obsahovala model DJI Matrice 600 Pro, ktorého snímky neboli obsiahnuté v datasete určenom na tréovanie modelu. Implementovaný algoritmus bol schopný klasifikovať natrénovaného drona s väčšou istotou v porovnaní s novým typom drona. Istota klasifikácie závisí od vzdialenosti medzi dronom a kamerou, pričom istota klasifikácie drona pohybujúceho sa pred kamerou dosahovala 85%, no s narastajúcou vzdialenosťou táto istota klesala (Obrázok 1a) až dokým model nezačal klasifikovať drona ako vtáka či pozadie. Nenatrénovaný typ drona dosiahol istotu klasifikácie 70%, no priemerné hodnoty sa pohybovali okolo 60% (Obrázok 1b).



(a) Klasifikácia Natrénovaného Typu Drona (b) Klasifikácia Nenatrénovaného Typu Drona

Obrázok. 1: Rozdiel Istoty Klasifikácie Vzhľadom na Typ Drona

Celkový výkon navrhnutého algoritmu aplikovaného na nahrávku `untrained.mp4` je zhrnutý v (Tabuľka 1). Vzhľadom na tieto zistenia vieme určiť, že nasadenie grafickej karty produkuje horšie výsledky ako využitie procesora. Vyhodnotená nahrávka pozostávala z 61 sekundového záznamu so snímkovou frekvenciou 30, zatiaľ čo navrhnutý algoritmus bol schopný detegovať a klasifikovať 1 objekt v snímke o priemernej frekvencii 16 snímkov za sekundu, čím narástol aj celkový beh skriptu, ktorý sa trojnásobne zvýšil v porovnaní s pôvodnou nahrávkou.

Resolution	GPU/CPU	Kernel Size	Average FPS	Average Detections	Wall Time
1080x1920	CPU	(3,3)	16.47	0.56	2:56
864x1536	CPU	(3,3)	16.13	0.40	2:59
1080x1920	CPU	(1,1)	16.02	1.00	3:01
864x1536	CPU	(1,1)	15.81	0.89	3:00
1080x1920	GPU	(3,3)	14.21	0.56	3:21
864x1536	GPU	(3,3)	14.21	0.40	3:20
1080x1920	GPU	(1,1)	11.49	1.00	3:59
864x1536	GPU	(1,1)	12.18	0.89	3:46

Tabuľka. 1: Vyhodnotenie Aplikovaného Algoritmu na Nahrávku `untrained.mp4`

Detection and Recognition of Drone Movement in Video

Declaration

I hereby declare that this Bachelor's thesis was prepared as an original work by me under the supervision of prof. Ing. Martin Drahanský Ph.D. The paper has been written by me and has not been submitted for any previous degree. I have listed all the literary sources, publications and other sources, which were used during the preparation of this thesis.

.....
Simon Lapšanský
May 17, 2022

Acknowledgements

I would like to express my gratitude towards my supervisor, prof. Ing. Martin Drahanský Ph.D., who helped me arrange the drone dataset recording covered by Ing. Martin Sakin, to whom I would also like to express my gratitude for his time and quick response.

Contents

1	Introduction	2
2	Technologies used for drone detection	3
3	Moving Object Detection and Classification Algorithms	4
3.1	Feature-based Methods	4
3.2	Deep Learning-based Methods	8
4	Proposed Approach	16
4.1	Dataset	16
4.2	CNN Model	17
4.3	Moving Object Detection module	17
4.4	Object Classification module	18
5	Implementation	19
5.1	Dataset Creation	19
5.2	Object detection module	20
5.3	Training of the Model	21
5.4	Object Classification Module	22
6	Evaluation	24
6.1	Model Evaluation Metrics	24
6.2	Numerical Metrics	24
6.3	Object detection Metrics	25
6.4	Evaluation of the Trained Model	26
6.5	Evaluation of Object Detection Module	30
6.6	Evaluation of Object Classification Module	32
6.7	Performance of the Approach	34
7	Conclusion	35
7.1	Proposed Improvements	35
	Bibliography	36

Chapter 1

Introduction

In recent years, the usage of drones has become more available and affordable for the general public. Drones are also being used for commercial purposes such as surveillance, delivery of medical goods, search and rescue, and many more. With the commercial use of drones many risks occur. As mentioned before, commercial drones are being used for surveillance, this provides attackers with an affordable way to observe restricted areas. On July 27th, during protests at the federal courthouse man was charged for operating a drone in a restricted zone.[17] There have also been plenty of incidents where drones collided with aircraft. One of the incidents happened when a drone collided with a commercial airplane. The drone struck one of the plane's wings with eight people aboard though the aircraft managed to land safely. [1] These accidents present a huge problem that needs to be solved. This paper focuses on solving this problem by detecting drones in real time.

Among the detection and classification technologies, falls the usage of radar, radio-frequency (RF), acoustic sensors, and camera sensors.[24] This paper is based on camera-based drone detection. Camera-based drone detection on its own is ineffective. That's why these methods are combined with the software used for the detection and classification of drones.

Detection of drones is achieved by feature-based methods such as background subtraction. Simple versions of these methods such as frame differencing are not efficient enough, so to increase their efficiency, background subtraction methods based on the gaussian mixture model are used for this approach. The Convolutional neural network (CNN) handles the classification process.

The following paper introduces drone detection technologies in the (Chapter 2), followed by furthermore explanation of the Camera-based detection algorithms unfolds in the (Chapter 3). After an explanation of the utilized theory within the paper, the proposal of the approach is described within the (Chapter 4). The implementation of the proposed approach contained within the (Chapter 5), gives throughout insight into the proposed approach. This implementation is then applied to evaluation dataset, producing results clarified in the (Chapter 6).

Chapter 2

Technologies used for drone detection

This chapter provides a basic introduction to drone detection technologies. Each technology has its pros and cons, which will be unfolded in the following sections.

Radar-based Drone Detection

Radar detection is mostly used for large aircraft. Detection fails when the flying object is as small as a drone. If radars could pick up smaller objects, there would be an increase in false alarms due to the detection of birds and other small flying objects.[7]

RF-based Drone Detection

RF-based detection is one of the most efficient ways to long-range detect drones. The effective range of drone detection is roughly 500m. This approach captures communication between the drone and ground controller. All non-drone signals are classified as noise. Drone controllers have a unique RF fingerprint. These fingerprints combined with machine learning are used for drone classification.[5] The issue occurs when the drone is operated without ground control and has already pre-programmed flying course.[26]

Acoustic Sensor-based Drone Detection

The main benefit of using acoustic sensors as detection is efficiency in low-visibility. The sensitivity of the sensors to ambient noise affects detection. Sensors are practically pointless in a loud environment. Windy conditions can also affect the quality of detection.[25]

Camera-based Drone Detection

The main focus of camera-based drone detection is to detect drones without RF transmission. The Problem of camera detection appears in bad weather conditions or in dark. These issues can be partially solved by thermal sensors.[19] In this paper, we look into solving of the camera problems provided by various backgrounds with distractions such as leaf movements.

Chapter 3

Moving Object Detection and Classification Algorithms

Based on the research done in recent years, camera-based drone detection can be divided into two groups. The first approach utilizes feature-based methods, while the second approach focuses on deep learning-based methods. The deep learning-based methods are in most cases used for detection followed by classification.

3.1 Feature-based Methods

Feature-based methods consist of optical flow, background subtraction, frame differencing, and edge detection. The focus of these methods includes low-level image processing operations.

Background subtraction is used for the detection of static scenes. This method tries to subtract the current image pixel-by-pixel from a referenced background image. Pixels, where the difference surpasses the threshold, are classified as foreground. After the creation of foreground operations such as dilation, filtering and erosion are needed to reduce noise and enhance detected objects. This method can be modified by different classification methods of foreground and its post-processing.[20]

Frame differencing uses pixel difference between two or three consecutive frames. This method is adaptive to dynamic scene changes. It is simple to implement and has low computational complexity. This simplicity brings the issue of not being able to extract all relevant pixels of moving objects. [29] This method fails to segment non-background objects if they stop moving. [4]

The optical flow method detects objects based on the relative velocity of objects in a scene. This method is adaptive to dynamic scene changes, however, has high computation complexity and because of that it is not applicable for real-time detection.[23]

The moving edge method works with a difference of an image as a time gradient, while the edge image is a space gradient. Moving edge is defined by logic operation AND of these two images. Unlike the frame differencing method, the moving edge method has no issues with noise caused by illumination, since this method does not rely on brightness. However, this does not mean that this method is not prone to noise from other sources.[31]

3.1.1 Background subtraction approach

This method is widely used for detecting moving objects in videos from static cameras. The goal of this method is to detect the difference between current frame and a background image. Background subtraction uses function $V(x, y, t)$ as a video sequence where x and y are pixel spatial location variables and t represents time dimension.

3.1.1.1 Background Subtraction using Frame Differencing

Frame difference (absolute) of this method is defined as

$$D(t + 1) = |V(x, y, t + 1) - V(x, y, t)|$$

The image background is a frame at time t . The issue with this assumption is that the method works only if all foreground pixels are moving and background pixels are static. Outputs of these functions can be filtered by thresholding. The threshold is compared with a difference and depending on the value of the threshold is the difference filtered out or accepted. The speed of object movements determines the values of the threshold. Faster movement requires higher threshold values.[20]

3.1.1.2 Gaussian Mixture Model (GMM)

Model is used for the representation of normally distributed subpopulations within an overall population. This model is used for multiple object tracking, where the number of mixture components and their means predict object location at each frame.[11]

One-Dimensional Model

This design is used to model the probability distribution of one characteristic at one pixel. Distribution function (Eq. 3.1) is based on sum of K 1-Dimensional Gaussians. Normalized gaussian (Eq. 3.2) where σ is the width of Gaussian and μ is the mean and ϕ represents the scale of each gaussian. The normal distribution is achieved when the sum of ϕ is equal to one. (Eq. 3.3)

$$p(x) = \sum_{i=1}^K \phi_i \mathcal{N}(x | \mu_i, \sigma_i) \quad (3.1)$$

$$\phi_i \mathcal{N}(x | \mu_i, \sigma_i) = \phi_i \frac{1}{\sigma_i \sqrt{2\pi}} \exp \left(-\frac{(x - \mu_i)^2}{2\sigma_i^2} \right) \quad (3.2)$$

$$\sum_{i=1}^K \phi_i = 1 \quad (3.3)$$

Multi-Dimensional Model

Unlike the one-dimensional model, this model can be used to normally distribute the probability of multiple characteristics at one pixel. Distribution function (Eq. 3.4) is same as in one-dimensional approach, the only difference is in multi-dimensional Gaussian function (Eq. 3.5). The vector of means from all dimensions is represented by μ and Σ represents the covariance matrix. The normal distribution is achieved in the same demeanor as in the previous model. (Eq. 3.6)

$$p(\vec{x}) = \sum_{i=1}^K \phi_i \mathcal{N}(\vec{x} \mid \vec{\mu}_i, \Sigma_i) \quad (3.4)$$

$$\phi_i \mathcal{N}(\vec{x} \mid \vec{\mu}_i, \Sigma_i) = \phi_i \frac{1}{\sqrt{(2\pi)^K |\Sigma_i|}} \exp\left(-\frac{1}{2} (\vec{x} - \vec{\mu}_i)^T \Sigma_i^{-1} (\vec{x} - \vec{\mu}_i)\right) \quad (3.5)$$

$$\sum_{i=1}^K \phi_i = 1 \quad (3.6)$$

3.1.1.3 MOG2 method

MOG2 is a background subtraction method based on the Gaussian mixture model. This method is based on these two papers [32] [33]. Unlike its predecessor the MOG with the static number of Gaussian kernel distributions. The newer improved version automatically selects the number of Gaussian kernels individually for each pixel. This change results in a more resilient method with higher adaptability to illumination changes within the frame.

3.1.2 Morphological Operations

The approach of the background subtraction methods comes with drawbacks such as false object detection of the noise. Basic morphological operations such as Dilation and Erosion can improve these object detection algorithms. Application of these operations can provide removal or noise, filling out of the holes within the detected objects, and such.

3.1.2.1 Dilation

Dilation is a process where the binary image is expanded from its original shape. The expansion of the image is defined by the structuring element. In most cases, the structuring element is interpreted as a rectangle or circle with the size represented by the kernel. The size of the kernel is usually defined by odd numbers with the anchor point of the kernel being the center.

The dilation process iterates over the input image using the kernel anchor point until the kernel pixels overlap with the image pixels. If the overlap occurs, the actual anchor pixel position gets replaced by the maximal value. This operation causes the image to grow.

3.1.2.2 Erosion

Erosion is the counter-process of dilation. Erosion takes the input image and shrinks it depending on the structuring element. The kernel of the erosion process serves in the same manner as in the dilation process.

The iteration of the erosion process is the same as in the dilation case, however, the erosion replaces anchor pixel position values by minimum value, until complete overlap between the image and structuring element occurs. The state of complete overlap doesn't occur that often, depending on the image, thus the outputted image shrinks.

3.1.3 Optical Flow

Optical flow is a pattern of per-pixel motion estimation between two consecutive frames. This phenomenon is represented as a 2D vector field, where each vector shows a displace-

ment vector based on the movement from one frame to the other. The cause of this motion is based on object or camera movement. [16]

For optical flow to work properly illumination of the frame shouldn't change drastically, so that pixel intensities stay consistent from frame to frame. It is also required for the neighboring pixels to traverse in a similar motion as the observed pixel.

While these assumptions are fulfilled, we can take the intensity of the pixel in the frame $I(x, y, t)$, at positions x and y , with additional time value as t is added. Since the intensity of the pixel does not change, the pixel position in the next frame can be represented by the formula (Eq. 3.7).

$$I(x, y, t) = I(x + dx, y + dy, t + dt) \quad (3.7)$$

(dx, dy) represent the change in the coordinates from one frame to the other, also the change of time is shown by dt .

By Taylor-polynomial approximation of the right side of the equation and removal of the common terms and time division dt , the optical flow equation (Eq. 3.8) is produced.

$$f_x u + f_y v + f_t = 0 \quad (3.8)$$

Where:

$$\begin{aligned} f_x &= \frac{\partial f}{\partial x} & u &= \frac{dx}{dt} \\ f_y &= \frac{\partial f}{\partial y} & v &= \frac{dy}{dt} \end{aligned}$$

(f_x, f_y) are the gradients of the image, f_t represents gradient along time, however, the (u, v) values are unknown. This so-called Optical Flow equation (Eq. 3.8) cannot be solved with two unknown values, so in order to achieve a solution, few optical methods are provided.

These methods can be divided into two groups sparse and dense optical flow. The sparse optical flow focuses only on the pre-processed features such as edges and thus does not compute the motion vectors for every pixel, therefore the chance of not detecting a moving object is higher. The Dense optical flow eliminates this drawback, by calculating the motion vector for every pixel in the frame.

3.1.3.1 Lucas-Kanade Method

Lucas-Kanade is one of the most used sparse optical flow methods. This approach assumes that the displacement of the pixel is within the neighborhood of the specific pixel. Each point in the neighborhood contains a similar vector movement. therefore can be calculated in the same manner. The number of the pixels within the neighborhood is specified by value n , which also sets the number of optic flow equations (Eq. 3.9), where each pixel inside the neighborhood is represented by p_i .

$$\begin{aligned} f_x(p_1) u + f_y(p_1) v - f_t(p_1) &= 0 \\ f_x(p_2) u + f_y(p_2) v - f_t(p_2) &= 0 \\ \vdots & \\ f_x(p_n) u + f_y(p_n) v - f_t(p_n) &= 0 \end{aligned} \quad (3.9)$$

These equations can be written as matrices using $Av = b$, where

$$A = \begin{bmatrix} f_x(p_1) & f_y(p_1) \\ f_x(p_2) & f_y(p_2) \\ \vdots & \vdots \\ f_x(p_n) & f_y(p_n) \end{bmatrix} \quad v = \begin{bmatrix} u \\ v \end{bmatrix} \quad b = \begin{bmatrix} -f_t(p_1) \\ -f_t(p_2) \\ \vdots \\ -f_t(p_n) \end{bmatrix}$$

The solution of these equations is achieved by using least-squares principle. By applying the principle onto $Av = b$ formula, we get $A^T Av = A^T b$, where A^T is equal to transpose of the matrix A . v value is derived from the formula, resulting in $v = (A^T A)^{-1} A^T b$ (Eq. 3.10), which produces desired output.

$$\begin{bmatrix} u \\ v \end{bmatrix} = \begin{bmatrix} \sum_i f_x(p_i)^2 & \sum_i f_x(p_i) f_y(p_i) \\ \sum_i f_y(p_i) f_x(p_i) & \sum_i f_y(p_i)^2 \end{bmatrix}^{-1} \begin{bmatrix} -\sum_i f_x(p_i) f_t(p_i) \\ -\sum_i f_y(p_i) f_t(p_i) \end{bmatrix} \quad (3.10)$$

The sum used in these equations runs from 1 to n , where n is the number of pixels from within the neighborhood.

3.1.3.2 Farneback algorithm

Farneback algorithm is one of the most used dense optical flow algorithms. The dense optical flow provides better accuracy at cost of an increase in the computation.

The farneback algorithm utilizes the polynomial expansion to approximate the neighborhood of each pixel while the approach focuses only on the usage of quadratic polynomials [6]. The estimation of displacement is estimated over the neighborhood as a whole, rather than point-wise, since the point-wise estimation produces a lot of noise.

This approach introduces a series of refinements, providing more accurate estimations such as multi-scale displacement estimation. The multi-scale displacement estimation tackles the occurring issue of large displacements. With higher displacements, the improvements based on the iterations are meaningless. This problem is reduced by displacement analysis, done at a coarser scale by applying the multi-scale approach. The estimation takes place at a coarser scale, providing rough but reasonable displacement estimations, which will be then passed through lower scales, obtaining more accurate results. The drawback of this approach is the computation increases quite a bit, since the polynomial expansion must be recomputed for each scale.

3.2 Deep Learning-based Methods

Deep learning methods consist of algorithms based on the structure and functionality of artificial neural networks. These methods utilize many hidden layers, creating robust networks.

3.2.1 Artificial Neural Networks (ANN)

ANNs are computational models inspired by the human brain. These networks consist of many connected nodes. Each node performs a simple mathematical operation, while the outputs of nodes differ from one node to the other and are determined by their mathematical operation. In order to learn and calculate very complex functions, connected nodes

within the network must have their parameters carefully set up. Artificial neural networks have enabled many advances in artificial intelligence, including image recognition or voice recognition.

3.2.1.1 Artificial and Biological Neuron connections

The human nervous system consists of roughly 90 billion neurons (Figure 3.1b) which are connected with roughly 10^{14} synapses. Each neuron inputs signals through its dendrites and produces output signals on its axon. Axon uses synapses while creating connections with another neuron's dendrites. There are many types of biological neurons with custom properties. Dendrites of biological neurons execute complex nonlinear computations and synapses form complex non-linear dynamical systems. However, the artificial neuron is a very coarse model of a biological neuron. Mathematical model of artificial neuron (Figure 3.1a) can be interpreted as function where inputs x_n represent the dendrites. These inputs are multiplied with the dendrites of another neuron. These weights w_{nj} are used in order to control the strength of the influence of the neuron on the adjacent neuron. Multiplied values are then summed up in the body of an artificial neuron. If the sum exceeds the threshold level, the neuron „fires“ a spike along its axon. The rate of this process is modeled with an activation function.

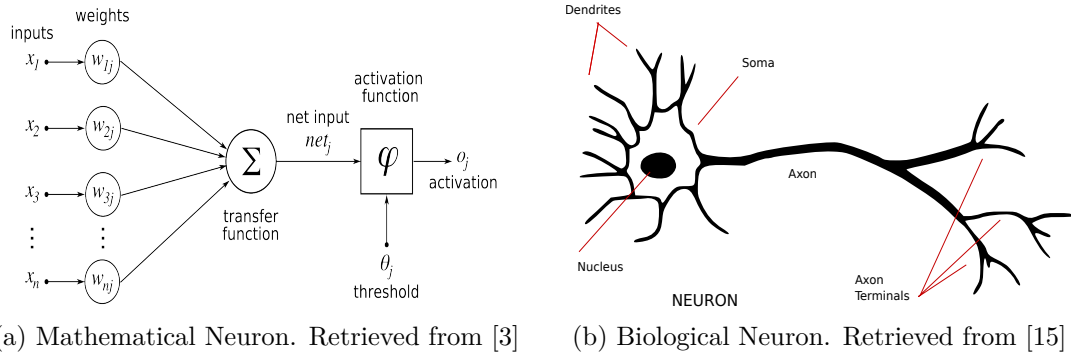


Figure 3.1: Comparison of Neuron Models

The activation function takes a single number and performs certain mathematical operations on it. In the past, sigmoid function $\sigma(x) = \frac{1}{(1+\exp(-x))}$ (Figure 3.2a). Among the activation functions, this was the most common activation function. Sigmoid non-linearity takes a real-valued number and distributes it into a range from 0 to 1. Large negative numbers become 0 while large positive numbers become 1. These values interpret the firing rate of the neuron. Not firing neuron is represented by (0) and fully-saturated firing at the maximum frequency by (1). The main drawback of this activation function comes from either tail of the range, where arise issues with saturation. [10] The gradient at these regions is close to none and almost no signal flows through the neuron to its weights and recursively to its data. To prevent saturation and failure of learning, initial weights must be optimally determined.

Hyperbolic tangent non-linearity is very similar to the sigmoid function. This function can be interpreted as scaled sigmoid $\tanh(x) = 2\sigma(2x) - 1$ (Figure 3.2b). The hyperbolic tangent's distribution ranges from -1 to 1, this makes the function outputs zero-centered in contrast to the non-centered sigmoid outputs. Which is clearly shown in figure (Figure 3.2). However, the problem with the saturation of neurons remains.

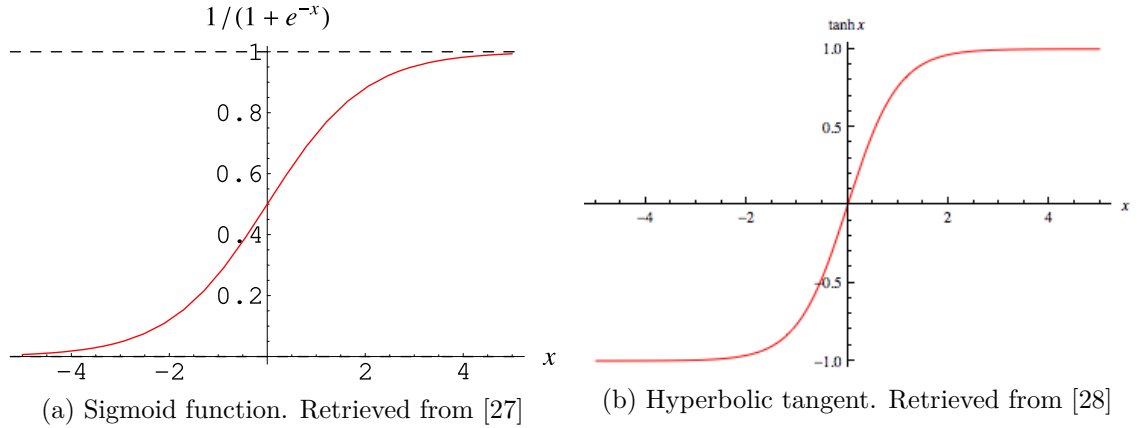


Figure 3.2: Non-linearity activation functions

In recent years the Rectified Linear Unit (ReLU) became very popular. ReLU computes $ReLU(x) = \max(0, x)$ (Figure 3.3a), which means that activation is thresholded at zero. Unlike ReLU's predecessors, ReLU doesn't rely on expensive mathematical operations and its implementation is feasible by simple thresholding of the matrix of activations at zero. ReLU is prone to dying out during the training stage. This issue occurs when a large gradient flowing through a neuron updates neuron in such a manner, that it causes the neuron to never activate on the arrival of any data. This makes the flowing gradient through the neuron forever equal to zero.

Leaky ReLU was created in order to solve the dying out of neurons in the previous version. This approach doesn't threshold activation at zero for negative inputs. However, uses a small positive slope α . $LeakyReLU(x) = \max(\alpha x, x)$ shown at (Figure 3.3b), where α is small constant that prevents, neurons from dying. The performance of this approach is not consistent, although if used correctly, it can prevent parts of the network from dying out.

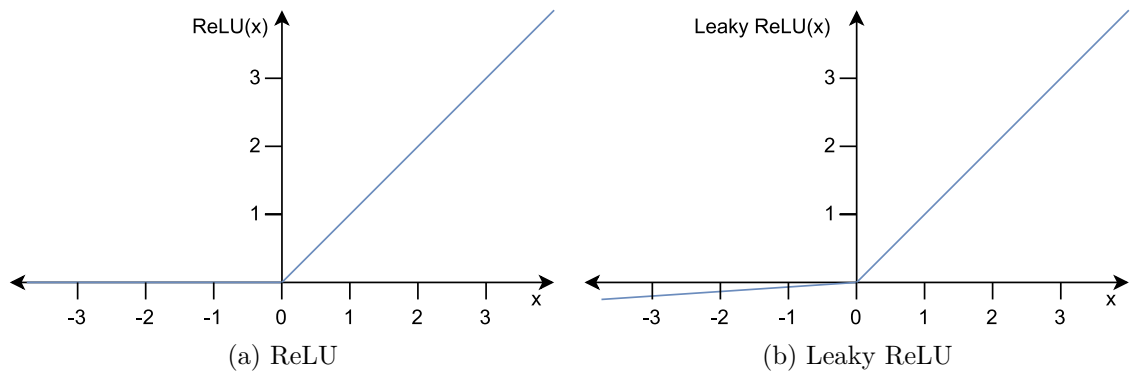


Figure 3.3: Comparison of ReLU activation functions

3.2.1.2 Neural Networks

Neural networks are made up of collections consisting of neurons, which are joined in acyclic graphs. NNs are often modeled into layers of neurons. The most common type of layering

neurons is the usage of a fully-connected layer. With this approach, neighboring layers are fully connected, however neurons within a single layer share no connections at all.

Fully connected NNs use three types of layers. N-layer feed-forward neural networks consist of one input layer, N-2 hidden layers, and a single output layer. Single-layer NNs do not contain hidden layers, thus inputs are directly mapped onto outputs (Figure 3.4b). In figure (Figure 3.4a) is shown simple 2-layer feed-forward neural network. This network takes three inputs (x_1, x_2, x_3), producing a single output. The Inputs and outputs are chosen based on the given problem. However, the number of neurons within hidden layers is provided by the design of the network [10].

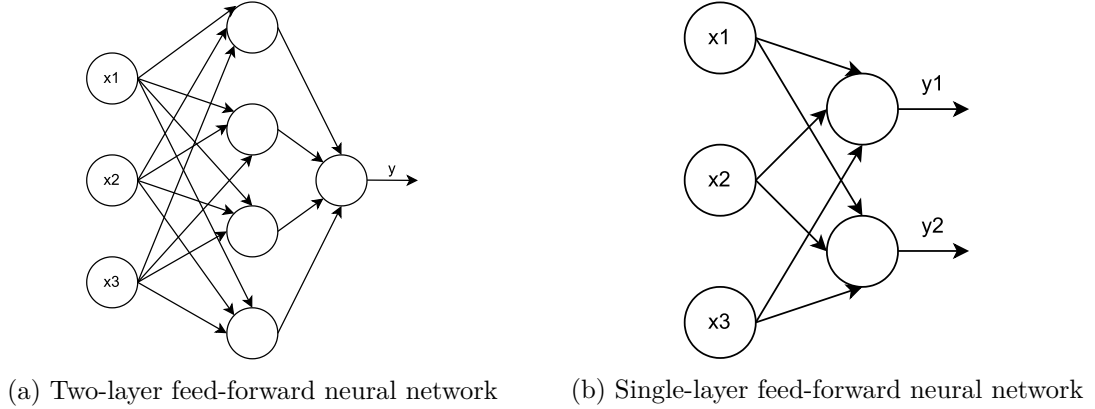


Figure 3.4: Examples of Feed-forward Neural Networks

3.2.2 Convolutional Architectures

Convolutional neural networks (ConvNet, CNN) are commonly used in image recognition for their ability on complex images. CNNs consist of multiple neuron layers, where each neuron computes weights by non-linear operations depending on the previous layer's outputs. These layers consist of mostly convolutional layers, pooling layers, and fully-connected layers. [21]

3.2.2.1 Convolution layer

2D Convolution is a simple operation, which takes a set of weights and multiplies it with input. A set of weights is represented by a kernel filter, which slides over the input image matrix, performing multiplication with current overlapping input, summing it all up into a single output pixel [22].

The convolution layer uses kernel filters, computing the convolutional operations. Kernel filters contain the same dimensions mostly 3x3. However, these dimensions can be larger as long as they don't exceed input image dimensions. Depending on the size of the filter, features will get extracted from the input image. Larger filter sizes are likely to miss out on the important features of the image and are more prone to producing noise inside the output matrix. Parameters of these filters are learned throughout the training process. Kernel filter traverse through the input image depending on its stride.

In figure (Figure 3.5), input image dimensions are 4x4, while the kernel filter dimensions are 3x3, this creates 2x2 output matrix. Stride is equal to one, this means that after calculating the first value (red), the kernel filter position is right-shifted by one, getting the next value (orange).

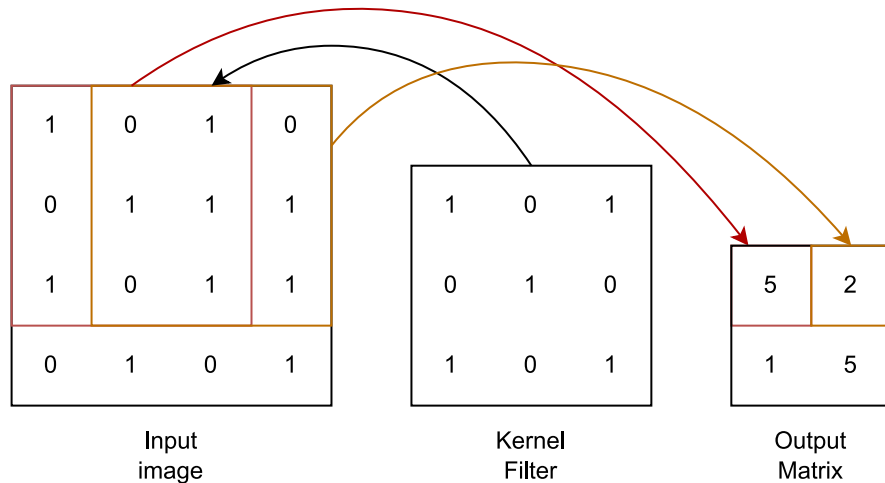


Figure 3.5: Convolution

3.2.2.2 Pooling layer

Pooling layers are used in-between successive convolutional layers. These layers reduce the spatial size of the image to cut down on computation within the network. The most common form of pooling layer is the usage of a 2x2 filter with the stride of 2 (Figure 3.6). With this approach, the MAX operation is used to get downsized values. In the past operations such as average pooling were used, but have recently fallen out.

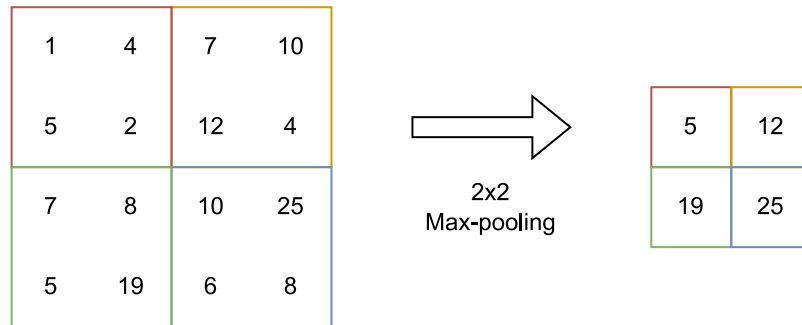


Figure 3.6: Max pooling example

3.2.2.3 Fully-connected layer

Fully-connected layer takes outputs of preceding layers as input. The main task of this layer is to collect previous features and perform logical operations on them. The results of these operations provide the final decision. Algorithms based on deep learning are split into two and one-stage detectors.

3.2.3 Two-Stage Detectors

This approach divides detection into region proposal and classification stages. In the first stage, several object candidates are located, using reference boxes. These objects are also known as regions of interest (RoI). In the second stage, these regions are classified and their localization is furthermore refined.[2]

3.2.3.1 R-CNN

Region Convolutional Neural Network (R-CNN) paved the way for the two-stage convolutional neural network (CNN) object detection.[30] R-CNN consists of three modules. The first module generates category-independent region proposals from the input image. These proposals define a set of available candidate detections for a detector. From each region is extracted a fixed-length feature vector using CNN which forms the second module. The last module is a set of class-specific linear support vector machines also known as SVMs.[8]

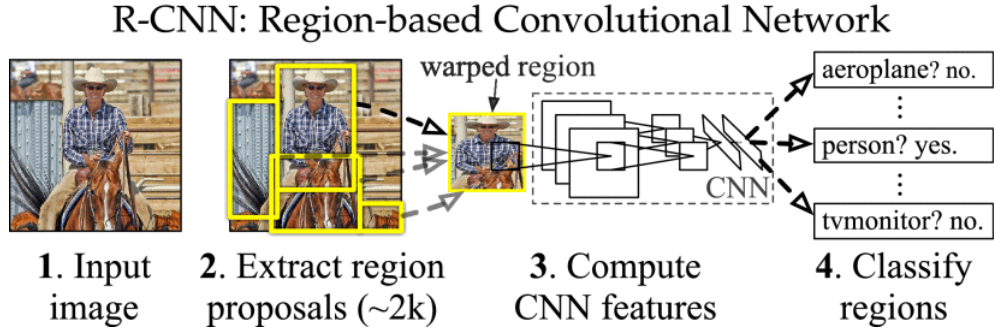


Figure 3.7: R-CNN. Retrieved from [8]

3.2.3.2 Fast R-CNN

Fast R-CNN was created by the same author as the previous paper R-CNN. This method focuses on the drawbacks of R-CNN in order to create a more sophisticated object detection algorithm. The difference between these methods is the approach of feeding input images to CNN. Unlike in R-CNN whole input image is fed to CNN in order to generate a convolutional feature map. RoI are pulled from obtained feature map which is then reshaped into fixed size, using RoI pooling layer so that they be fed into fully connected layers. The softmax layer predicts the class of the proposed region-based of RoI feature vector. RoI feature vector is also used to get offset values of the bounding box. Fast R-CNN is much faster than R-CNN because convolution operation is done once per image.[9]

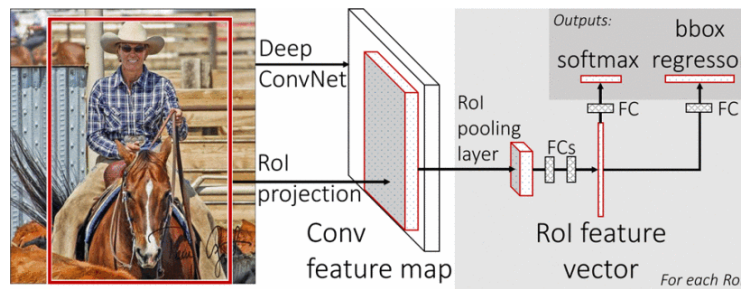


Figure 3.8: Fast R-CNN. Retrieved from [9]

3.2.3.3 Faster R-CNN

Faster R-CNN does not rely on selective search to find out regional proposals. Unlike predecessors, this method uses a separate network to predict region proposals on convolutional feature maps.[18] As a result of this improvement, this method is applicable to real-time object detection.

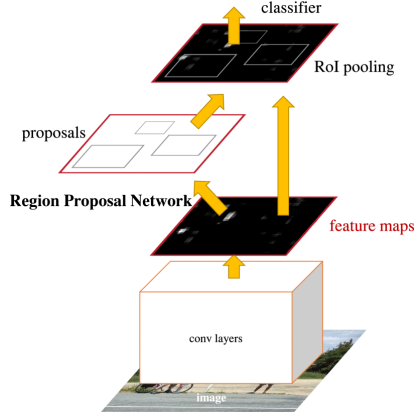


Figure 3.9: Faster R-CNN. Retrieved from [18]

3.2.4 One-Stage Detectors

Unlike the two-stage detector approach one-stage, the detector contains a fully convolutional network that directly provides bounding boxes and object classification. The approach eliminates regional proposal generation.

3.2.4.1 SSD

SSD (Single Shot Multibox Detector) (Figure 3.10) uses ideas from the regional proposal network from Faster R-CNN and multiscale convolutional features from the YOLO algorithm. SSD operates similarly to YOLO by predicting a fixed amount of bounding boxes.

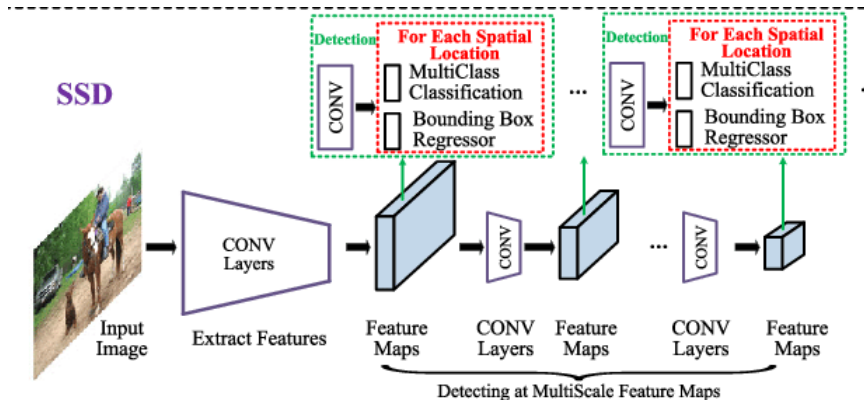


Figure 3.10: SSD algorithm. Retrieved from [12]

3.2.4.2 YOLO

YOLO (You Look Only Once) (Figure 3.11) is an anchor-based algorithm that divides the input image into $S \times S$ grid. For each grid are predicted bounding boxes and probabilities. Predictions are encoded as tensor $S \times S \times (5B + C)$. YOLO algorithm may experience issues while detecting a small object, because of the coarseness of grid division.

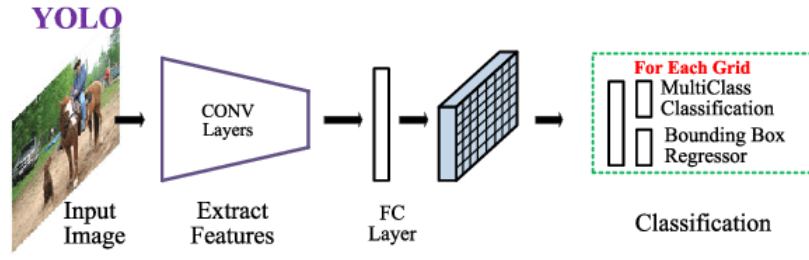


Figure 3.11: YOLO algorithm. Retrieved from [12]

3.2.4.3 YOLOv5

YOLOv5 is the most recent version of the YOLO algorithm. However, this version of the algorithm was proposed by different authors compared to the previous versions of the algorithm. This algorithm was presented just a month after the release of the YOLOv4 version in April 2020. As of today, the paper on YOLOv5 by the creators is yet to be published. The architecture of this method can be displayed using the tensorboard. Based on the availability of the architecture, papers by many researchers were proposed. The details of the architecture were described within the [14] paper, which compares the efficiency of YOLO models, specifically the YOLOv3, YOLOv4, and YOLOv5 model.

The YOLOv5 utilizes PyTorch instead of the previously used DarkNet. The architecture of YOLOv5 is formed by the backbone, neck, and head of the model.

The backbone of the model utilizes the extraction of important features from the given input image. The backbone of YOLOv5 was represented by CSPDarknet53. The utilization of this backbone solves repetitive gradient information in large backbones by integrating the gradient changes into a feature map, thus reducing the speed of inference while increasing accuracy and reducing the size of the model. [14]

The model's neck is mainly used for the generation of feature pyramids. The usage of feature pyramids helps the model to efficiently generalize object scaling, thus giving the model ability to correctly identify the same object of different sizes. The YOLOv5 uses the path aggregation network (PANet), in order to increase information flow. The information flow is obtained by PANet's feature pyramid network (FPN), including bottom-up and top-down layers, which improves the propagation of low-level features within the model. This provides an increase in the accuracy of the object localization.

The head of the model is the same as proposed with the YOLOv4 and YOLOv3 methods. The model's head mainly covers the final detection part by applying the anchor boxes onto the feature, while generating the final output vectors with class probabilities, objectness scores, and bounding boxes. The YOLOv5's head generates three different outputs of feature maps, achieving multi-scale prediction. This allows the model to efficiently predict small to large objects.

The YOLOv5 takes the input image and feeds it into the CSPDarknet53, in order to achieve feature extraction, which will be then fed into PANet, achieving the feature fusion. The fused features are then passed into the YOLO layer, generating the predicted results.

The YOLOv5 utilizes Leaky ReLU and sigmoid as activation functions. The Leaky Relu is used in combination with hidden layers, while the sigmoid is implemented within the final detection layer.

Chapter 4

Proposed Approach

The proposed approach of this paper is to detect moving objects within the frame, followed by the decision of whether the classified object falls into the drone category or not. The goal of this approach is to merge existing algorithms in a way, which produces a method specific for the detection and classification of drones depicted in different settings.

Detection and classification are split into two different modules. The first module handles the detection of drones using feature-based detection methods, while the other module uses CNN architectures in order to classify these previously detected objects.

These modules will be implemented within the script, which will take the inputs in form of a real-time camera stream or recordings. Proposed detection and classification will be then applied to these input frames, resulting in the plotting of the detected and classified objects bounded by boxes.

The evaluation of the approach will be based on the observations made while experimenting with the application of the implemented script onto the evaluation dataset.

4.1 Dataset

Dataset used in this paper will consist of multiple real-life drone videos situated in the sky or against different types of backgrounds. Dataset will also depict drones and birds. The dataset should include drone, bird, and background class. However, this could be expanded by aircraft or vehicles, depending on the setting of the recorded video.

The usage of this dataset will be split into two parts. Firstly the dataset will be used in order to train the proposed model. After the training stage, the model will be deployed onto the other part of the dataset, and which results will be then evaluated.

The first video dataset contains 4 videos each depicting a different setting. Drones within these recordings are up against different backgrounds with different kinds of distractions within the frame, such as strolling people or moving leaves. These videos were filmed on the 15th of April 2022 in Brno - Medlánky in the early afternoon. At the time of the recording, the weather conditions weren't the most favorable, which will help the model in the end, because these difficulties enable the model to correctly evaluate passed images in unfavorable conditions. The recorded drone type DJI Mavic Air black was filmed by camera Canon 700D using the Canon EF-S 18-55mm lens. The video format of these recordings consists of mp4 files in 1080p recorded at 24 fps using the H.264 codec.

The second dataset used for evaluation consists of 4 videos too. However, these videos were not recorded for this specific thesis, thus only parts of these videos are usable for

evaluation. The black drone Dji Matrice 600 is depicted in these videos. The videos were shot in 4K resolution with the codec and camera lens being unknown. The recording of the dataset took place in June of 2019 in a Czech village called Jínce. The main problem of these recordings comes with the camera movement, which is not applicable for detection using the background subtraction method.

In order to increase the size of the dataset, videos of birds¹ were added to the first dataset mentioned above. This additional data will help with the creation of a more robust model, resilient to distraction within the frame, caused by birds.

4.1.1 CNN Model Dataset

In order for the CNN model to be trained, the model must be provided with annotated images. These images will consist of the detected objects extracted out of previously mentioned dataset recordings. The detected objects must be then efficiently annotated. These annotated images will be then utilized in the creation of the training dataset compatible with the YOLOv5 model.

4.2 CNN Model

This approach focuses on the usage of the YOLOv5 model (Section 3.2.4.3) as the fully convolutional network used for object classification. The YOLOv5 is the leader in real-time object classification, based on its speed combined with high accuracy. For our approach, YOLOv5s is proposed as the go-to model for its speed of classification. This is the smallest version of the YOLOv5 model with a size of just 14 MB. Another reason for the utilization of this model is the straightforward ability of training and deployment.

The training of the model can be achieved using the training scripts provided by the GitHub repository² of the YOLOv5 model. The previously mentioned training dataset will be passed onto the script for the model to be trained.

4.3 Moving Object Detection module

Background subtraction and optical flow are two candidates for this module. Both methods come with drawbacks. Background subtraction becomes inaccurate when the detected object is close to the moving background which causes false detection. These conditions could be caused by clouds and moving leaves of trees. The major drawback of optical flow is the complexity of its computation. This complexity must be reduced in order for the method to be applied in real-time object detection.

4.3.1 Background subtraction approach

Based on [13] the MOG2 and MOG were best performing out of a bunch. The precision of foreground detection was pretty similar among these two algorithms, but there was a visible difference in processing time. Based on the paper the processing time of MOG2 was supposedly reaching 3 times the processing speed of the MOG algorithm.

This approach is based on the usage of the MOG2 subtraction method mentioned above (Section 3.1.1.3). The MOG2 algorithm will provide detected objects, which can be then

¹<https://github.com/DroneDetectionThesis/Drone-detection-dataset>

²<https://github.com/ultralytics/yolov5>

bounded by boxes. For better object extraction, morphological operations such as dilation and erosion can be used. The detected objects will be then wisely fit for model deployment.

4.3.2 Optical flow approach

The approach will work very similarly to the background subtraction method, with the only difference being the algorithms used for object detection. The Farneback dense optical flow (Section 3.1.3.2) will be used for the production of the motion vectors. These vectors will be then encoded into HSV for visualization purposes. These HSV outputs will be then converted into BGR and bounded by boxes, which will represent the detected object.

4.4 Object Classification module

The classification module will take the outputs from the detection module in a form of cropped-out images. These images will be then pre-processed and passed to the trained YOLOv5 model. The classification module will be also responsible for the final plotting of the classified objects. After the deployment of the YOLOv5 model onto the cropped out and pre-processed images, the coordinates of the results must be furthermore normalized for an accurate bounding of the classified object. The normalized coordinates will then provide bounding box coordinates, which will be plotted to the initial frame, providing visible results of the proposed algorithms.

Chapter 5

Implementation

This part of the thesis concludes the implementation of the previously mentioned approach. The python script `main.py`, using the python version 3.8.10, contains the implementation of this approach. The step-by-step creation of the dataset used for YOLOv5 model training is also clarified within the implementation chapter.

5.1 Dataset Creation

Dataset is created by a python script `annotate.py` that takes as input video, while outputting the annotated images of detected objects. The dataset consisted of 1080p recordings. However, while creating the training dataset, the scale of these frames was halved due to the low performance of the available graphics card, making this process very time-consuming. This script applies object detection module proposed in previous section (Section 5.2). Detected objects are annotated based on argument `--name`, which determines the class of detected objects. The main drawback of this approach is the amount of false positive detections of the foreground. Thresholding of the minimum and maximum area of detection by the trackbar can reduce the amount of false positive detections of the foreground. This approach doesn't erase the issue entirely but decreases it quite a bit. The False positive detections are annotated as background class, in order to improve the quality of the model.

5.1.1 Roboflow

Roboflow tool¹ is then used to process annotated images. Images are manually checked for misplaced bounding boxes, which can be easily fixed using the roboflow labeling interface. The Dataset consists of images used for training, validation, and lastly testing. Distribution among these folders is 70/20/10 with the training directory containing 70% of all the images within the dataset. Images used for validation make up 20% and the remaining 10% is represented by training images.

5.1.1.1 Pre-processing

Roboflow offers pre-processing features such as image resizing. For ConvNet's ability to perform classification, images must be multiples of 32. While testing various models and image sizes the 64x64 dimensions has shown superior results in comparison with other

¹<https://roboflow.com/>

resolutions. Within pre-processing stage, images can be static cropped or turned into grayscale, to reduce the computation. However these pre-processing tasks are not needed, since the median dimensions of the dataset are 23x15 pixels, hence the computation is rather low.

5.1.1.2 Augmentation

Image augmentation increases learning diversity for the model. Augmentation can be performed while training or directly via roboflow. Using built-in augmentation decreases the actual training time of the model. Roboflow supports image augmentations such as image rotations, blur, and random noise. The option used for our dataset is the gray-scale option. Gray-scale augmentation is applied to 25% of the dataset. This augmentation provides a more robust model, giving the model chance to classify drones depicted in various settings.

5.1.2 Dataset Metrics

The dataset created with the usage of the roboflow tool consists of 7271 images out of which 2692 represent drones, 2404 birds, leaving the last 2175 examples of background. The median image size is 23x15 pixels, meaning that the images are rather tiny.

The generated version of the dataset using the pre-processing and augmentation mentioned before produces 11690 images, creating the actual number of images used for training. These values are split up a three-way, in the same manner as mentioned above, but with different percentage distribution than before. The different distribution is caused by the augmentation process which does not apply augmentation equally onto each image, producing nearly 82% of images for training, 12% for validation, leaving just 6% for the testing stage.

For the dataset to be compatible with the YOLOv5 model, the dataset must contain annotations in text format. These annotations in text format obtain space split values specifying the `class_id` `center_x` `center_y` `width` `height` of the annotated image. The dataset must contain configuration as `yaml` file, which specifies number of classes by `nc` and with list of the classes names by `names`. This configuration also set the path of the folder containing images meant for training, setting the `train` value. The same applies to validation, which is set up using `val`. Each of these folders must contain two additional folders `images` and `labels`. The `images` folder as the name states includes images, while the annotations in text format are within `labels` folder.

5.2 Object detection module

As proposed in the approach, optical flow and background subtraction were both candidates for implementation. While implementing both methods, the optical flow processing time was significantly higher than the processing time of the background subtraction method. The main drawback of the dense optical flow was the trouble with the recognition of smaller objects. The solution to this problem could have been done by up-scaling of the frame, resulting in even higher computations, and causing an increase in the processing time.

Depending on these findings, the object detection module utilizes the MOG2 background subtraction algorithm based on the Gaussian mixture model. The implementation of this method is available within the OpenCV library².

²<https://opencv.org/releases/>

The detection module uses the implementation of the MOG2 background subtraction algorithm in order to detect moving objects. Without any pre-processing of the frame, the detected objects would contain a lot of noise, therefore before the detection of the objects takes place, a background subtractor is created and stored in `mog2Subtractor` variable using the OpenCV library method `createBackgroundSubtractorMOG2(200, 150, False)`, where the first parameter represents the length of the history of the background. The threshold of a decision on whether the pixel is well described by the background model is set using the second parameter, while the last parameter decides whether the shadows of the detected object will be marked. The pre-processing of the frame and actual detection of the moving objects is implemented within the `objectDetection` function, which takes the actual frame as input along with the scale value, which alters the scale of the frame. The `mog2Subtractor` is then applied to the resized frame.

In order to decrease the amount of noise, morphological operations such as erosion and dilation are applied to the subtracted frame. The erosion is applied using the OpenCV `morphologyEx` method, using the ellipse with the kernel size based on the value of `Kernel Size` trackbar, which is accessible at the bottom of the plotted output. The initial value of kernel size is set to (1,1), but this value can be changed using the `--ksize` argument, which takes the values ranging from 1 to 15. After the erosion, two-step dilation continues, where the first dilation achieved by OpenCV `dilate` method, uses square-shaped kernels with the size of (7, 7) in 3 iterations, followed by the second dilation carried out by the previously mentioned OpenCV method `morphologyEx`, using the circle-shaped kernels with the size of (25, 25). The other noise reduction method is based on the thresholding of the minimum area of the detected object. This thresholding is applied in real-time utilizing the trackbar at the bottom of the frame with the default value of three. However, the initial value can be set using the `--a` argument.

The detected objects must be then bounded utilizing the found out contours by the OpenCV `findContours` method. The contours are then processed in order to produce bounding box coordinates using the OpenCV `boundingRect` method. While iterating over these coordinates, detected objects are cropped based on these values and stored within the `crop_list` list. The `rect_list` list contains the bounding box coordinates of detected objects. The number of detected objects within the frame is stored in the `cnt` variable.

The `crop_list`, `rect_list` and `cnt` variables along with the scaled frame are then returned out of the `objectDetection` function back into main program.

5.3 Training of the Model

The training of the model was achieved using the roboflow's and google collab's template³. This template contains a basic run-through of the YOLOv5 training process. The training process using the template focuses on the usage of YOLOv5 scripts while adding the ability to easily import datasets using the roboflow tools. Using virtual GPU available from google collab, allows you to actively use your device, without straining your GPU. This also comes in handy, while working on a device missing GPU, hence the training time increases immensely.

First of all, the script clones the YOLOv5 Github repository⁴ and installs dependencies stored within `requirements.txt` file, using `pip` command. After installation of all depen-

³<https://colab.research.google.com/drive/1gDZ2xcT0gR39tGGs-EZ6i3RTs16wmzZQ>

⁴<https://github.com/ultralytics/yolov5>

dencies, the dataset needs to be imported using the roboflow library, which serves as a tool for dataset importing. Roboflow takes the API key in order to access the dataset and imports it into collab's directory. Before training takes place, the model size must be chosen. The larger the model, the more GPU memory it requires. The detection is also slower, although the precision in recognition also increases. While training this model, YOLOv5s was used. This is the smallest version, thus the fastest version. The usage of this model provides low classification time recognition, needed for real-time object classification. The model type with its values is defined as `yaml` file, which will be passed to the training script.

The `train.py` is a python script, which handles the training of the model. This script is available in the YOLOv5 Github repository and works with several arguments. In order for the script to work `--img` argument was used to define the size of the input images. As mentioned in the previous section, the dataset consists of 64x64 images, so the argument was set to 64. Then the `--batch` argument was used to specify the size of the batch of images evaluated at the same time. This was set to 24, so the training time was lowered. The number of training epochs is set using the `--epochs` arguments, for this case 130 was used because additional epochs would improve the coarseness of the model just slightly while increasing the training time by a lot. The `--data` argument takes the previously mentioned dataset configuration file. The defined model within `yaml` file is passed to script, using the `--cfg` argument. The last argument used while training was `--name`, which specified the name of the output folder of the training. The output of this script is the trained YOLOv5s model, which takes up roughly 14 MB.

5.4 Object Classification Module

As mentioned before, the object classification module is implemented within the `main.py` script. Before the classification takes part, the trained model needs to be deployed using the PyTorch⁵ method for inference of the pre-trained models. The `torch.hub.load()` method takes the YOLOv5 Github repository as the first parameter, while the `model` parameter is equal to the string `'custom'`, which in our case specifies the usage of custom pre-trained YOLOv5 model for deployment. The `source` parameter provides the path to the locally stored pre-trained model.

Before the classification of the detected objects, the object must be resized to 64x64 for model to correctly perform classification. The resized frames are then passed to the `scoreFrame` function, using the `im` parameter, which evaluate the frame, while saving the results as the DataFrame object `df` available from `pandas`. DataFrame values are then thresholded using the confidence values set up by the `--conf` argument of the `main.py` script. The output of this function consist of `labels`, `coords`, `confidence`, representing the labels, coordinates and confidence values of the classified objects.

The number of classified labels determines the number of iterations for the actual plotting of the classified objects onto the Original video frame. For an algorithm to be able to plot the bounding boxes based on the coordinates results provided by classification, the coordinates must be scaled using the original coordinates of the detected object.

The scaling of the coordinates is achieved by `cordCalculations` functions, which take the initial coordinates of the detected object, a ratio of the frame scaling used to reach 64x64 frame size, and coordinates produced by classification. These values are then normalized to fit the original frame and returned back to the main program.

⁵<https://pytorch.org/>

The plotting of the bounding boxes is carried out by `plotBoxes`, which takes the classification label, confidence values, and normalized coordinates. Based on these normalized coordinates rectangle is plotted using the OpenCV `rectangle` method with additional text plotted by OpenCV `putText` method consisting of the class name and confidence value in the top left corner of the rectangle.

Chapter 6

Evaluation

The ability to evaluate the performance of the presented method is essential. This chapter firstly describes the evaluation of training of the YOLOv5 model, but also the deployment of this model and its ability to perform in object detection and its further classification.

6.1 Model Evaluation Metrics

In order to evaluate the model, we need to define evaluation metrics. These metrics often called classification metrics, represent the possible outcomes that can occur. These four outcomes are True positives, True negatives, False positives, and last but not least False negatives, representing:

- *True positives (tp)* - occurs when model correctly predicts the positive class
- *True negatives (tn)* - occurs when model correctly predicts the negative class
- *False positives (fp)* - occurs when model incorrectly predicts the positive class
- *False negatives (fn)* - occurs when model incorrectly predicts the negative class

These metrics are then furthermore used to evaluate more coarse evaluation metrics.

6.1.1 Confusion Matrix

The evaluation of the model based on the confusion matrix takes the count of correctly and incorrectly predicted classes in order to evaluate the model. These values are based upon the previously mentioned metrics. These computed values give more insight into the quality of prediction for each class. The confusion matrix is then furthermore used for the measuring of other metrics, clarified within the upcoming section.

6.2 Numerical Metrics

Numerical metrics perform an evaluation of the model based on a single number. These numbers range from 0 to 1, where 0 represents the worst rating and 1 flawless score. Among these metrics are Accuracy, Precision, Recall, and F-score.

6.2.1 Accuracy

Accuracy (Eq. 6.1) represents the ratio of corrected predictions to the total number of predictions. This method is rather simple, therefore with many classes of inconsistent extent, this metric loses validity. This approach can give a false sense of achieving high accuracy. This issue occurs with false classifications of smaller populated classes. A solution to this drawback can be reached using the approach of a balanced accuracy metric.

$$Accuracy = \frac{tp + tn}{tp + tn + fp + fn} \quad (6.1)$$

6.2.2 Precision

Precision (Eq. 6.2) computes the proportion of correctly predicted predictions of positive class to all positively classified predictions. This metric focuses on the ability of a model to not make mistakes while classifying an object as positive.

$$Precision = \frac{tp}{tp + fp} \quad (6.2)$$

6.2.3 Recall

Recall (Eq. 6.3) represents the ratio between the predictions which were predicted to belong to a class while respecting the predictions that truly belong in the class. This metric depicts the ability of a model to recognize objects which should be classified as positive.

$$Recall = \frac{tp}{tp + fn} \quad (6.3)$$

6.2.4 F1-score

The usage of previously mentioned metrics Recall and Precision is not really common. However, these two metrics combined provide a metric called $F_1 - Score$ (Eq. 6.4), which gives valuable insight into the performance of the model. This metric determines the optimal ratio between Recall and Precision since the increase in Recall lowers the Precision values and vice versa.

$$F_1 = 2 \cdot \frac{precision \cdot recall}{precision + recall} \quad (6.4)$$

6.2.5 Precision-Recall Curve

The relationship between precision and recall mentioned before can be plotted with the usage of the precision-recall curve. The behavior of this curve is dependent on the value of the threshold for object detection. With higher threshold values, the precision increases, while the recall lowers. However, with lower threshold values, the precision goes down, which can lead to false detection. While using well-trained models, the lowering of the threshold can have little to no impact on the amount of false detection, keeping the precision high.

6.3 Object detection Metrics

For the detection of the objects within a frame, a different set of metrics must be provided. The essential metric is Intersection over Union or shortly (IOU).

6.3.1 Ground truth

Ground truth represents the ideal output of the proposed algorithm however it can be defined as the standard used for the evaluation of the algorithm. In object detection and classification, ground truth can be displayed by bounding boxes or masked-out objects. The closer outputs are to the ground truth, the better performing algorithm. Ground truth can be used in order to evaluate different metrics, which can provide a better assessment of the algorithm.

6.3.2 IoU

The essential metric using the ground truth is IoU. While detecting objects, previously mentioned metrics cannot provide information about the detection and classification within the frame however this metric focuses on the comparison of the ground truth with the predicted outputs. The main goal of this metric is to determine whether the predicted outcome is true positive or false positive. The IoU (Figure 6.1) shows the amount of the overlap between the predicted bounding box and ground truth, where the numerator is the overlapping area of the predicted bounding box and ground truth, while the denominator represents the united area of both the predicted bounding box and the ground truth.

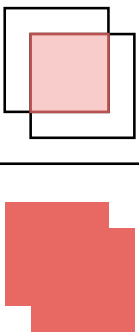
$$\text{IoU} = \frac{\text{Area of Overlap}}{\text{Area of Union}}$$


Figure 6.1: Intersection over Union for object detection

6.3.3 Average Precision (AP)

The plotting of the Precision-Recall curve is time-consuming, meaning that the usage of this metric to evaluate the model is not optimal. Average precision is used instead of the curve, by finding the area under the precision-recall curve. With higher curves, the area is also greater, making the AP also higher, creating a better-performing model.

6.3.4 Mean Average Precision (mAP)

Mean average precision is achieved by averaging the precision among classes. The mAP is often computed with specific IoU thresholds. The mAPs calculated among specific threshold values are labeled in such a manner $mAP@0.95$, where 0.95 equals the IoU threshold value.

6.4 Evaluation of the Trained Model

This section takes look at the evaluation of the trained model, applying the previously mentioned evaluation metrics. The YOLOv5 training python script `train.py` plots graphs

depicting the evaluation metrics. The whole process of training consisting of 130 epochs took 1 hour, 47 minutes, and 56 seconds.

6.4.1 Confusion Matrix

The confusion matrix produced by the YOLOv5 (Figure 6.2) is a normalized confusion matrix, which displays the percentage of correctly predicted classes. The confusion matrix also depicts the false positive (FP) background prediction, not to be confused with the background class. This value represents the prediction of the „background“, which in this case is a positive prediction of an object, which does not belong to the presented classes of the model. The background class caused the most false positive (FP) „background“ predictions with 48%, which is to be expected when the actual background class is very similar to the YOLOv5 „background“ prediction. The 29% of the FP „background“ predictions were caused by the bird class, while the bird images in the sky can be easily mixed up with the sky as background, leading to FP prediction. The remaining 25% were caused by the drone images. Regarding the ability to correct class predictions, the accuracy of the trained model is close to being perfect. These values are depicted as the diagonal of the matrix. The accuracy of the bird and drone prediction was really high with an accuracy of 99%. The background class prediction was shy of the 99%, however with still a respectable 98%. The confusion regarding the presented classes occurred with the background and drone class in 1% of the situations.

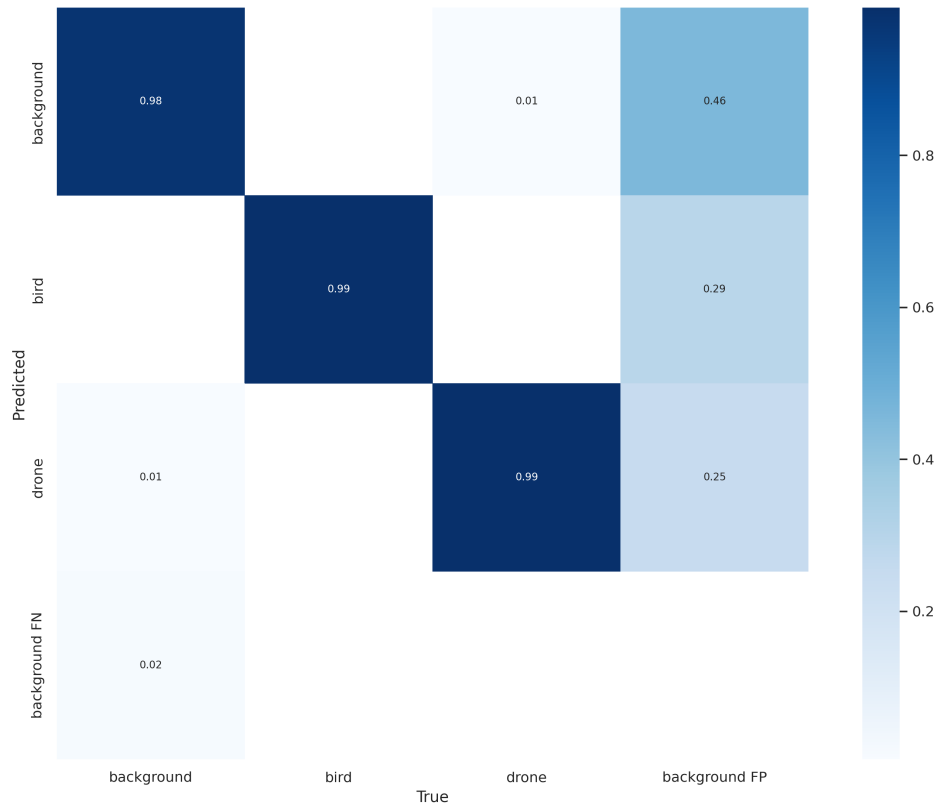


Figure 6.2: Confusion Matrix of Trained Model

6.4.2 Precision and Recall Correlation

As explained in numerical metrics (Section 6.2), with the increasing values of precision the recall values are lower. In the (Figure 6.3a), the precision values grow, with the increase of the confidence threshold. The peak precision among all classes is reached with the confidence threshold value of 0.972, meaning that 100% of the predictions were correct with the confidence value of 97.2%. According to this graph, the precision of bird prediction was rather high, compared to drone prediction, which was around 60% for the confidence values in the range [0.25, 0.92].

As for the recall of the model (Figure 6.3b). With the confidence value of 0, the recall peaks, resulting in many false negative predictions, since the value of precision is low, close to being zero.

The correlation of precision and recall among presented classes can be observed at (Figure 6.3c), where the x-axis represents the precision of the trained model, while the y axis represents the recall. These precision and recall values represent the model prediction with the overlap value of 50%.

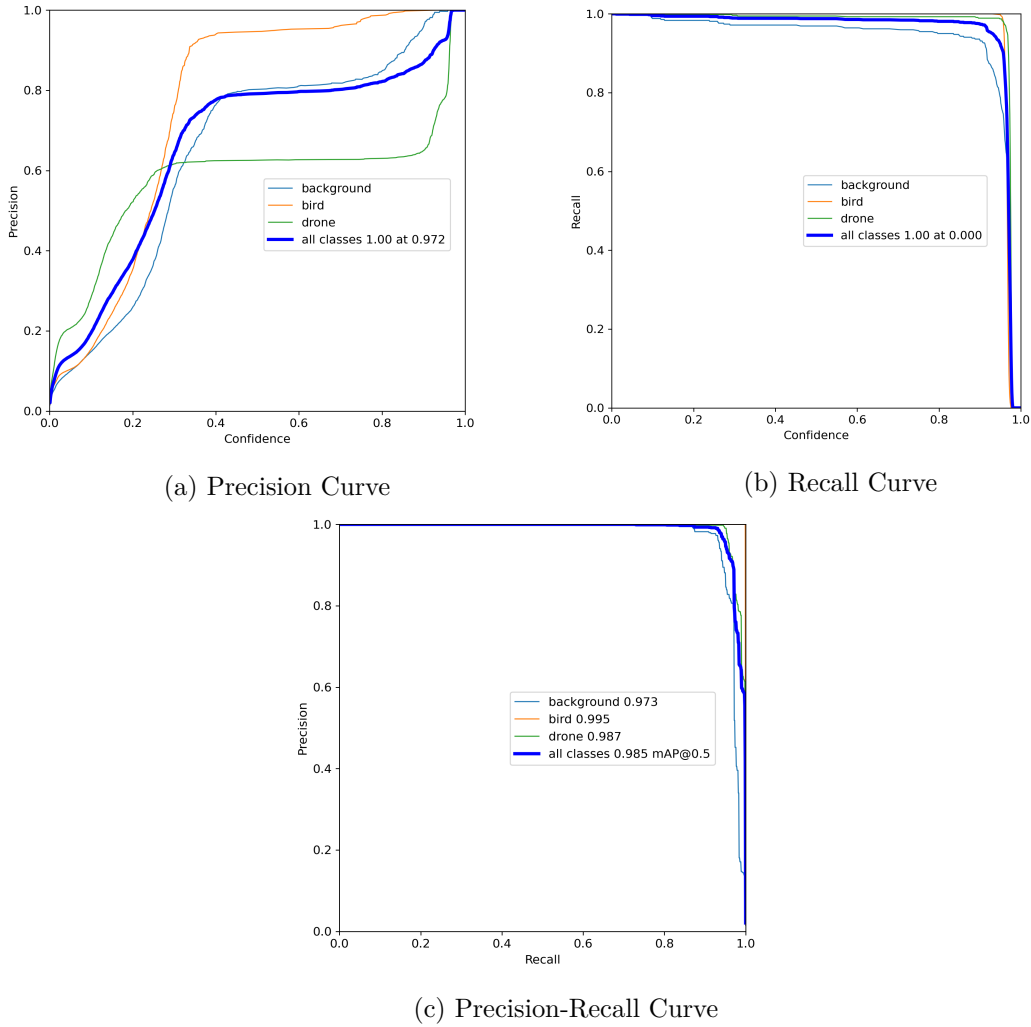


Figure 6.3: Depiction of the Evaluated Metrics

6.4.3 F1-Score analysis

Based on the (Figure 6.4) the confidence value responsible for the optimization of precision and recall among all classes is equal to 0.938, with the peak F1-Score value of 0.92. In most cases, the high F1-Scores with high confidence values are desired in order to evaluate the model as well-performing. Based on these observations, the quality of the model is quite high, since the performance of the model starts to suffer at fairly high confidence values.

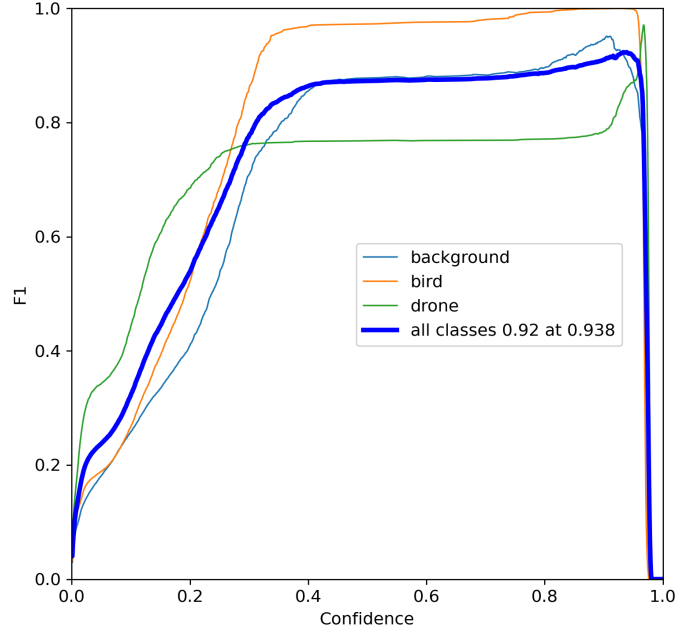


Figure 6.4: F1-Score Curve

6.4.4 Evaluation Based on the Number of Epochs

The development of the evaluation metrics throughout the training (Figure 6.5) depicts three metrics, precision, recall, and last but not least mAP for IoU of 50 and 95%. The precision values were first maximized at around 20 epochs of training however for recall to peak, it took close to 60 epochs. The mAP with 50% overlap stayed nearly the same, changing ever so slightly after peaking at 55 epochs. This wasn't the case with 95% overlap, where the results were not consistent throughout the whole training phase while peaking at around 0.35 mAP.

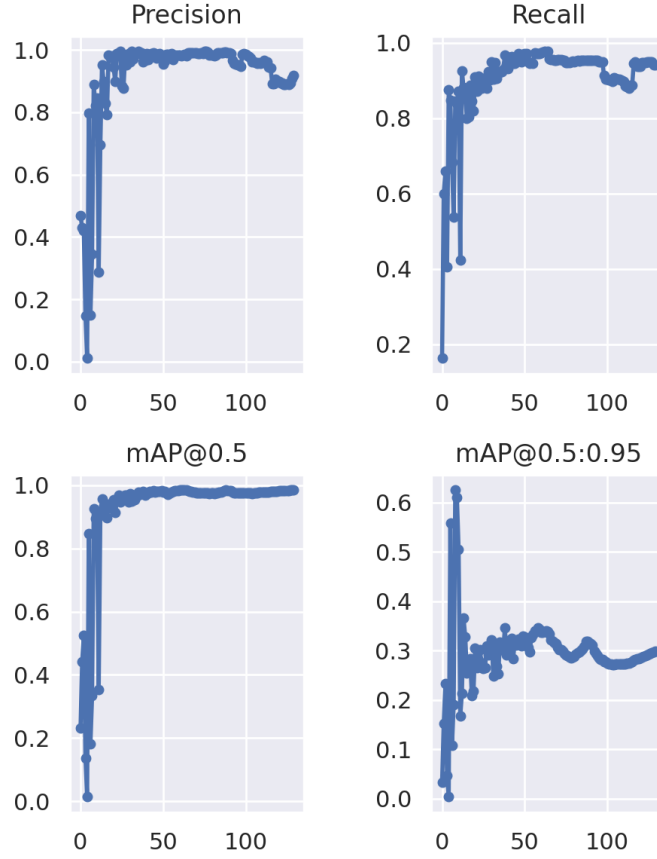


Figure 6.5: Training Results of the Model based on the Number of Epochs

6.5 Evaluation of Object Detection Module

This section concludes the evaluation of the object detection module utilizing the MOG2 background subtraction method onto the dataset. Since the second evaluation dataset wasn't created specifically for this approach of object detection, it is easier to depict the drawbacks of this approach.

6.5.1 Impact of Morphological Operations

Based on the performance observations of the module, the morphological operations, particularly the dilation with the kernel size of (25, 25) influence the computation time quite heavily. The application of the morphological operations onto subtracted foregrounds shown in the (Figure 6.6), depicts the importance of these morphological operations. The dilation provides filled images as a whole, without segmentation shown at (Figure 6.6a). The segmentation within the image leads to the inaccurate bounding of the detected object. The (Figure 6.6b) showcases the usage of erosion followed by dilation in order to produce a robust image for an accurate finding of contours, essential for the correct bounding of the object.

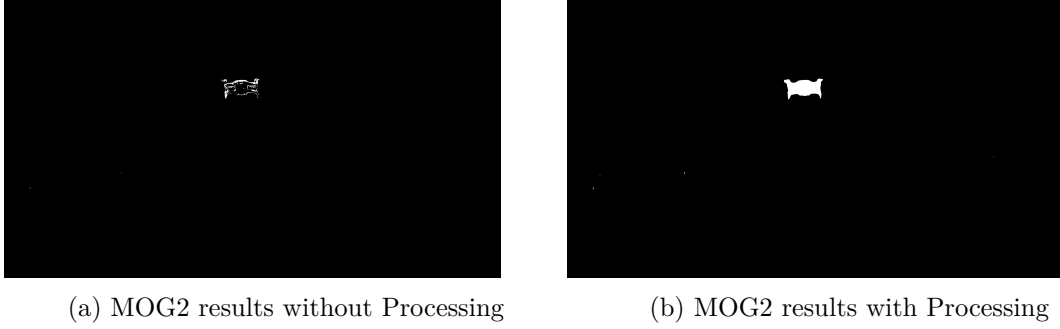


Figure 6.6: Application of Morphological operations

6.5.2 Frame Scale Influence on Detection

The influence on the detection of the objects, based on the resolution scale is quite small however observable. The higher the resolution, the more noise arises. The noise can be then bounded as a detected object, producing more objects for classification, thus additionally weighing down the computation of the model. With higher resolutions comes higher computation, since the number of pixels increases. This means that with a higher amount of pixels, more Gaussians are needed for the background subtraction. The combination of these hindrances slows down the computation quite a bit, making the down-scaled frames more appropriate for the application of the approach. The drawback of the down-scaled frames is the ability to recall objects further away from the camera.

6.5.3 Detection Module Drawbacks

As mentioned within the theory, the background subtraction method is used exclusively for static cameras, making the deployment of the model onto a moving camera impossible. This drawback was expected from the beginning however the issue based on the same principle occurred with the change of the lens focus, shown in (Figure 6.7). The displayed frame (Figure 6.7a) depicts the blurry frame, which occurred with the loss of the lens focus. The issue of the sudden change in the frame is interpreted by MOG2 as a movement of an object, thus false detecting the background as foreground (Figure 6.7b). In order to minimize the amount of noise, the kernel size used for erosion is increased, thus erasing the noise. However, with bigger erosion kernels, the likelihood of correctly detected object removal is quite high.

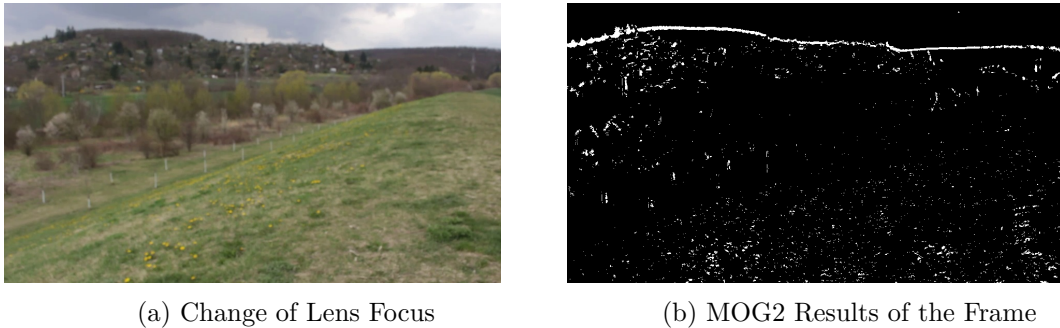


Figure 6.7: Focus Change Influence on the Detection Module

6.6 Evaluation of Object Classification Module

The evaluation of the classification module concludes the results of the deployment of the trained YOLOv5 model onto the outputs attained from the object detection module. The performance of the classification is closely related to the performance of the detection module since the number of outputs influences the total computation time of the model classification.

6.6.1 Dji Matrice 600 Pro classification

While evaluating the dataset depicting the DJI Matrice 600 Pro, which wasn't included in the training dataset of the model was used in order to test the coarseness of the model. The 1080p video available at `eval_dataset/untrained.mp4` was passed into the implemented script, applying the erosion kernel size of (3, 3), without any downscaling of the frame, with the plotting of classifications of minimal confidence of 50%. Based on the (Figure 6.8a) depicting the 172nd frame, the confidence of the classification for the unknown drone was rather low. The observed frame also contains a false-positive classification, one of which was caused by the moving clouds and the other was due to some sort of flying insect close to the camera lens. The false classification of clouds is caused by the insufficient amount of background images depicting the clouds. Since the images of birds were in clouds or up against a sky, the model classifies the moving clouds as birds. The flying insect is technically a false classification however the resemblance to a bird is within this frame debatable. The classification confidence of the drone within the first half of the video was around 55% with occasional alternation between background and drone classification. The confidence peaked around 70% however after a while the drone moved away from the camera quite a bit, making the model unable to classify the drone. The drone was therefore classified as a bird (Figure 6.8b), depicting the 289th frame.



Figure 6.8: Classification of `untrained.mp4`

6.6.2 Dji Mavic Air classification

Three out of the four produced recordings depicting the DJI Mavic Air drone were used in the creation of the dataset used for the model training. Leaving the last recording for evaluation. The recording is available at `eval_dataset/drone_1.mp4`. The recording was fed to the script in the same manner as in the previously mentioned evaluation. Since the drone captured within the frame was the same drone used for the model training, higher confidence classification values were expected. The classification values were much higher

with the DJI Mavic Air however the best results occurred with the drone being closer to the camera (Figure 6.9a). The further away the drone got, the worse confidence values along with false classifications. The model performed well with the drone not being distanced too far from the camera, producing consistent classifications with fairly decent classification values (Figure 6.9b). However, the long-distance detection was inconsistent and highly affected by the angle of the drone. The drone within the (Figure 6.9c) was classified as a drone with the 80% accuracy, although the same drone traversing the frame at a different angle with a similar distance from the camera got classified as a bird (Figure 6.9d) with the confidence value of 76%, which is rather high. The addition of more training images depicting drones at various angles would increase the accuracy and coarseness of the model.

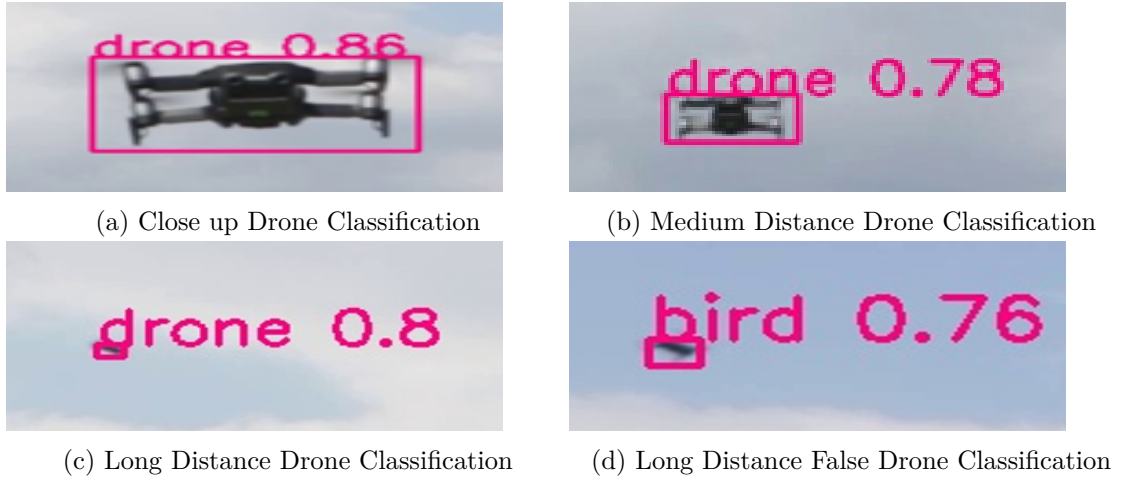


Figure 6.9: Classification of `drone_1.mp4`

6.6.3 Classification Module Drawbacks

The issues occurring with the classification were caused by the insufficient size of the training images. This caused false classifications, mentioned in previously mentioned evaluation scenarios. The issues based on these observations took place when the classification was challenged by the unlearned background type. In our case, this was caused by the strolling people within the frame (Figure 6.10a), which were incorrectly classified as a drone however the closer they got to the frame, the more accurate classification was (Figure 6.10b). In this case, the quality of the detection influences the further classification of the detected objects. Thus with insufficient detection of an object, comes inaccurate classifications. This is the case of the previously mentioned example, the smaller object is, the harder is to detect the object as a whole, passing the partial images for classification, producing inaccurate classifications.

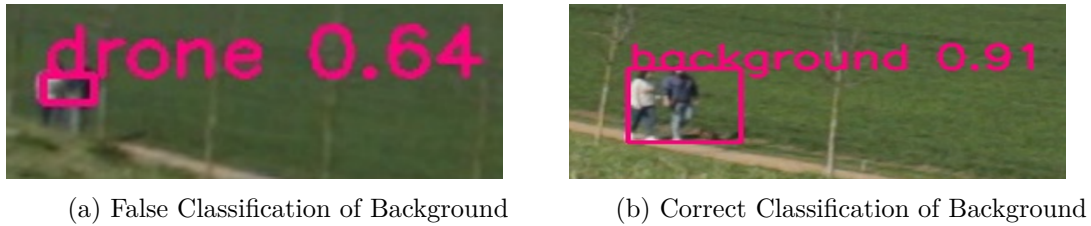


Figure 6.10: Classification Inaccuracy Based on Camera Distance

6.7 Performance of the Approach

The performance of the approach was initially tested on WSL2 using Ubuntu however the subsystem with the usage of CUDA gave skewed results. Based on this, the performance was evaluated using the Visual studio 22 PyTorch environment. While evaluating the performance of the approach, the used GPU was GTX 1060 3GB, while the CPU was the i5-7600K. Evaluation of performance was based on the previously mentioned video available at `eval_dataset/untrained.mp4`, since the length of the video is rather small, which cuts down the amount of time spent running the experiments. The evaluated video in 1080p depicts 30 frames per second with a length of 1 minute and 1 second. The performance evaluation using the `untrained.mp4` recording, summarized at (Table 6.1), shows the influence of the input arguments on the performance. Based on these results the usage of CPU performed better than GPU. This was caused by the complexity of the computation. Since this approach works with the smallest YOLOv5 model consisting of 232 layers, the CPU computes the classifications more efficiently than GPU. The resized frame made up 64% of the original frame, achieved by setting the (`--s`) argument to 0.8. The average amount of detections per frame decreased based on the erosion kernel size. With a higher kernel size being applied onto lower scaled frames, the number of detected objects is lower. This could cause a decrease in the false detections however this could also result in the removal of the observed object from within the frame. The experiments also showcase that the kernel size influences the runtime of the script. Since the amount of detected objects is lower, fewer classifications are performed, thus the runtime of the script also lowers.

Resolution	GPU/CPU	Kernel Size	Average FPS	Average Detections	Wall Time
1080x1920	CPU	(3,3)	16.47	0.56	2:56
864x1536	CPU	(3,3)	16.13	0.40	2:59
1080x1920	CPU	(1,1)	16.02	1.00	3:01
864x1536	CPU	(1,1)	15.81	0.89	3:00
1080x1920	GPU	(3,3)	14.21	0.56	3:21
864x1536	GPU	(3,3)	14.21	0.40	3:20
1080x1920	GPU	(1,1)	11.49	1.00	3:59
864x1536	GPU	(1,1)	12.18	0.89	3:46

Table 6.1: Performance Evaluation of the `untrained.mp4` recording

Chapter 7

Conclusion

This paper proposes the approach of drone detection and classification method, which can be applied to real-time camera streams or recordings. The approach utilizes the background subtraction method with the usage of Gaussian mixture models, while the classification is achieved using the CNN neural networks utilizing the YOLOv5 model architecture.

The implemented background subtraction method is the MOG2 algorithm available with OpenCV. With this approach, the OpenCV focused on CPU utilization in order to detect the foreground. The usage of OpenCV with CUDA could possibly provide better frames per second however the current implementation performs better using CPU, meaning that the user can achieve results with the system operating without a graphics card. The installation process of OpenCV with CUDA support is rather complicated. The performance of the detection module wasn't quite what was anticipated, but still sufficient and usable. The problems of the MOG2 occurred with the change of camera focus, producing many false detections, and slowing down the module heavily.

The performance of the used model was very high however the evaluation of the model doesn't take into consideration the distance between the camera and the detected object. The ability of classification took a hit with the drones of different colors, unable to detect them. This was caused by insufficient training since the training was done solely on the DJI Mavic Air in black color. The issues of classification occurred with unlearned drone angles, mistaking drones for birds.

7.1 Proposed Improvements

In order to improve on found drawbacks, the training dataset must be increased with various types of drones in different colors. With the larger amount of drone images, the background and bird images must be increased in a similar manner. This enlargement of the training dataset would provide better classification results. As for the actual improvement of implementation, the passing of the object detection module to the object classification module could be done in a more efficient manner. The proposed background subtraction could have been swapped with the optical flow approach however the usage of OpenCV with CUDA would be necessary in order to provide results applicable for real-time drone detection and classification.

Bibliography

- [1] BBC. *Drone collides with commercial aeroplane in Canada* [online]. 2017 [cit. 2021-12-28]. Available at: <https://www.bbc.com/news/technology-41635518>.
- [2] CARRANZA GARCÍA, M., TORRES MATEO, J., LARA BENÍTEZ, P. and GARCÍA GUTIÉRREZ, J. On the Performance of One-Stage and Two-Stage Object Detectors in Autonomous Vehicles Using Camera Data. *Remote Sensing*. 2021, vol. 13, no. 1. DOI: 10.3390/rs13010089. ISSN 2072-4292. Available at: <https://www.mdpi.com/2072-4292/13/1/89>.
- [3] CHRISLB. *ArtificialNeuronModel english.pngn*. 2005. Available at: https://upload.wikimedia.org/wikipedia/commons/6/60/ArtificialNeuronModel_english.png.
- [4] ELHABIAN, S., EL SAYED, K. M. and AHMED, S. H. Moving Object Detection in Spatial Domain using Background Removal Techniques - State-of-Art. *Recent Patents on Computer Science*. 2008, vol. 1, p. 32–54.
- [5] EZUMA, M., ERDEN, F., ANJINAPPA, C. K., OZDEMIR, O. and GUVENC, I. *Micro-UAV Detection and Classification from RF Fingerprints Using Machine Learning Techniques*. 2019.
- [6] FARNEBÄCK, G. Two-Frame Motion Estimation Based on Polynomial Expansion. In: JOSEF, B. and TOMAS, G., ed. *Image Analysis*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2003, p. 363–370. ISBN 978-3-540-45103-7.
- [7] GANTI, S. R. and KIM, Y. Implementation of detection and tracking mechanism for small UAS. In: *2016 International Conference on Unmanned Aircraft Systems (ICUAS)*. 2016, p. 1254–1260. DOI: 10.1109/ICUAS.2016.7502513.
- [8] GIRSHICK, R., DONAHUE, J., DARRELL, T. and MALIK, J. Rich Feature Hierarchies for Accurate Object Detection and Semantic Segmentation. In: *2014 IEEE Conference on Computer Vision and Pattern Recognition*. 2014, p. 580–587. DOI: 10.1109/CVPR.2014.81.
- [9] GIRSHICK, R. B. Fast R-CNN. *CoRR*. 2015, abs/1504.08083. Available at: <http://arxiv.org/abs/1504.08083>.
- [10] GOLDSCHMIDT, P. *Mitigation of DoS Attacks Using Machine Learning*. September 2021. Master's thesis. Faculty of Information Technology, Brno University of Technology. Available at: <https://dspace.vutbr.cz/bitstream/handle/11012/201270/final-thesis.pdf>.

- [11] JOHN MCGONAGLE, A. D. *Gaussian Mixture Model*. Available at: <https://brilliant.org/wiki/gaussian-mixture-model/?fbclid=IwAR2c0cF5qY6DuVRiLzIKtSiBT2D0qS0xGNMdbf6SFGij6L7vHE3jWy931Ic#the-model>.
- [12] LIU, L., OUYANG, W., WANG, X., FIEGUTH, P. W., CHEN, J. et al. Deep Learning for Generic Object Detection: A Survey. *CoRR*. 2018, abs/1809.02165. Available at: <http://arxiv.org/abs/1809.02165>.
- [13] MARCOMINI, L. A. and CUNHA, A. L. A Comparison between Background Modelling Methods for Vehicle Segmentation in Highway Traffic Videos. *CoRR*. 2018, abs/1810.02835. Available at: <http://arxiv.org/abs/1810.02835>.
- [14] NEPAL, U. and ESLAMIAT, H. Comparing YOLOv3, YOLOv4 and YOLOv5 for Autonomous Landing Spot Detection in Faulty UAVs. *Sensors*. 2022, vol. 22, no. 2. DOI: 10.3390/s22020464. ISSN 1424-8220. Available at: <https://www.mdpi.com/1424-8220/22/2/464>.
- [15] NOTJIM. *Neuron - annotated.svg*. 2008. Available at: https://upload.wikimedia.org/wikipedia/commons/5/52/Neuron_-_annotated.svg.
- [16] *Optical flow*. 2016. Available at: https://opencv24-python-tutorials.readthedocs.io/en/latest/py_tutorials/py_video/py_lucas_kanade/py_lucas_kanade.html.
- [17] PORTLAND, F. . *Portland chaos: 22 face federal charges over weekend violence* [online]. 2020 [cit. 2021-12-28]. Available at: <https://www.foxnews.com/us/portland-protests-federal-charges-weekend-violence>.
- [18] REN, S., HE, K., GIRSHICK, R. B. and SUN, J. Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks. *CoRR*. 2015, abs/1506.01497. Available at: <http://arxiv.org/abs/1506.01497>.
- [19] SAMARAS, S., DIAMANTIDOU, E., ATALOGLOU, D., SAKELLARIOU, N., VAFEIADIS, A. et al. Deep Learning on Multi Sensor Data for Counter UAV Applications—A Systematic Review. *Sensors*. 2019, vol. 19, no. 22. DOI: 10.3390/s19224837. ISSN 1424-8220. Available at: <https://www.mdpi.com/1424-8220/19/22/4837>.
- [20] SHAIKH, HOSSAIN, S., SAEED, KHALID, CHAKI et al. Moving Object Detection Using Background Subtraction. In: *Moving Object Detection Using Background Subtraction*. Cham: Springer International Publishing, 2014, p. 15–23. ISBN 978-3-319-07386-6. Available at: https://doi.org/10.1007/978-3-319-07386-6_3.
- [21] SINGH, S. A., MEITEI, T. G. and MAJUMDER, S. 6 - Short PCG classification based on deep learning. In: AGARWAL, B., BALAS, V. E., JAIN, L. C., POONIA, R. C. and MANISHA, ed. *Deep Learning Techniques for Biomedical and Health Informatics*. Academic Press, 2020, p. 141–164. DOI: <https://doi.org/10.1016/B978-0-12-819061-6.00006-9>. ISBN 978-0-12-819061-6. Available at: <https://www.sciencedirect.com/science/article/pii/B9780128190616000069>.
- [22] SINGH, S. A., MEITEI, T. G. and MAJUMDER, S. 6 - Short PCG classification based on deep learning. In: AGARWAL, B., BALAS, V. E., JAIN, L. C., POONIA, R. C.

- and MANISHA, ed. *Deep Learning Techniques for Biomedical and Health Informatics*. Academic Press, 2020, p. 141–164. DOI: <https://doi.org/10.1016/B978-0-12-819061-6.00006-9>. ISBN 978-0-12-819061-6. Available at: <https://www.sciencedirect.com/science/article/pii/B9780128190616000069>.
- [23] T, S., R, S., R, V. and T, K. Flying Object Detection and Classification using Deep Neural Networks. *International Journal of Advanced Engineering Research and Science*. january 2019, vol. 6, p. 180–183. DOI: 10.22161/ijaers.6.2.23.
- [24] TAHA, B. and SHOUFAN, A. Machine Learning-Based Drone Detection and Classification: State-of-the-Art in Research. *IEEE Access*. 2019, vol. 7, p. 138669–138682. DOI: 10.1109/ACCESS.2019.2942944.
- [25] TAHA, B. and SHOUFAN, A. Machine Learning-Based Drone Detection and Classification: State-of-the-Art in Research. *IEEE Access*. 2019, vol. 7, p. 138669–138682. DOI: 10.1109/ACCESS.2019.2942944.
- [26] UNLU, E., ZENOU, E., RIVIERE, N. and DUPOUY, P.-E. Deep learning-based strategies for the detection and tracking of drones using several cameras. *IPSI Transactions on Computer Vision and Applications*. Jul 2019, vol. 11, no. 1, p. 7. DOI: 10.1186/s41074-019-0059-x. ISSN 1882-6695. Available at: <https://doi.org/10.1186/s41074-019-0059-x>.
- [27] WEISSTEIN and W., E. *Hyperbolic Tangent*. 1999. Available at: <https://mathworld.wolfram.com/SigmoidFunction.html>.
- [28] WEISSTEIN and W., E. *Hyperbolic Tangent*. 1999. Available at: <https://mathworld.wolfram.com/HyperbolicTangent.html>.
- [29] YANG, M. A moving objects detection algorithm in video sequence. In: *2014 International Conference on Audio, Language and Image Processing*. 2014, p. 410–413. DOI: 10.1109/ICALIP.2014.7009826.
- [30] YU, L., CHEN, X. and ZHOU, S. Research of Image Main Objects Detection Algorithm Based on Deep Learning. In: *2018 IEEE 3rd International Conference on Image, Vision and Computing (ICIVC)*. 2018, p. 70–75. DOI: 10.1109/ICIVC.2018.8492803.
- [31] ZHAN, C., DUAN, X., XU, S., SONG, Z. and LUO, M. An Improved Moving Object Detection Algorithm Based on Frame Difference and Edge Detection. In: *Fourth International Conference on Image and Graphics (ICIG 2007)*. 2007, p. 519–523. DOI: 10.1109/ICIG.2007.153.
- [32] ZIVKOVIC, Z. Improved Adaptive Gaussian Mixture Model for Background Subtraction. In: September 2004, vol. 2, p. 28 – 31 Vol.2. DOI: 10.1109/ICPR.2004.1333992. ISBN 0-7695-2128-2.
- [33] ZIVKOVIC, Z. and VAN DER HEIJDEN, F. Efficient adaptive density estimation per image pixel for the task of background subtraction. *Pattern Recognition Letters*. 2006, vol. 27, no. 7, p. 773–780. DOI: <https://doi.org/10.1016/j.patrec.2005.11.005>. ISSN 0167-8655. Available at: <https://www.sciencedirect.com/science/article/pii/S0167865505003521>.