UKRAINIAN CATHOLIC UNIVERSITY

BACHELOR THESIS

Development of a Machine Learning method for Detecting objects of military equipment on drone footage

Author:
Danyil HUTSUL

Supervisor:
Oles DOBOSEVYCH

A thesis submitted in fulfillment of the requirements for the degree of Bachelor of Science

in the

Department of Computer Sciences and Information Technologies
Faculty of Applied Sciences

Lviv 2023

Declaration of Authorship

I, Danyil HUTSUL, declare that this thesis titled, "Development of a Machine Learning method for Detecting objects of military equipment on drone footage" and the work presented in it are my own. I confirm that:

- This work was done wholly or mainly while in candidature for a research degree at this University.
- Where any part of this thesis has previously been submitted for a degree or any other qualification at this University or any other institution, this has been clearly stated.
- Where I have consulted the published work of others, this is always clearly attributed.
- Where I have quoted from the work of others, the source is always given. With the exception of such quotations, this thesis is entirely my own work.
- I have acknowledged all main sources of help.
- Where the thesis is based on work done by myself jointly with others, I have made clear exactly what was done by others and what I have contributed myself.

Signed: _____

Date: _____

UKRAINIAN CATHOLIC UNIVERSITY

Faculty of Applied Sciences

Bachelor of Science

Development of a Machine Learning method for Detecting objects of military equipment on drone footage

by Danyil HUTSUL

Abstract

Visual object detection is one of the most common research topics in the sphere of computer vision research. It has a vast range of areas of application, from medical to automotive. One such area is the detection of objects from videos and images taken by unmanned aerial vehicles for both military and civilian purposes. With the start of the latest phase of the Russo-Ukranian war, a great number of videos, taken both military-grade and repurposed civilian drones, have begun appearing all over the internet. In this paper, we collect a number of such videos to create a dataset and train several object detection models with the goal of finding one best suited for the task. The code used in this paper is available on GitHub.

Acknowledgements

I would like to thank my supervisor Oles Dobosevych for his help in this research project. I also want to thank the Ukrainian Catholic University, the Faculty of Applied Sciences, and the Armed Forces of Ukraine.

Contents

List of Figures

List of Tables

List of Abbreviations

UAV	**U**nmanned **A**erial **V**ehicle
CVAT	**C**omputer **V**ision **A**nnotation **T**ool
CNN	**C**onvolutional **N**eural **N**etwork
R-CNN	**R**egion Based **C**onvolutional **N**eural **N**etwork
YOLO	**Y**ou **O**nly **L**ook **O**nce

Chapter 1

Introduction

In recent years, commercially available options for unmanned aerial vehicles (commonly named drones) have increased in popularity due to both lowering prices and increasing ease of use. Commercial drones are currently used in surveying, mapping, photography, entertainment, and other areas. However, one surprising avenue of use for such drones is in the military. Different militaries have employed unmanned combat aerial vehicles (UCAVs) since the start of the 21st century for both scouting and strike operations. However, they have a steep initial investment cost and require extensive training and repairs to be used effectively. However, in the most recent stage of the Russo-Ukranian war, both sides began to utilize commercial drones as a cheap replacement for the previously mentioned UCAVs. They have several advantages: with most important being their price and difficulty of detection by enemy forces.

Most of the existing drone datasets focus on annotating cars and pedestrians. However, no such dataset has been created for military vehicles at the time of the writing. As such, our goal was to create such a dataset by using aerial war footage and find the best method for working with this dataset.

1.1 Thesis structure

This paper is organized in the following way. We discuss related dataset works in Chapter 2. In Chapter 3, we outline the process of creating the dataset and its format. Chapter 4 contains an overview of chosen methods for object detection and their metrics. In Chapter 5, we showcase our experiments' results and summarize our findings in Chapter 6.

Chapter 2

Related Works

2.1 Datasets

2.1.1 VisDrone

Detection and Tracking Meet Drones Challenge[25] is a large-scale benchmark for a number of different computer-vision tasks. The benchmark dataset consists of 288 video clips formed by 261908 frames and 10209 images captured using drone-mounted cameras. The dataset covers a wide range of locations, environments, objects, density, and weather and lighting conditions. The dataset features manually annotated bounding boxes and additional contextual labels for each object, like occlusion and visibility. We've ended up using a similar type of object labeling in our dataset.

2.1.2 CARPK

Drone-based Object Counting by Spatially Regularized Regional Proposal Network[10] is a large dataset containing over 90000 cars captured from different parking lots using a drone. Each car is annotated as a bounding box.

2.1.3 TrajNet

An Evaluation of Trajectory Prediction Approaches and Notes on the TrajNet Benchmark[1] is a large forecasting benchmark containing 11448 trajectories of different moving agents. The data we collected mostly featured stationary objects, as such trajectory wasn't tracked in the final result.

2.2 Object detection

Object detection is a computer vision task in which the goal is to detect an locate objects of interest in images or videos. The task involves finding the positions and bounds of objects and classifying them.

Currently, there are two main types of Object detection methods:

- One-stage - bounding boxes and their classes are predicted using a single network pass.
- Two-stage - model proposes a set of regions of interest, then for each proposal, bounding boxes and their classes are predicted

Models that use two-stage detectors usually have higher accuracy than one-stage models but perform much slower. For our experiments, we've decided to use both types of models.

2.3 Models

2.3.1 Faster R-CNN

R-CNN[6] family of models utilize a two-stage approach to object detection. First published in 2013, R-CNN's main contribution was the introduction of a Convolutional Neural Network for feature extraction. The model works by first creating region proposals - regions of original image that have the highest likelihood of containing detectable objects. The model uses a Selective Search[22] method, though other approaches can be used as well. It processes each region with a CNN model which is then used passed to Support Vector Machine for classification.

Faster R-CNN[19] is an improvement upon its predecessor, Fast-R-CNN[5]. Fast-R-CNN improved R-CNN by using a shared layer for the whole image and regions instead of processing each region independently. Faster R-CNN further improves on it by using a region proposal network instead of selective search to increase the speed of the model.

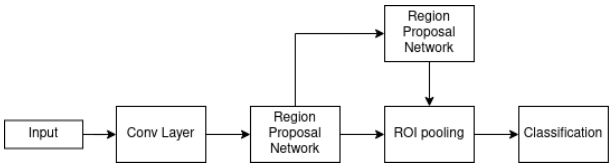


FIGURE 2.1: Schematic diagram of Faster R-CNN

2.3.2 YOLO

YOLO is a family of one-stage object detection models. The model works by first splitting the input image into an $S \times S$ grid[18]. Each grid cell is then used to predict B bounding boxes, their confidence and C class probabilities which is then encoded in a $S \times S \times (B * 5 + C)$ tensor Figure 2.4[18].

YOLOv5

YOLOv5[12] is a YOLO family object detection model released in 2020. It is built on the base of YOLOv4[3], utilizing CSP-Darknet53[3] for the backbone. It improves on its predecessor in a few ways by replacing the first three layers with a single Focus layer and replacing the SPP[9] structure with SPPF[12] - a more efficient version of SPP.

YOLOR

YOLOR[24], [23] expands on previous iterations by adding Explicit and Implicit knowledge to the model. Implicit knowledge, i.e. knowledge that does not require an input, is modeled using a trainable vector representation, while the neural

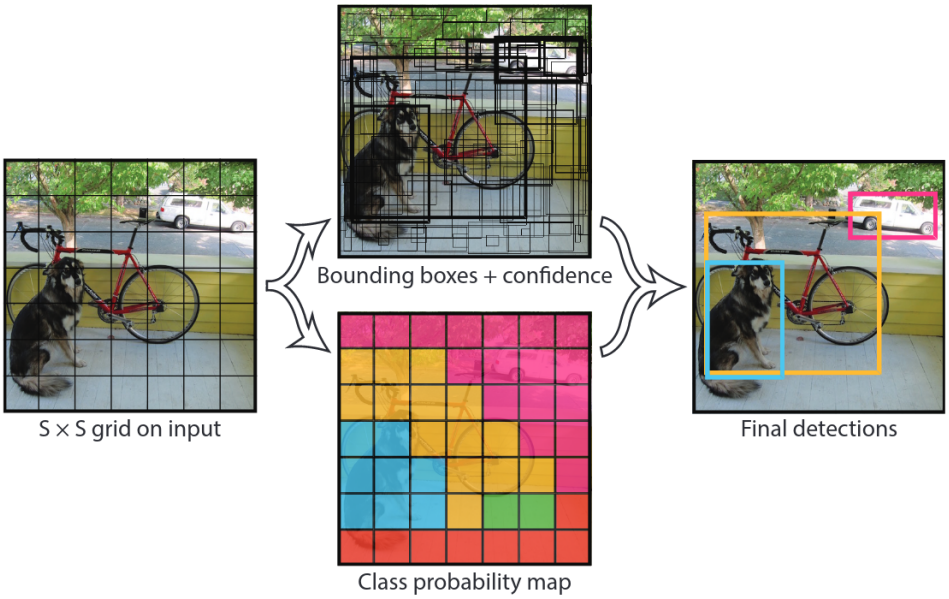


FIGURE 2.2: Example of YOLO model

network handles explicit knowledge. Like YOLOv5[12], YOLOR[24], [23] utilizes YOLOv4[3] as its base model.

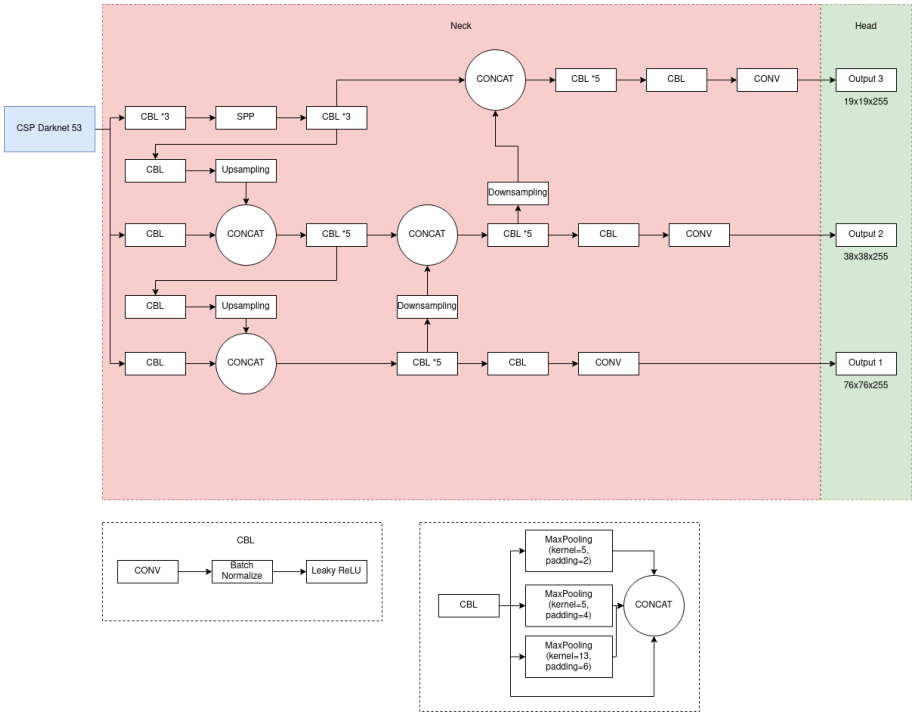


FIGURE 2.3: YOLOv4 model architecture, used as base for YOLOv5 and YOLOR. Utilizes a CSPDarknet53 Backbone, Spatial Pyramid Pooling + Path Aggregation Network neck and YOLOv3 Head

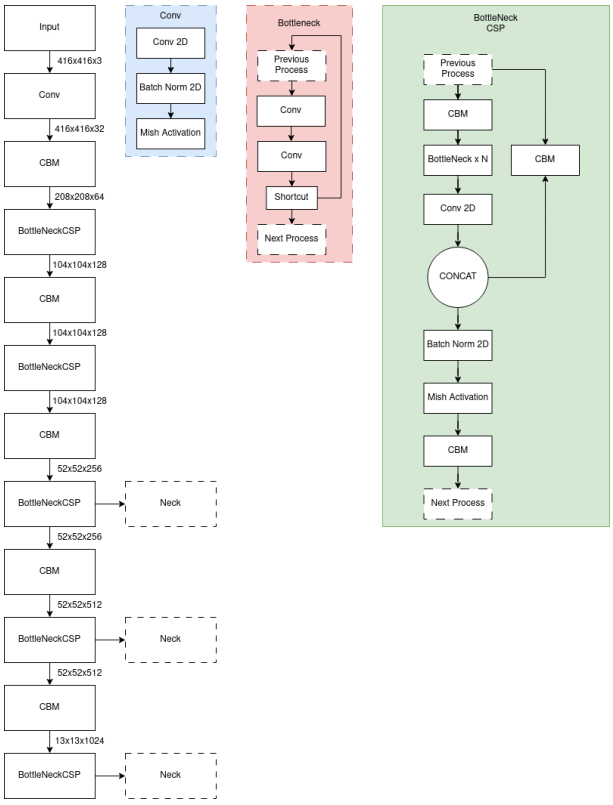


FIGURE 2.4: CSP Darknet 53, used as Backbone in YOLOv4 model

Chapter 3

Dataset Overview

3.1 Data Collection

The dataset was composed of videos collected from two social media sites: Telegram and Twitter. At the beginning stages of the work, Facebook was also considered as another potential source of data. However, due to difficulties with searching for and downloading videos, it was ultimately decided to only use the two previously mentioned platforms. Example frames are shown in Figure 3.1.

FIGURE 3.1: Example frames of videos from the dataset

3.1.1 General Workflow

Computer Vision Annotation Tool was used to create the dataset - a free, open-source image and video annotation tool. The creators of CVAT[4] provide a publicly accessible instance of their tool with limited storage space, but we have decided to use a private install of the tool. A Google Cloud Platform Virtual Machine was used to host the tool.

CVAT[4] uses tasks to organize its workflow. A task is a collection of images or video frames. A project is a collection of tasks containing annotation parameters, such as label classes.

The newest available version of CVAT[4] at that time provided rudimentary support for working with tasks in the form of a Python-based Command Line Interface.

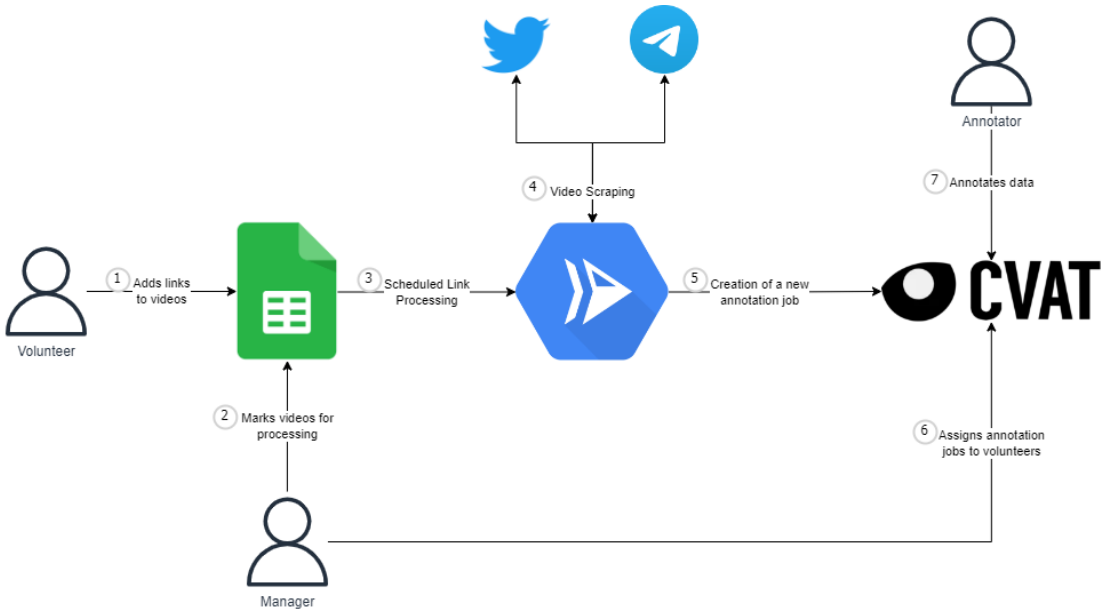


FIGURE 3.2: Diagram of workflow

URLs	Status	Confirmation
https://t.me/priva	Processed	Confirmed
https://t.me/usinf	Processed	Confirmed
https://t.me/usinf	Processed	Confirmed

FIGURE 3.3: Example spreadsheet

An API would not be added to CVAT until after the finalization of the dataset. The main feature of CLI that we have ended up using is the creation of new tasks. Content for the task could be uploaded to CVAT[4] from local storage or from a URL. We have decided to use the second option to simplify the workflow. Unfortunately, CLI lacked a major function - access to the created tasks could not be specified and would always default to the account of the uploader. This complicated the workflow as annotation was done by third-party volunteers with accounts that lacked admin privileges. Unfortunately, no workaround was found, and the distribution of tasks between volunteer accounts had to be done manually.

3.1.2 Overview of Video Collection Process

A spreadsheet was created and hosted using Google Drive to manage the creation of new tasks. The spreadsheet would contain links to social media posts and whether the link was processed, rejected, or waiting to be uploaded. A Google Cloud Service account was also created and given editor privileges to the spreadsheet. We have decided to automate the creation of new tasks using Google Cloud Run[7]. Google Cloud Run[7] allows users to run container instances, either as a service (continually available code that responds to web requests) or as a job (scheduled code that runs and quits after performing its function). We created a Python script that would use the previously mentioned Service account to access the spreadsheet, choose links that were marked to be processed, and extract the source of the video to be added to the CVAT[4] project. The script could be launched at any time by sending a GET request to the Cloud Run instance. Additionally, every 24 the script would be launched automatically using a scheduled Google Cloud Run Job

3.1.3 Twitter Video Collection

During research we've found two approaches for collecting videos from the Twitter: Twitter API+Tweepy[8] and Twint.

Twitter API+Tweepy

Twitter provides users with the ability to access their API using a developer account. To gain access, a user must submit an application detailing the use case of Twitter API. Tweepy[8] is a Python library that provides a range of functions, some requiring API access. As we did not need access to advanced tools, we ended up using this Python library in our script as it provided an easy way to fetch video sources. The script would extract the ID of a tweet from the provided URL and return the highest quality video source along with other miscellaneous information like date of upload, number of views, and URL of the poster.

Pros	Cons
Plentiful documentation	Advanced features require a developer account
Wide range of functions	Rate limitations

TABLE 3.1: Pros and Cons of Twitter API

Twint

Twint[21] is a tool designed for data collection from Twitter without using official API. It has no rate limitations and does not require a developer account. Unfortunately, it lacks many of the advanced features of official API.

Pros	Cons
Ease of use	Lack of advanced features
No rate limit	Lack of documentation

TABLE 3.2: Pros and Cons of Twint

3.1.4 Telegram Video Collection

For collecting videos from Telegram, we have settled on using Beautiful Soup[20] - a Python package for parsing HTML and XML documents. A shared message from an open Telegram channel could be viewed by unauthorized users as a separate web page. This webpage would always retain the same HTML structure. As such, it was easy to set up a consistent way to collect videos. The script would access the least abstracted version of the web page and look for any <video> HTML elements. It would then extract and return their source, along with other web page information, like date of upload, number of views, and URL of the poster.

3.2 Data annotation

3.2.1 Annotation Difficulties

The main difficulty when creating an image dataset is the process of annotation. Each image requires a human volunteer to find and mark all required objects. The effectiveness of an image dataset grows with its size. For example, the YOLOv4[2] documentation recommends at least 2000 images per class. As such, even a simple model that strives to detect three different classes of objects will need 6000 images minimum for training, with results being improved by having images of objects with different scales, rotations, lighting, sides, and backgrounds. Annotating this many distinct images requires a lot of man-hours, even without additional time spent verifying annotations' correctness.

Thankfully annotation can be made easier when working with videos as opposed to distinct images. Instead of annotating each frame as a separate image, we can arrange them in the order of appearance, annotate two frames and linearly interpolate the annotation in between them. The annotator can then go back and fix any mistakes in the intermediate frames. This works best when either the camera or the filmed object is stationary. Considering that most videos that we've collected feature either UAV footage taken of disabled/non-moving vehicles or moving vehicles taken by mostly stationary UAVs, this is the method that we've used during annotation.

FIGURE 3.4: Annotator manually positions bounding boxes in the start and end frames. All frames between the two are linearly interpolated

Another problem that we have run into during annotation is the image quality. This manifested in a few different ways:

- Video Compression. The videos were not downloaded from the primary source but instead from social media posts, so they had undergone compression. This was not a problem for most of the videos we've annotated, but some did contain objects that were hard to accurately identify.
- High altitude/zoom of the UAV. This usually was not a problem in most cases, but in combination with the previous point, objects could blend in with their surroundings.

One solution was to omit annotating hard-to-see objects entirely, but we have decided to create a separate class for them instead.

Video Compression

High UAV altitude or zoom

FIGURE 3.5: Example of image quality issues. Objects in these frames were labeled as Unidentified Moving Objects

3.2.2 Object Class and Tags

During the annotation of the dataset, each object was assigned one of the following classes:

- Vehicle
- Soldier
- Unidentified Moving Object

The Unidentified Moving Object class encompassed two other classes and was used if the object's total area was too small, caused either due to video compression, high altitude of the UAV, or high zoom. Additionally, if more than half of the total area of the object was obscured by foliage, debris, or smoke, it would be given an "Obscured" tag. However, we ended up not using this tag in training the models.

3.2.3 Object representation

An annotated object's location in the image can be represented in multiple different ways. For our dataset, the two approaches that fit best with the type of objects we were working with were Bounding Boxes and Polygons.

Bounding boxes are rectangles that overlay the detected object. They can be represented in multiple ways:

- Two opposing corner points of rectangle (x_1, y_1, x_2, y_2) (Pascal VOC[11])
- A center point of the rectangle and its width and height (x_1, y_1, w, h) (YOLO[18])
- A single point of the rectangle and its width and height (x_1, y_1, w, h) (COCO[13])

Additionally, a bounding box can have an additional parameter to represent its rotation angle.

The main benefit of the bounding box is the ease of annotation - the annotator only needs to mark two points of the bounding box. The required work is increased a bit if a bounding box can have a rotation box. Another benefit is the ease of implementation of the model that utilizes a bounding box. The main problem of the bounding box becomes apparent when marking a non-rectangular object as space inside the bounding box is "wasted."

Pros	Cons
Fast to annotate	Does not represent objects with complex shapes well
Easier to predict	

TABLE 3.3: Pros and Cons of Bounding Boxes

Polygon, as opposed to a bounding box, can have an arbitrary number of points. This allows for accurate annotation of complex objects but significantly increases the difficulty of both annotation and detection.

Pros	Cons
Accurately represents objects with complex shapes	Hard to annotate
	Harder to predict

TABLE 3.4: Pros and Cons of Polygons

For this dataset, we decided to utilize bounding boxes. This was done due to the rectangular shape of the vehicles and to simplify the annotation process. Figure 3.6 We exported the dataset in COCO format first and later converted it to YOLO format for further use in training. We've also ended up not using rotation to represent objects to speed up both annotation and training. Each bounding box was positioned in such a way as to contain the main body of the object. If the object was partially obscured, either by buildings or trees, the area of the bounding box was estimated using contextual information. Examples of annotated frames are shown in Figure 3.7.

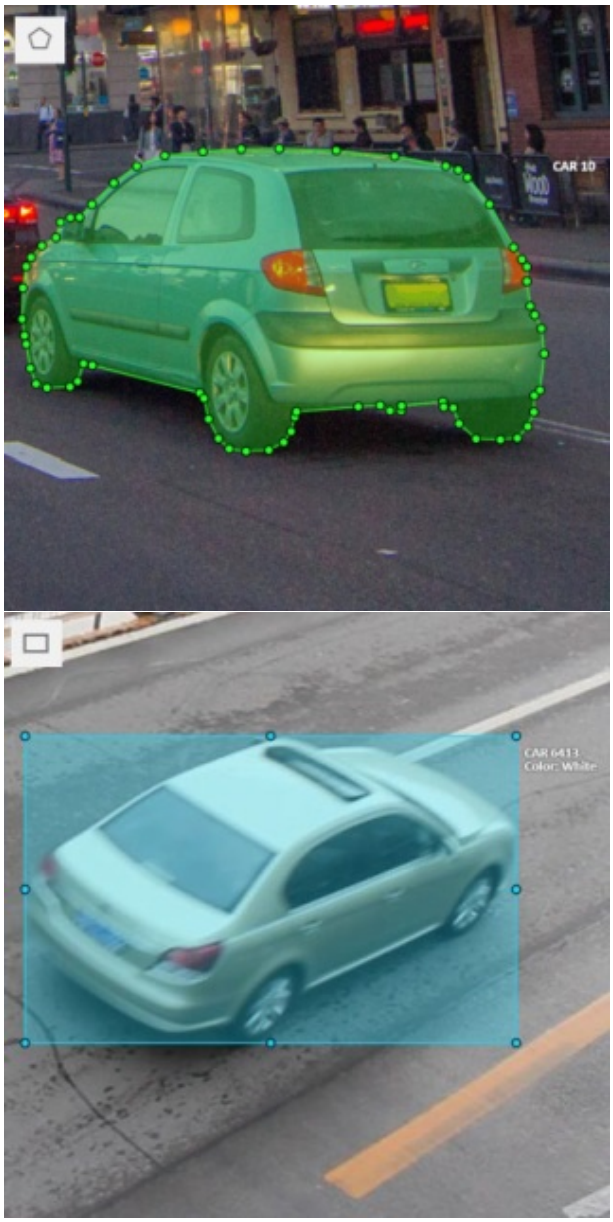


FIGURE 3.6: Comparison of Annotation shapes in CVAT interface

3.2.4 Collected Data

In total, the dataset contains 120 different videos, with 86360 labeled frames and 124691 bounding boxes. All frames are saved in JPEG format and are named per the video and frame number (e.g., video_109–frame0000062.jpg). The dataset is split into test/validation/train subsets with 80641/13163/13163 frames.

FIGURE 3.7: Example of annotated frame

Chapter 4

Experiments Overview

4.1 Experiments Setup

All models were trained using an Nvidia Tesla T4 GPU using Google Cloud VM. For Faster R-CNN[19], we've used Pytorch[16] implementation of the models and the official paper source code for YOLOR[23] and YOLOv5[12]. We've used transfer learning using pre-trained COCO 2017[13] weights to speed up the training process. The training was done on the previously discussed dataset, with 80640 images. Per the requirements in YOLO[17] images were resized to squares. Final dimensions used were 640x640 px.

4.2 Performance Metrics

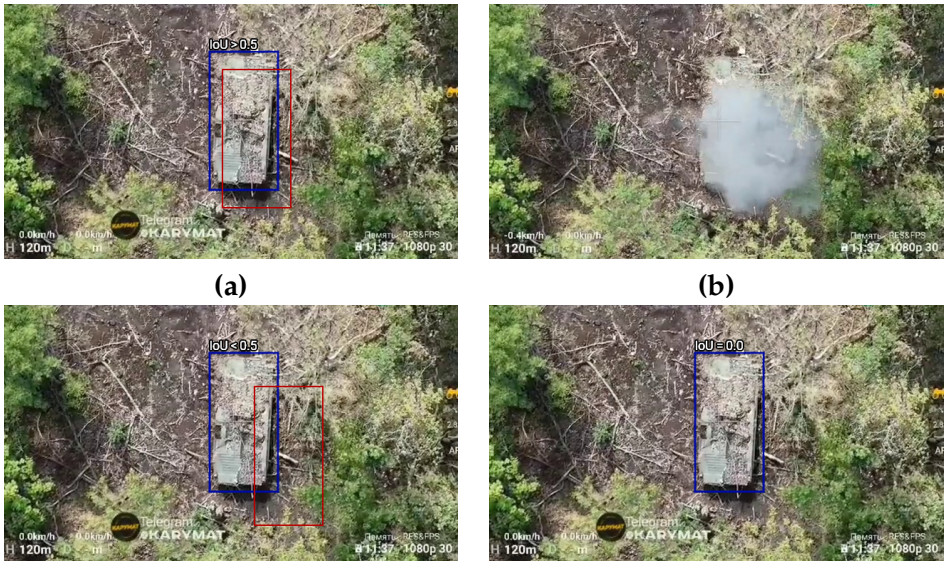
4.2.1 Mean Average Precision

Mean Average Precision[15], [14] is a measure of how close the predicted are is to the ground truth. It is calculated by taking a mean of each class' Average Precision. Per-class AP is calculated by the area under the precision/recall curve of predictions.

The first step in calculating the AP of a class is to map each detection to its most-overlapping ground truth. This is done by calculating Intersection over Union for each detection-ground truth pair. An IoU threshold is used to filter the detections; those that do not pass it are discarded and counted as False Positives. The highest-scored detection is counted as a True Positive; all others are instead counted as False Positives. Next, Precision and Recall are calculated for each class. They are defined as a ratio of the number of True Positives to the number of Total Predictions and Total Ground Truths, respectively. Afterward, Precision/Recall values are plotted, and the area under the precision-recall curve is calculated.

4.2.2 Frames per Second

FPS is a common measure of the model's detection speed. It denotes the number of inputs a model can process in a second. This is useful to measure the real-time effectiveness of a model.



(a) **(b)**

(c) **(d)**

FIGURE 4.1: Confusion Matrix (Blue bounding box - Ground Truth, Red - prediction):
(a) TP - Detected correctly
(b) TN - Correct background detection
(c) FP - Wrong Bounding Box or Label
(d) FN - Wrong background detection

Chapter 5

Results

All models were tested on an Nvidia Tesla T4 GPU with a threshold of $IoU = 0.5$. In total, 13162 images were used for testing, all scaled to 640x640 px.

Model	mAP@0.5	FPS
Faster R-CNN + Resnet50	90.02%	12
Faster R-CNN + MobilenetV3	83.66%	52
YOLOv5 S	90.1%	106
YOLOR P6	80.4%	47

TABLE 5.1: Model Speed Comparison

Model	Soldier	Vehicle	Unidentified Object
Faster R-CNN + Resnet50	76.53%	96.86%	96.83%
Faster R-CNN + MobilenetV3	61.35%	93.12%	96.51%
YOLOv5 S	79.5%	95.1%	72.6%
YOLOR P6	84.6%	96.3%	60.5%

TABLE 5.2: Per class AP@0.5 Comparison

Model	MiB
Faster R-CNN + Resnet50	1975
Faster R-CNN + MobilenetV3	1271
YOLOv5 S	1099
YOLOR P6	1281

TABLE 5.3: Model Inference GPU memory

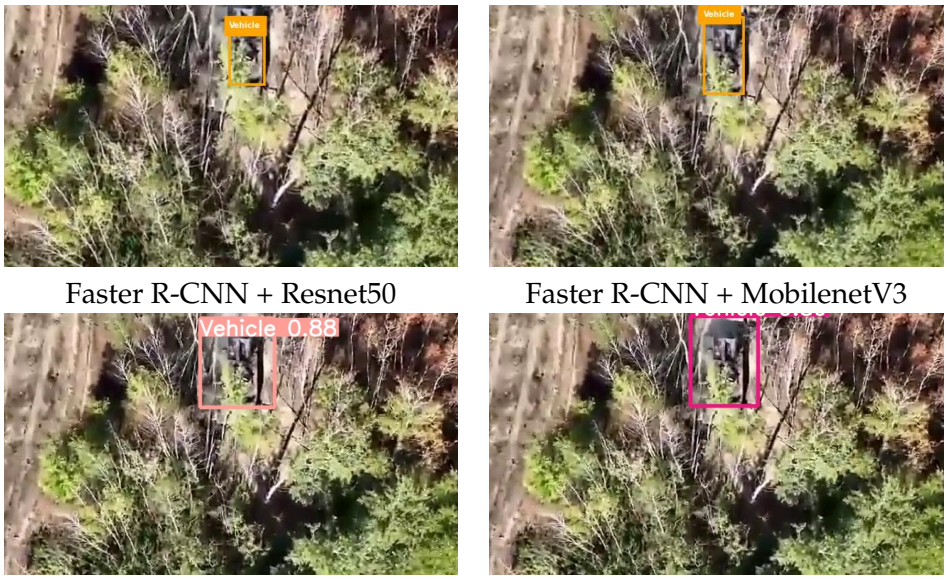


FIGURE 5.1: Detection sample comparison

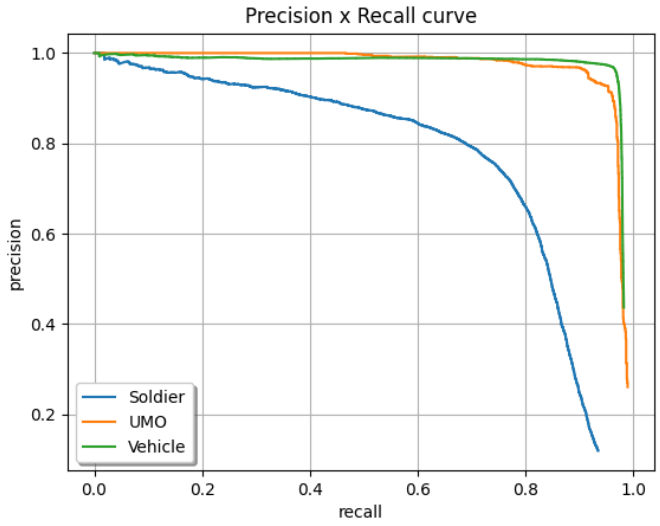


FIGURE 5.2: Resnet50 Precision/Recall Graph

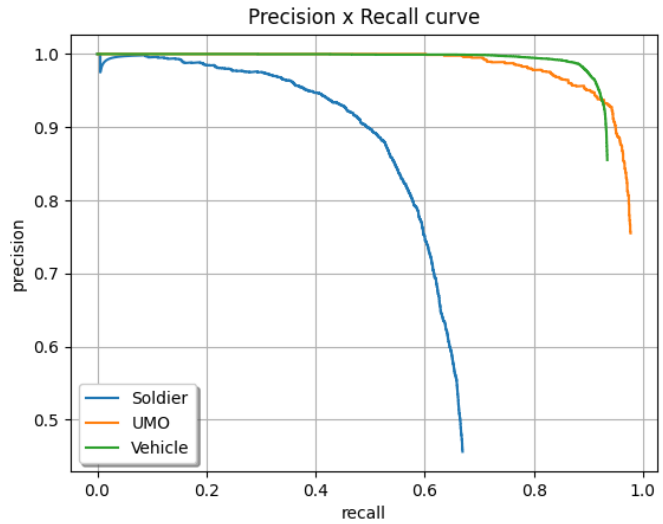


FIGURE 5.3: MobileNet V3 Precision/Recall Graph

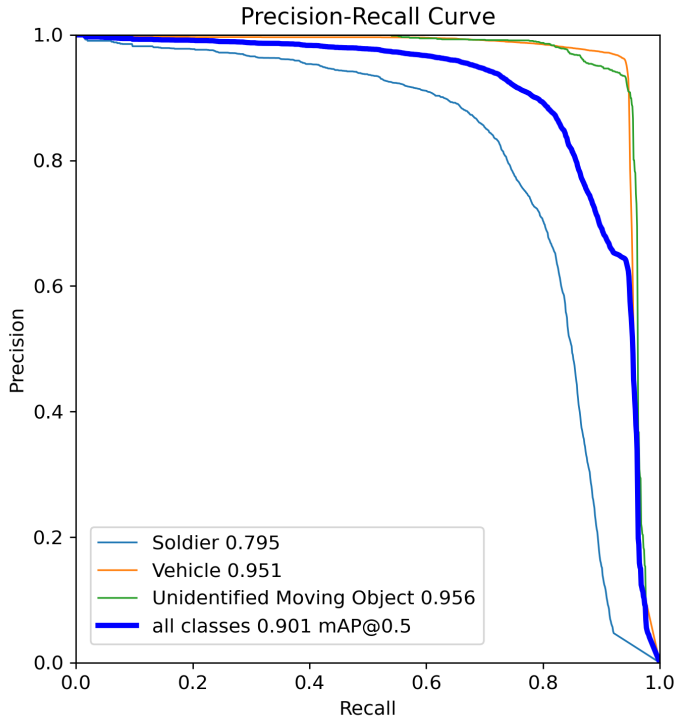


FIGURE 5.4: YOLOv5 Precision/Recall Graph

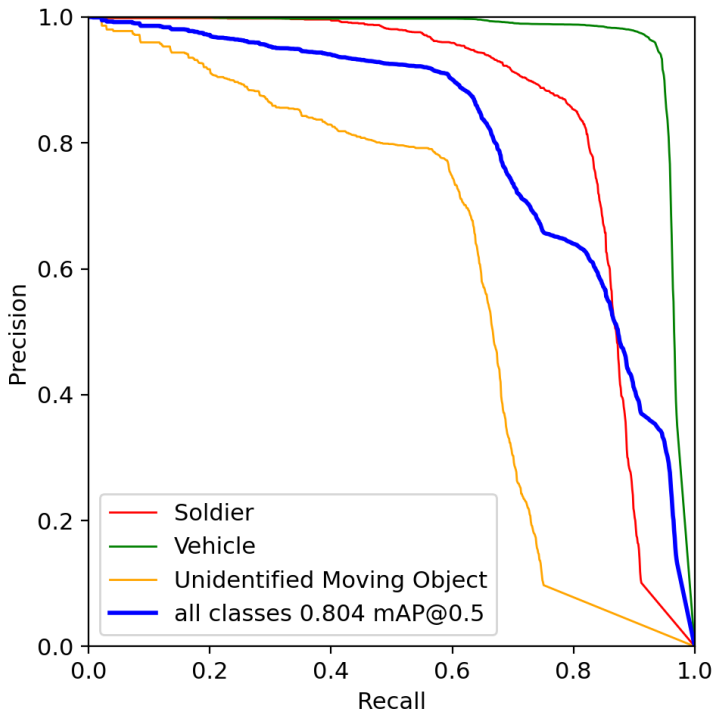


FIGURE 5.5: YOLOR Precision/Recall Graph

Chapter 6

Conclusion

In this paper, we've established an image dataset for detecting objects of military equipment. It contains 86360 labeled frames and 124691 bounding boxes collected from 120 videos. We've also trained and compared four detection models: 2 based on the Faster R-CNN family of models and two based on YOLO. YOLOv5 showed the best results in inference speed and memory usage. Both Faster R-CNN models exceeded in detecting the Unidentified Object class but lacked in all other aspects. YOLOR exceeded the accuracy of YOLOv5 in detecting Soldier and Vehicle objects but performed almost twice as slowly. For the task of this work, YOLOv5 is overall the best choice, trading a negligible accuracy difference for a very fast inference speed.

Current work can be improved in several ways, namely, by increasing the size of the dataset and training the models from scratch as opposed to transfer learning. The next step would be the implementation of a multi-object tracker that would utilize one of the trained models.

Bibliography

[1] Stefan Becker et al. *An Evaluation of Trajectory Prediction Approaches and Notes on the TrajNet Benchmark*. 2018. arXiv: 1805.07663 [cs.CV].

[2] Alexey Bochkovskiy. *YOLOv4 / Scaled-YOLOv4 / YOLO - Neural Networks for Object Detection (Windows and Linux version of Darknet)*. URL: https://github.com/AlexeyAB/darknet#how-to-train-to-detect-your-custom-objects.

[3] Alexey Bochkovskiy, Chien-Yao Wang, and Hong-Yuan Mark Liao. *YOLOv4: Optimal Speed and Accuracy of Object Detection*. 2020. arXiv: 2004.10934 [cs.CV].

[4] CVAT.ai Corporation. *Computer Vision Annotation Tool (CVAT)*. Version 2.2.0. Sept. 2022. URL: https://github.com/opencv/cvat.

[5] Ross Girshick. *Fast R-CNN*. 2015. arXiv: 1504.08083 [cs.CV].

[6] Ross Girshick et al. *Rich feature hierarchies for accurate object detection and semantic segmentation*. 2014. arXiv: 1311.2524 [cs.CV].

[7] Google. *Google Cloud Platform*. 2008. URL: cloud.google.com.

[8] Harmon, Joshua Roesslein, and other contributors. *Tweepy*. DOI: 10.5281/zenodo.7259945. URL: https://github.com/tweepy/tweepy.

[9] Kaiming He et al. "Spatial Pyramid Pooling in Deep Convolutional Networks for Visual Recognition". In: *Computer Vision – ECCV 2014*. Springer International Publishing, 2014, pp. 346–361. DOI: 10.1007/978-3-319-10578-9_23. URL: https://doi.org/10.1007%2F978-3-319-10578-9_23.

[10] Meng-Ru Hsieh, Yen-Liang Lin, and Winston H. Hsu. *Drone-based Object Counting by Spatially Regularized Regional Proposal Network*. 2017. arXiv: 1707.05972 [cs.CV].

[11] Longlong Jing and Yingli Tian. *Self-supervised Visual Feature Learning with Deep Neural Networks: A Survey*. 2019. arXiv: 1902.06162 [cs.CV].

[12] Glenn Jocher et al. *ultralytics/yolov5: v7.0 - YOLOv5 SOTA Realtime Instance Segmentation*. Version v7.0. Nov. 2022. DOI: 10.5281/zenodo.7347926. URL: https://doi.org/10.5281/zenodo.7347926.

[13] Tsung-Yi Lin et al. *Microsoft COCO: Common Objects in Context*. 2015. arXiv: 1405.0312 [cs.CV].

[14] R. Padilla, S. L. Netto, and E. A. B. da Silva. "A Survey on Performance Metrics for Object-Detection Algorithms". In: *2020 International Conference on Systems, Signals and Image Processing (IWSSIP)*. 2020, pp. 237–242.

[15] Rafael Padilla et al. "A Comparative Analysis of Object Detection Metrics with a Companion Open-Source Toolkit". In: *Electronics* 10.3 (2021). ISSN: 2079-9292. DOI: 10.3390/electronics10030279. URL: https://www.mdpi.com/2079-9292/10/3/279.

[16] Adam Paszke et al. "PyTorch: An Imperative Style, High-Performance Deep Learning Library". In: *Advances in Neural Information Processing Systems 32*. Ed. by H. Wallach et al. Curran Associates, Inc., 2019, pp. 8024–8035. URL: http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf.

[17] Joseph Redmon et al. "You Only Look Once: Unified, Real-Time Object Detection". In: *CoRR* abs/1506.02640 (2015). arXiv: 1506.02640. URL: http://arxiv.org/abs/1506.02640.

[18] Joseph Redmon et al. *You Only Look Once: Unified, Real-Time Object Detection*. 2016. arXiv: 1506.02640 [cs.CV].

[19] Shaoqing Ren et al. *Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks*. 2016. arXiv: 1506.01497 [cs.CV].

[20] Leonard Richardson. *Beautiful Soup*. 2004. URL: https://www.crummy.com/software/BeautifulSoup/.

[21] *Twint*. URL: https://github.com/twintproject/twint.

[22] Jasper Uijlings et al. "Selective Search for Object Recognition". In: *International Journal of Computer Vision* 104 (Sept. 2013), pp. 154–171. DOI: 10.1007/s11263-013-0620-5.

[23] Chien-Yao Wang, I-Hau Yeh, and Hong-Yuan Mark Liao. "You Only Learn One Representation: Unified Network for Multiple Tasks". In: *arXiv preprint arXiv:2105.04206* (2021).

[24] WongKinYiu. "YOLOR". In: *GitHub repository* (2021). URL: https://github.com/WongKinYiu/yolor.

[25] Pengfei Zhu et al. "Detection and Tracking Meet Drones Challenge". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* (2021), pp. 1–1. DOI: 10.1109/TPAMI.2021.3119563.