Міністерство освіти і науки України
НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «КИЄВО-МОГИЛЯНСЬКА АКАДЕМІЯ»
Кафедра інформатики факультету інформатики

РОЗПІЗНАВАННЯ І КЛАСИФІКАЦІЯ ВІЙСЬКОВОЇ ТЕХНІКИ МЕТОДАМИ КОМП'ЮТЕРНОГО ЗОРУ.

RECOGNITION OF MILITARY EQUIPMENT USING COMPUTER VISION METHODS.

Текстова частина до кваліфікаційної роботи
за спеціальністю «Комп’ютерні науки» 122

Керівник курсової роботи
ст.в. Салата К. В.

_____

(підпис)

Виконав студент 3 курсу
Антощук Р.О.

«____»____2025 р.

Київ 2025

Міністерство освіти і науки України
НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «КИЄВО-МОГИЛЯНСЬКА АКАДЕМІЯ»
Кафедра інформатики факультету інформатики

ЗАТВЕРДЖУЮ

Зав. кафедри інформатики,

к.ф.-м.н., доц. Гороховський С.С

_____

(підпис)

«___» _____ 2025 р

Календарний план виконання курсової роботи

Тема: Розпізнавання військової техніки на фото і відео за допомогою технологій комп'ютерного зору.

№	Назва етапу курсового проекту (роботи)	Термін виконання етапу	Примітка
1.	Погодження теми з науковим керівником.	06.11.2024	
2.	Аналіз матеріалів за темою дипломної роботи.	20.02.2025	
3.	Визначення підходів і рішень до реалізації, на основі порівняння з іншими роботами на дану тему.	10.03.2025	
4.	Реалізація програмної компоненти.	30.04.2025	
5.	Написання текстової частини дипломної роботи.	05.05.2025	
6.	Створення анотації. Надання дипломної роботи на перевірку науковим керівником.	06.05.2025	
7.	Коригування на основі результатів перевірки.	08.05.2025	
8.	Подання роботи на перевірку на відповідність вимог академічної доброчесності.	08.05.2025	
9.	Захист дипломної роботи.	15.05.2025	

Антощук Р.О. _____

Салата К.В. _____

«____» _____ _____ р.

ABSTRACT

This research explores how to achieve the minimum sufficient performance for military equipment recognition within a hypothetical, wide-area situational-awareness system. Once demanded deep expertise in neural-network architecture design and complex deployment pipelines can now be realized using two YOLOv8 models built entirely on open-source datasets, frameworks, and free cloud GPUs delivering 69.5% and 72.4% mAP@50 accuracy.

We met the problem of merging sources with wildly varying scopes, resolving labeling inconsistencies and duplicate images that inflated confidence scores, addressing severe class imbalance across 14 vehicle categories, and witnessing inter-class confusion among visually similar targets.

This work demonstrates that, with modern tools, building a robust military equipment reconnaissance pipeline - from raw images to deployable video inference is both practical and accessible, achieving high accuracy on fine-grained classes while running smoothly on limited GPU resources.

TABLE OF CONTENTS

INTRODUCTION

Object recognition is a task of computer vision that includes identification and determination of locations of different objects on image or video. Referring to that statement and other projects on that problem name “recognition” already includes research in classification paradigm of set topic as itself. So by saying Recognition it is meant recognition and classification.

Reconnaissance of open-source data, especially publicly available road cameras to detect the presence of military vehicles and equipment. Further development of this reconnaissance tool, for example as part of a wide-broad system that combines data from multiple sources and uses counts of specific military equipment types observed by road cameras in a particular region of supply chain to confirm or refute information from other intelligence sources. Or deployed as a dash-cam, to collect continuous real-time imagery and feed it into the same pipeline for immediate alerting and enhanced situational awareness

1 REVIEW

1.1 Evolution of YOLO Architectures

Since its debut in 2016, [26] YOLO has pioneered object detection—merging proposal and classification into a single neural network pass to achieve real-time speeds. Early YOLO versions relied on anchor boxes and sacrificed a small amount of accuracy for this efficiency. Over time, [27] YOLOv5, [28] YOLOv6 and [29] YOLOv7 each introduced refinements: better multi-scale feature fusion, streamlined training recipes, and optimized layer designs to gradually close the accuracy issues. In 2023, Ultralytics released [19] YOLOv8, which further simplifies the workflow by integrating built-in augmentations [23] (Mosaic tiling and RandomPerspective) and providing [30] PyTorch, making it exceptionally easy to prototype and deploy on both cloud servers and local devices(if only processing powers were so accessible).

1.2 Transfer Learning & Fine-Tuning

Aurelien Geron’s [31] Hands-On Machine Learning with Scikit-Learn, Keras & TensorFlow (Ch 10.6) it is shown how to load a pre-trained convolutional backbone and then “unfreeze” its top layers for further training on a new, smaller

dataset. This approach - initializing weights from a model already versed in general image features - speeds up convergence and often yields higher accuracy than training from scratch. In our pipeline, we mirror this by initializing YOLOv8 with its official pretrained “best.pt” weights and then fine-tuning for few tens epochs on our military equipment dataset (rather than full re-training), which reduced training time by nearly 40% while improving mAP%50 by +0.030 on rare classes.

1.3 On-the-Fly Data Augmentation

In [31] Chapters 10–11, it is emphasized using real-time augmentation (random flips, translations, zooms, brightness and contrast jitters) within the training loop to synthetically enlarge small datasets and regularize models. We implement an analogous strategy via Roboflow’s augmentation pipeline: combining random perspective transforms, Mosaic tiling, and HSV jitter to increase both the diversity and robustness of our training images. As it notes, proper augmentation can boost generalization by up to 10% on limited data [31] (Ch 10.9), which aligns with our observed +0.050 mAP%50 gain when comparing augmented vs non-augmented runs.

2 APPROACH

To find an approach firstly we need to determine the problem. The biggest obstacle I met from the first glance was dataset: some sources are too specific, containing only a few vehicle types or very unique scenarios, while others are too vast, mixing non-military content and irrelevant classes. Trying to distinguish many similar looking categories like IFV, APC etc., leads to confusion with classes. Compounding many public datasets suffer from inconsistencies with labeling: bounding boxes that stretch outside the object, missing annotations for partially visible vehicles or for no reason at all, and ambiguous class definitions as Tank referring to several types of vehicles. In conclusion there is no one dataset you can take and move on. Another problem is to find training model that meets my demands. I need a network that delivers high accuracy on complex classes yet remains easy to integrate into my workflow and being efficient enough to train in a reasonable time. Heavyweight architectures may show high metrics results but

demand too much processing power and time for training, which are severely limited due to high GPU and energy costs.

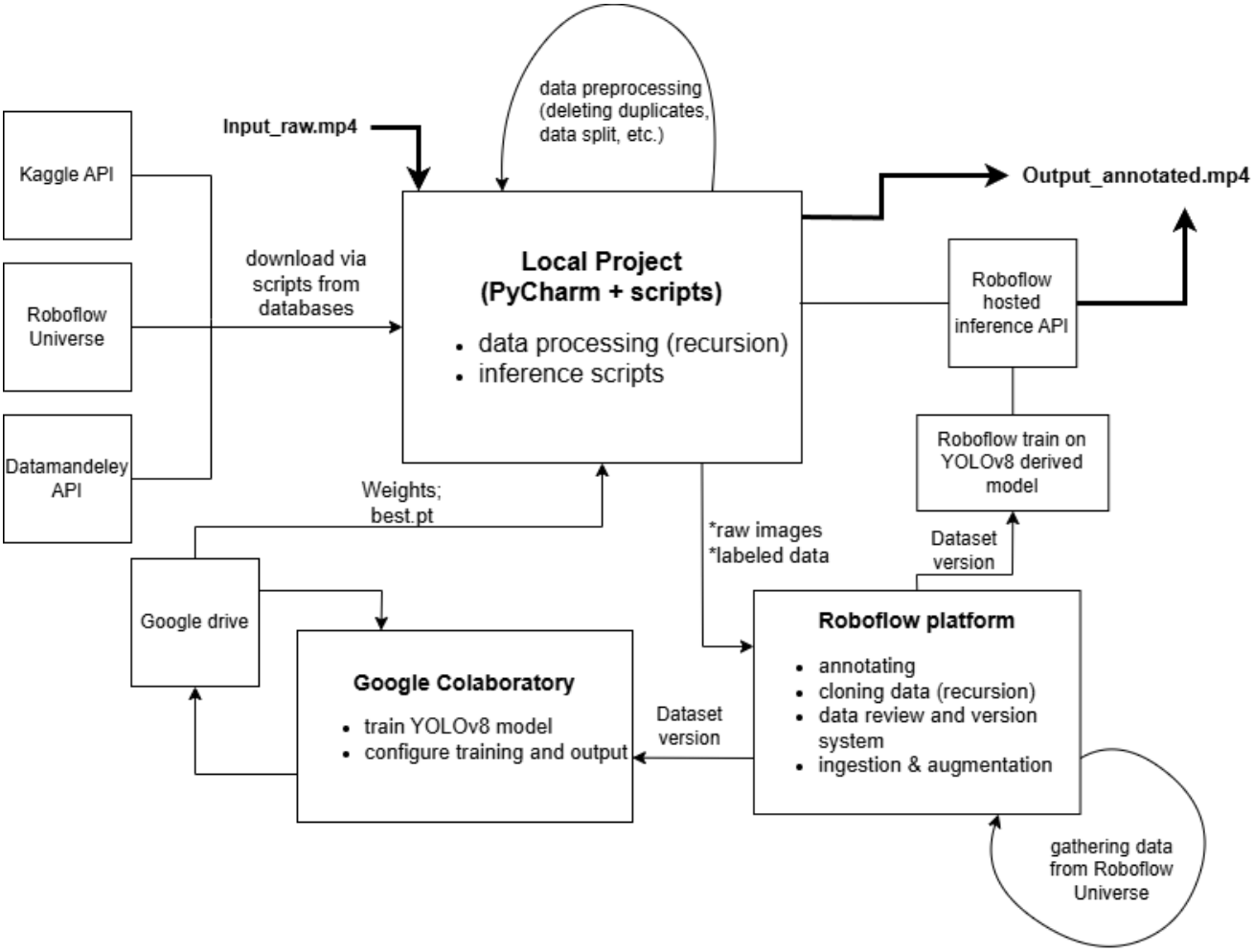


Figure 2.1 – Workflow of my Approach with data flows [34]

I set process from gathering data to deploying working model through different platforms. Combining open sources of data, online tools, python scripts, Command-Line interface and API interaction to find optimal strategy of creating object recognition model in our days.

3 SOLUTION

3.1 Dataset

Dataset has the biggest impact on the final result and it takes the major part of time, so I needed to find optimal way to manage workflow with it.

3.1.1 Dataset base

For base I chose [1]Military and Civilian Vehicles Classification dataset by Priyanka Gupta , Bhavya Pareek , Gaurav Singal , D Vijay Rao. Because of variety of military equipment and civilian vehicles already included which is the important part of distinguishing vehicles on public roads. It consists of 6772 images and several label formats. As a row it includes 4 classes: Military Truck, Military Tank, Military Aircraft, Military Helicopter, Civilian Car, Civilian Aircraft.

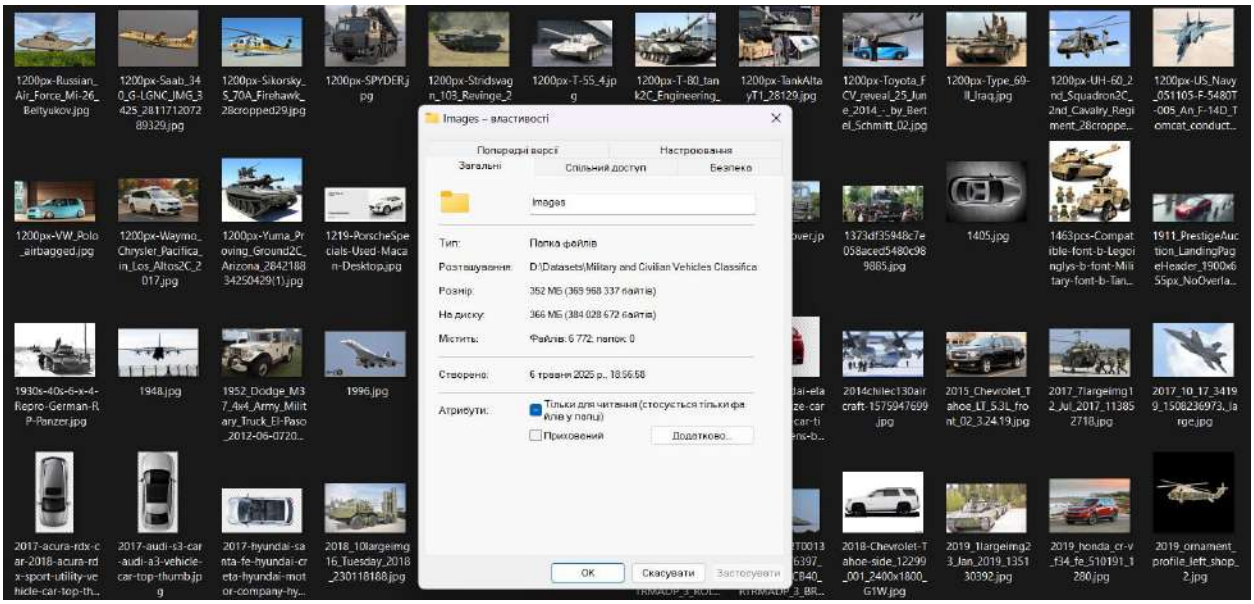


Figure 3.1.1 – raw version [1] dataset look

4 formats of annotations:

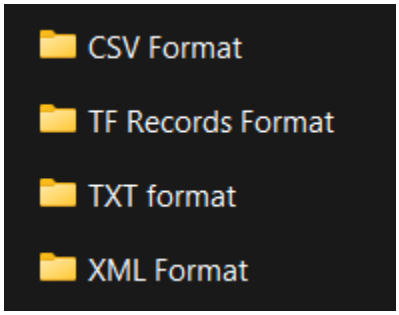


Figure 3.1.2 – present labeling formats

Vast majority of images were pure copies or augmentations, most cases were those that would negatively influence the result in the end. Removed part of them using open-source [6] dedupe script by dvschultz and part of them by hand.

3.1.2 Dataset management approach

My approach goes with [2] Roboflow for finding new data, merging it with existing and annotating. The platform itself represents whole self-sufficient habitat with enormous base of different open-sourced datasets called Roboflow Universe.

I found a problem uploading datasets bigger than 2000 instances with browser: some images couldn't download some annotations were lost. Uploading via CLI or custom script as a more reliable variant left no space for error:

```
(pythonProject1) PS C:\Users\Romchik\PycharmProjects\pythonProject1> pip install roboflow
Collecting roboflow
  Downloading roboflow-1.1.63-py3-none-any.whl.metadata (9.7 kB)
Collecting certifi (from roboflow)
```

```
(pythonProject1) PS C:\Users\Romchik\PycharmProjects\pythonProject1> roboflow login
visit https://app.roboflow.com/auth-cli to get your authentication token.
Paste the authentication token here:
(pythonProject1) PS C:\Users\Romchik\PycharmProjects\pythonProject1> roboflow import -w science-paper -p milita
ry-equipment-recognition D:\Datasets\"1 Military and Civilian Vehicles Classification"
loading Roboflow workspace...
100%|
| 2967/2967 [00:00<00:00, 2937101.72it/s]
loading Roboflow project...
Uploading to existing project science-paper/military-equipment-recognition
```

Figures 3.1.3 - 3.1.4 – uploading of dataset via CLI

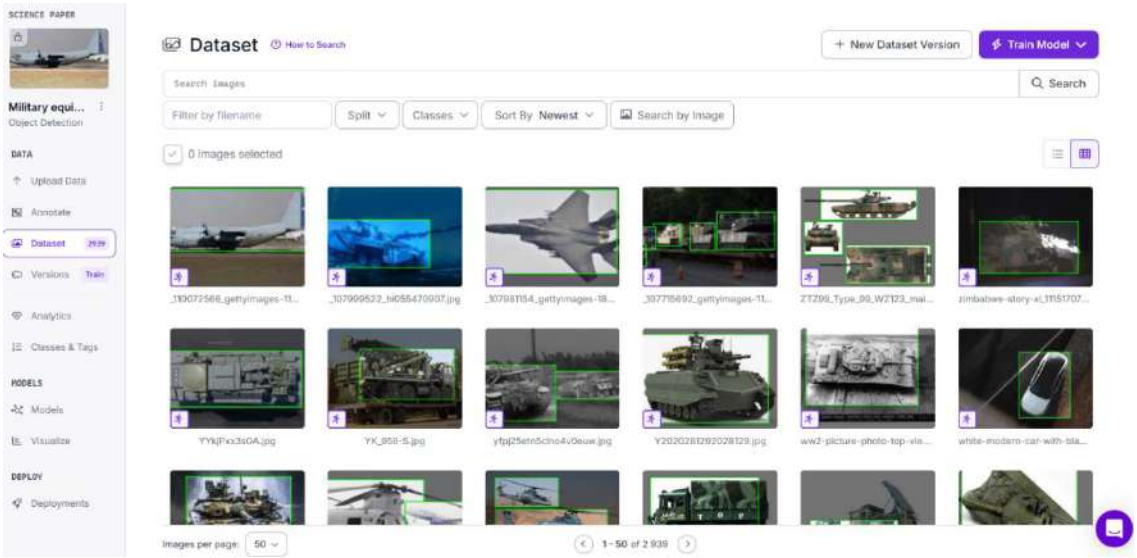


Figure 3.1.5 – uploaded dataset demonstration

3.1.3 Classes determination according to subject area

By my subjective opinion it is the best to distinguish such classes in this dataset to achieve the goal determined in the usage explanation:

Vehicle Class	NATO symbology	Key Distinguishing Features
Main Battle Tank (MBT)		Heavy armor, large-caliber main gun, tracked mobility, high cross-country performance. Low silhouette for russian tanks. On tracks.
Infantry Fighting Vehicle (IFV)		Carries infantry squad, equipped with autocannon, often includes anti-tank missiles, provides direct fire support. Primarily wheeled.
Armored Personnel Carrier (APC)		Lightly armed, primarily for transport, less armored than IFVs, usually wheeled.
Self-Propelled Artillery (SPA)		Mounted artillery on armored chassis, provides mobility and protection for artillery units. Large-caliber gun, significantly long barrel.
Multiple Launch Rocket System (MLRS)		Launches multiple rockets in quick succession, mounted on wheeled or tracked vehicles.
Anti-Aircraft System (AAS)		Equipped with anti-aircraft guns or missiles, may be self-propelled or towed.
Aircraft		Fixed-wing aircraft.
Helicopter		Rotary-wing aircraft; variants include attack helicopters, utility helicopters, and transport helicopters.
Unmanned Aerial Vehicle (UAV)		Remotely piloted or autonomous, varies in size and capability, used for ISR and strike missions.
Logistics Vehicle		Includes trucks and cargo carriers, may be armored or unarmored, essential for supply chains.
Engineer Vehicle		Equipped for tasks like mine clearing, bridge laying, and obstacle removal; often based on tank chassis. Crane or mine-sweeper.
Mine-Resistant Ambush Protected Vehicle (MRAP)		V-shaped hull for blast deflection, designed to withstand IEDs and ambushes, heavily armored. Typical HMMWV.
Radar Acquisition and Display System (RADS)		System designed to record and display data from a research Huge Doppler radar.
Non-military / Undetermined		Any vehicle that isn't included in military classes above.

Table 3.1.1 – defined classes with NATO symbols

[16] NATO symbology icons created using [17] Fotor AI image cropper.

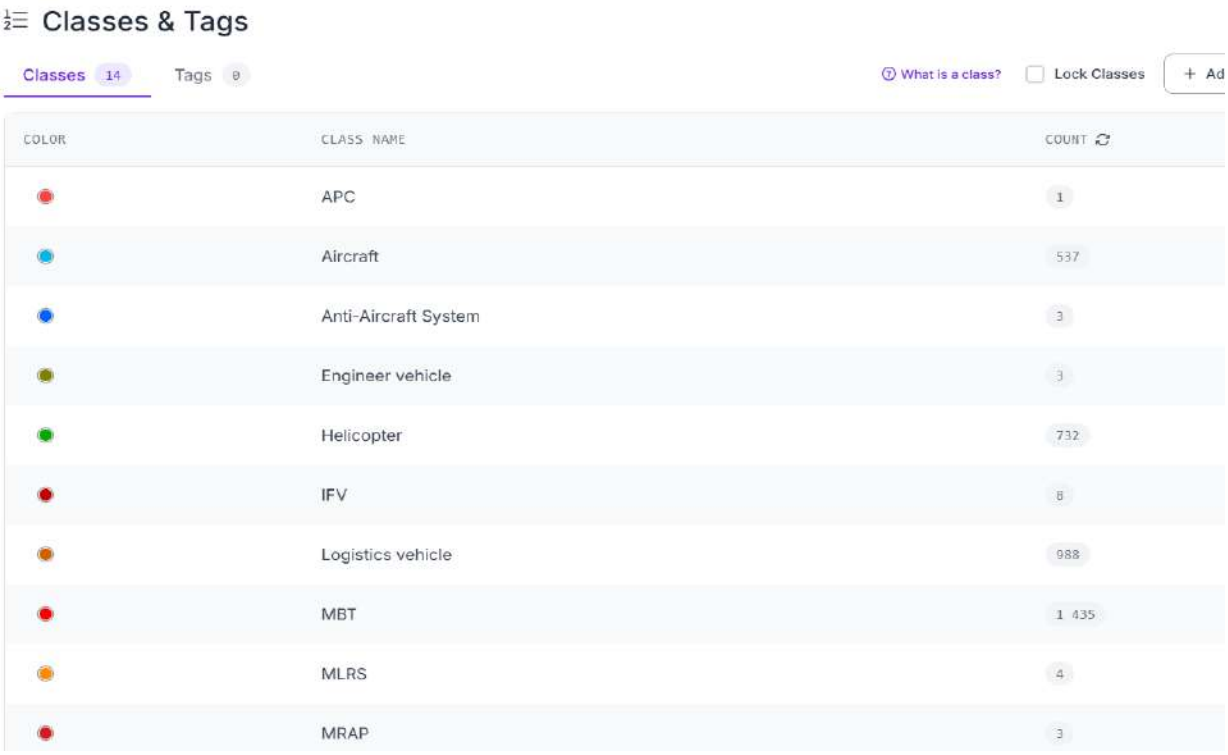


Figure 3.6 – Classes demonstration in [2] Roboflow UI

3.1.4 Roboflow Annotations

[5] Roboflow Annotations is useful tool for correcting existing labels as my goal was to accomplish more detailed distinguishment between ground military vehicles classification.

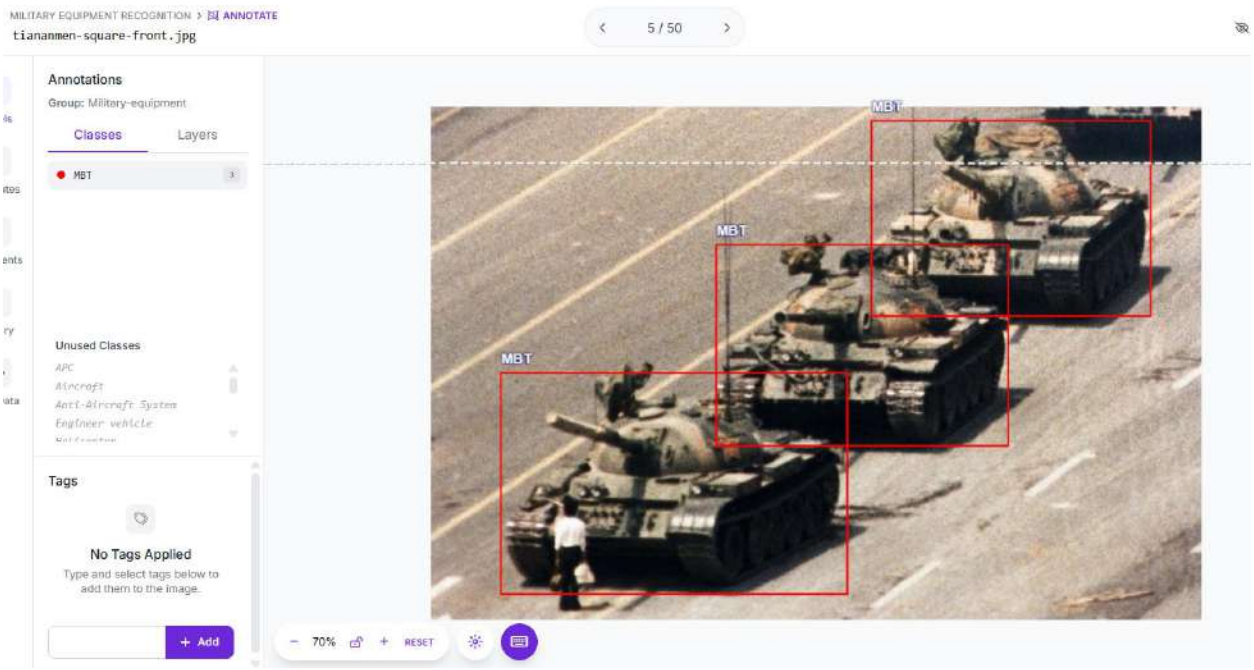


Figure 3.1.7 – Annotating process demonstration

for managing datasets classes and images. In my case its to differentiate military vehicles from two classes MBT and Logistics vehicle to several more specific as IFV, APC or RADS and Anti-Aircraft System

Figure 3.1.8 – Annotating process demonstration

As a result of removing useless data and relabeling got 2482 images with such class distribution:

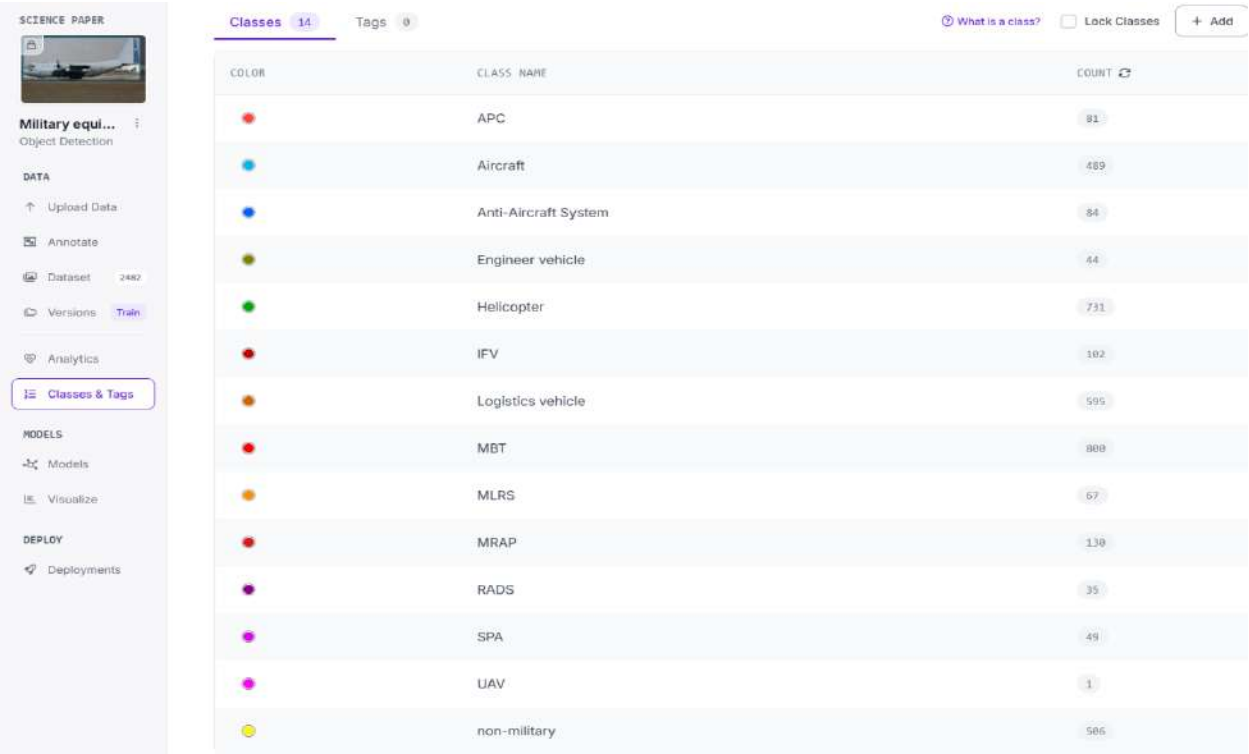


Figure 3.1.9 – Dataset and Classes after processing

Next obvious step was to fill insufficient classes with data from other datasets.

3.1.5 Merging datasets; CLI approach

This is where huge environment of products plays role [7] Roboflow Universe gives access to thousands of computer vision datasets:

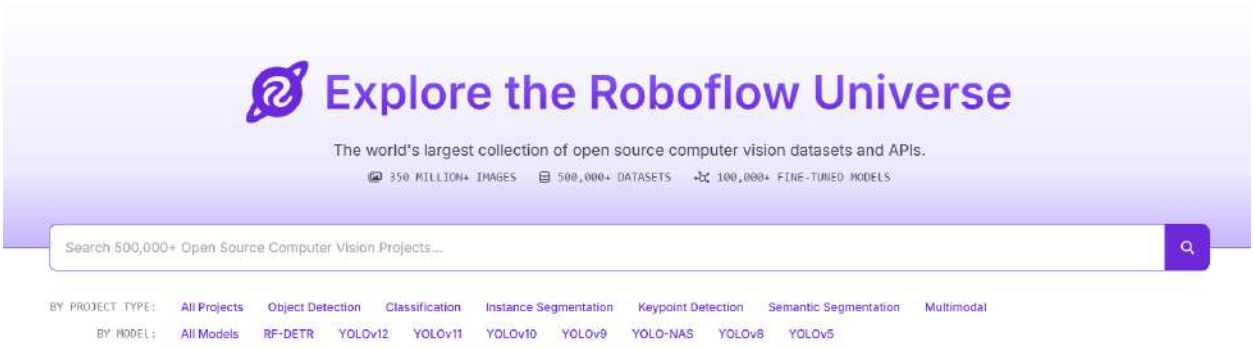


Figure 3.1.10 – [7] Roboflow Universe view

Here I cloned annotated images from [8] “kek Computer Vision Project” by OLD SkyDefender ; [9] Russian-military-annotated Computer Vision Project by CapstoneProject ; [10] “BTR Computer Vision Project” by ZST ; [11] “BMP3_Reference BTR80_Reference T90A_Reference Computer Vision Project”s by TankIdentifier ; [12] “multi_rocket_launcher_ODv1 Computer Vision Project” by yolotank .

Those are very specific datasets, so labeling is almost always perfect, but to merge them with my classes after cloning I needed to set BMP-2 to IFV, BMP-3 to IFV, T-90 to MBT, BM-21 to MLRS, etc. and needed to delete support classes for distinguishing, even though it’s a good practice to determine specific models of Armored Personal Carier or Multiple Launch Rocket System by their parts such as wheel-base or rocket launcher type from [12]dataset by yolotank shown on pictures below, it would cause excessive noise in my already very wide-class dataset.

Figure 3.1.11-3.1.12 – subclasses annotations for specific models

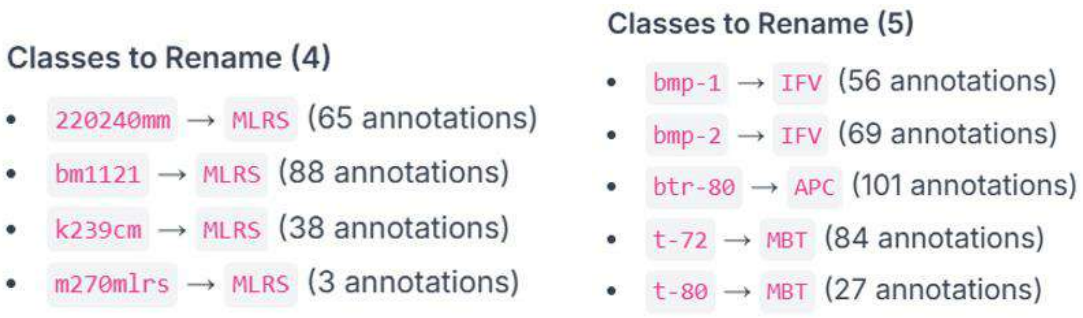


Figure 3.1.13 – merging classes

Also I tried merging [14] CaseStudy3 dataset with RADS by btl in my dataset with upload_RADS_dataset.py script and API_key. The point of interest was split data:

```
Uploaded  train | 93_png.rf.2edc696b35631824f29cff89e3e8d408 |  1.39s
Uploaded  train | 9_png.rf.0300a5c33bf31a0c86c588131eb5e8a6 |  1.66s

 Uploading dataset part(split): valid
Uploaded  valid | 11_png.rf.919f48e57011bb9859d058d6cdee2698 |  1.22s
Uploaded  valid | 14_png.rf.8b1a37f8542b483308fbda2a99a958ab |  1.23s
Uploaded  valid | 23_png.rf.67bef6f5c5ad4576bfb6eadd26a34a35 |  1.82s
Uploaded  valid | 25_png.rf.295dde87dc550ed30aa0ae003023a6ab |  1.22s
Uploaded  valid | 27_png.rf.e6f61069900e3bc308c3f17609959fdf |  1.25s
Uploaded  valid | 2_png.rf.4dbe876cd394ed1e66f25e6247ef1289 |  1.28s
Uploaded  valid | 70_png.rf.63de3881e66621b26f32ab775e93ed9b |  1.93s
Uploaded  valid | 90_png.rf.ee1d5bbf7ad5d51821f864b055c01322 |  1.25s
Uploaded  valid | 91_png.rf.71e1f3134d8d647a344470634eed153a |  1.39s
Uploaded  valid | 97_png.rf.94b61c46be88d0bac1e6b62a0e0240e3 |  1.29s

 Uploading dataset part(split): test
Uploaded  test  | 30_png.rf.1696e1a0dbd0fe49bbc4a48f05d000b4 |  1.65s
Uploaded  test  | 32_png.rf.d918a5fc93612f4d210f26038fc17bea |  1.44s
Uploaded  test  | 46_png.rf.5d56f8134626197776f05b00a4762adb4 |  1.27s
Uploaded  test  | 51_png.rf.84cafa131dcaaa4b7ab50d0e5c4297a1 |  1.42s
Uploaded  test  | 78_png.rf.440e6e2d2e02e9af608841813ff945f9 |  1.29s
Uploaded  test  | 94_png.rf.5d78103c270dc64abb5970ad0d7669aa |  1.23s
Uploaded  test  | 99_png.rf.44824bc6343fce50bfe5afdfe54ffa6e |  1.48s

Total upload time: 145.92 seconds.
```

Figure 3.1.14 – uploading split dataset with python script

However it cause no problem because that split data management in Roboflow is clear and simple:

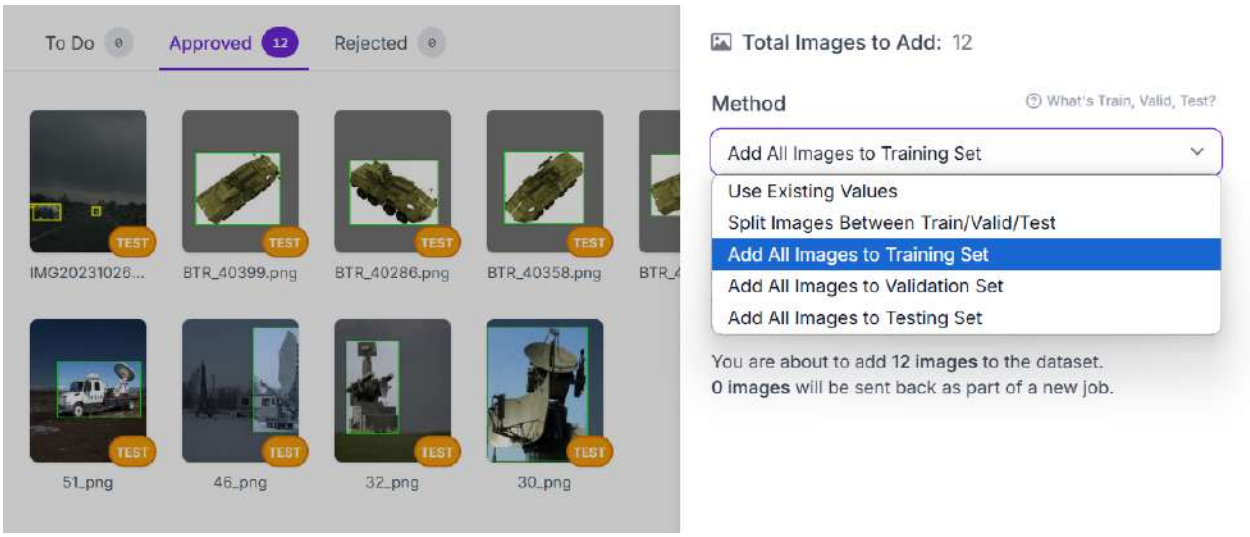


Figure 3.1.15 – split data management tool

At this point I decided not to fill UAV class with data, because of its lack of necessity for task it is currently set to perform; and rise of processing power it demands. But left it as class with only one instance to check how it will influence the model as a part of experiment. Running ahead of analysis [17] YOLO model neglects such values and give absolutely same results in metrics for dataset that includes single instance class and one that doesn't.

So that the final annotations numbers for 3299 images with total of 4857 are:

Category	***Count***	*IFV*	*284*
MBT	981	*SPA*	159
Helicopter	731	*RADS*	146
Logistics vehicle	634	*MRAP*	132
Non-military	545	*Anti-Aircraft System*	111
Aircraft	488	*Engineer vehicle*	44
APC	307	*UAV*	1
MLRS	294		

Table 3.1.2 – total annotations distribution

All datasets used in this work are split into: 71% train, 20% valid and 9% test.

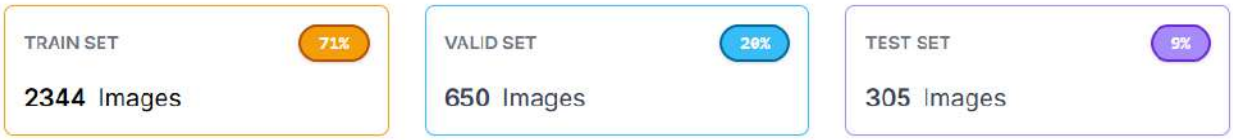


Figure 3.1.16 – data split in Roboflow

3.2 Training a model

3.2.1 Model choice

Easy to run, huge choice of up-to date models for different tasks and biggest computer vision community – choosing [3]You Only Look Once neural network was obvious decision after made review [1.1 Evolution of YOLO Architectures].

And considering [19][20] metrics and infographics proposed by Ultralytics on their documentation and media.

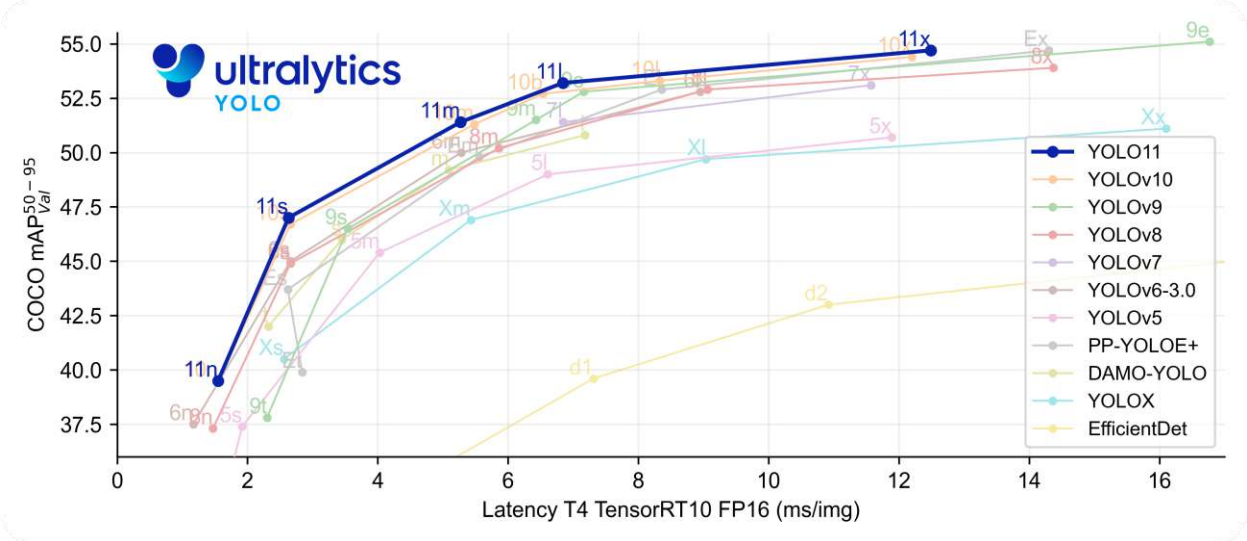


Figure 3.2.1 – ultralitics model comparison via mAP50-95

YOLOv8 suits my purposes the best as a supported and well-known(enormous open-source projects base, problem solving and issues resolving), but still high-performing model.

3.2.2 Training on local machine

Figure 3.2.2 – ensuring GPU usage

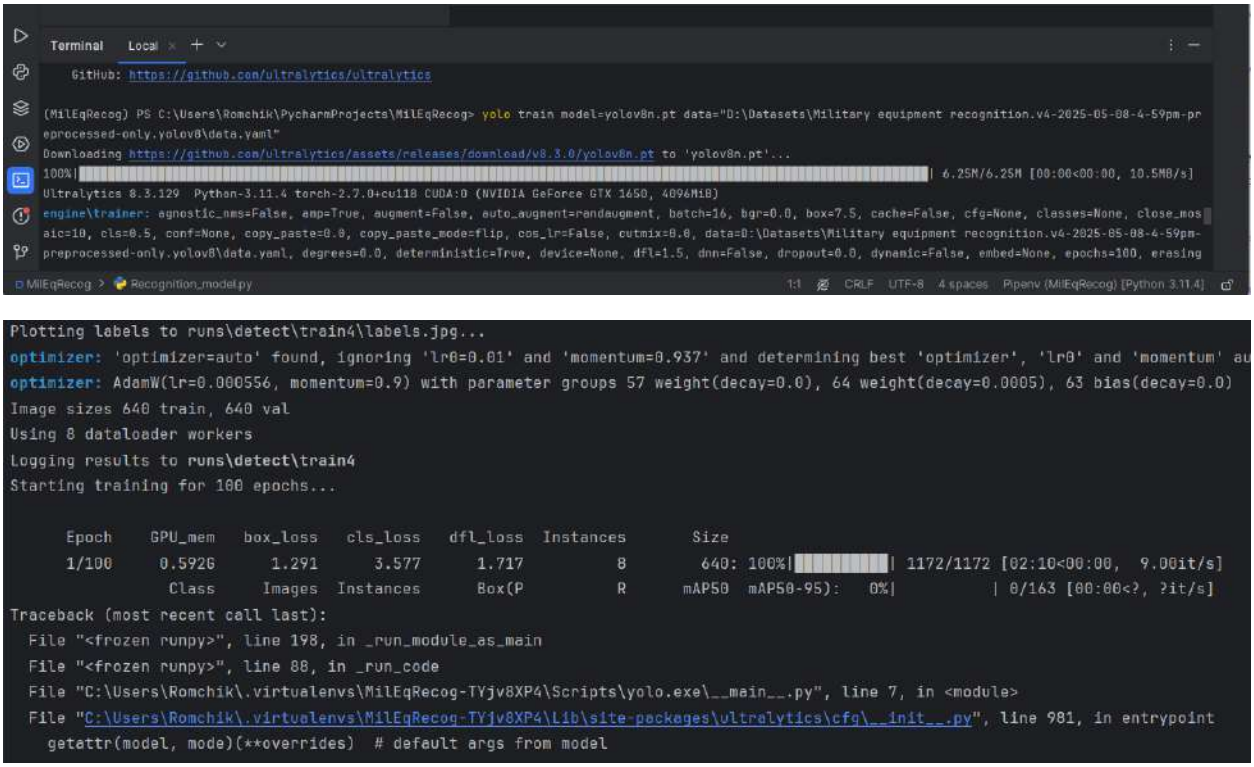


Figure 3.2.3 - 3.2.4 – training model locally on GPU via CLI command

Met the problem of running out of memory working with [30] PyTorch. Could be solved by setting virtual environment on another disk, setting project, installing all packages and training there. But considering the power capabilities of my GPU (completing 1 epoch in 130 seconds with nano model) took decision to continue work in [4] Google Colaboratory.

3.2.3 [4] Google Colaboratory training

Ensuring usage of powerful GPU (Tesla T4 16Gb):

```
[ ] !nvidia-smi

Thu May  8 16:44:58 2025
+-----------------------------------------------------------------------------------------+
| NVIDIA-SMI 550.54.15              Driver Version: 550.54.15      CUDA Version: 12.4     |
|-----------------------------------------+------------------------+----------------------+
| GPU  Name                 Persistence-M | Bus-Id          Disp.A | Volatile Uncorr. ECC |
| Fan  Temp   Perf          Pwr:Usage/Cap |           Memory-Usage | GPU-Util  Compute M. |
|                                         |                        |               MIG M. |
|=========================================+========================+======================|
|   0  Tesla T4                       Off |   00000000:00:04.0 Off |                    0 |
| N/A   51C    P8             10W /   70W |       0MiB /  15360MiB |      0%      Default |
|                                         |                        |                  N/A |
+-----------------------------------------+------------------------+----------------------+

+-----------------------------------------------------------------------------------------+
| Processes:                                                                              |
|  GPU   GI   CI        PID   Type   Process name                              GPU Memory |
|        ID   ID                                                               Usage      |
|=========================================================================================|
|  No running processes found                                                             |
+-----------------------------------------------------------------------------------------+
```

```
[ ] import torch

    print("CUDA available:", torch.cuda.is_available())
    print("Device name:", torch.cuda.get_device_name(0))

CUDA available: True
Device name: Tesla T4
```

Figure 3.2.5 – ensuring T4 GPU usage

Mounting google drive to save and export weights:

```
[ ] from google.colab import drive
    drive.mount('/content/drive')

Drive already mounted at /content/drive; to attempt to forcibly remou
```

```
[ ] DRIVE_RUNS = "/content/drive/MyDrive/MilEqRecog/runs"
    import os
    os.makedirs(DRIVE_RUNS, exist_ok=True)
```

Figure 3.2.6 – google drive connection

Connecting my preprocessed database from Roboflow via API and ensuring connection:

```
[ ] !pip install ultralytics roboflow --quiet
```

```
[ ] from roboflow import Roboflow
```

```
[ ] rf = Roboflow(api_key="")
    project = rf.workspace("science-paper").project("military-equipment-recognition")
    dataset = project.version(5).download("yolov8") #version of dataset 5 - 640x640
```

```
loading Roboflow workspace...
loading Roboflow project...
```

```
[ ] !ls {dataset.location}
```

```
data.yaml  README.roboflow.txt  test  train  valid
```

Figure 3.2.7 – connecting database via Roboflow_API

Choosing YOLO model, configuring training details and saving results to google drive:

```
[ ] from ultralytics import YOLO
```

```
[ ] model = YOLO("yolov8n.pt")
    result = model.train(
        data=f"{dataset.location}/data.yaml",
        epochs=50,
        imgsz=640,
        #writting out to google drive
        project=DRIVE_RUNS,
        name="exp1",
        exist_ok=True
    )

    print("Saved run to:", result.save_dir)
```

Figure 3.2.8 – YOLOv8n training configuration

Deployment of the model is shown in **Appendix A**

Analyze video using our model:

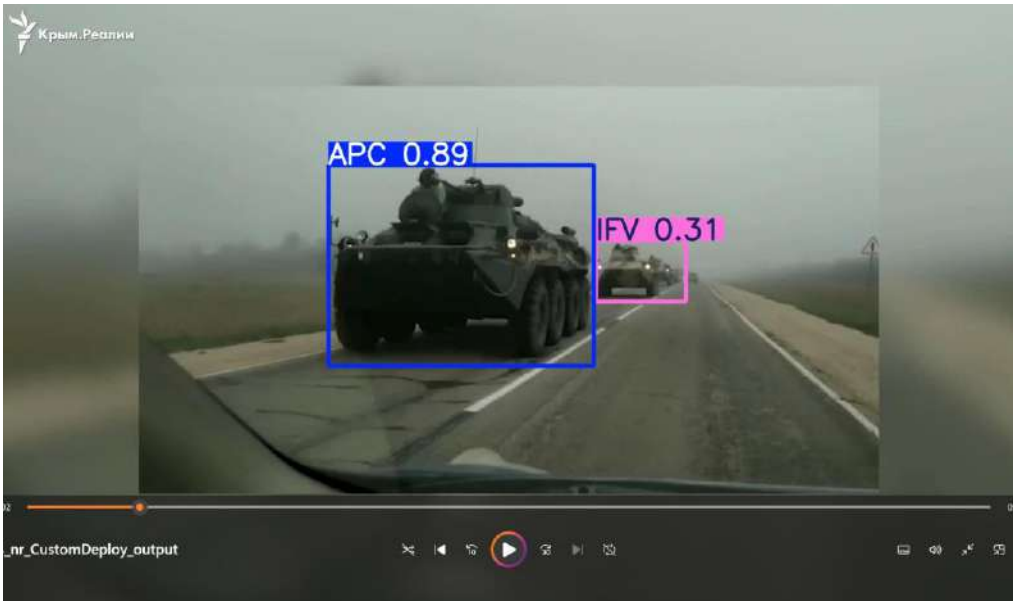


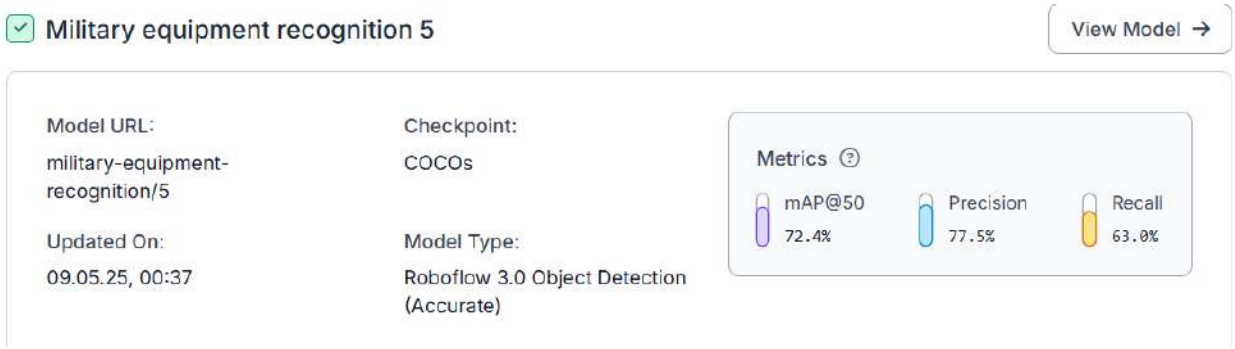
Figure 3.2.9 – Frame from our locally deployed model output of video that captures russian military convoy, posted by [22] Crimea Reality - regional news outlet of RFE/RL's Ukrainian Service.

3.2.4 Training on Roboflow

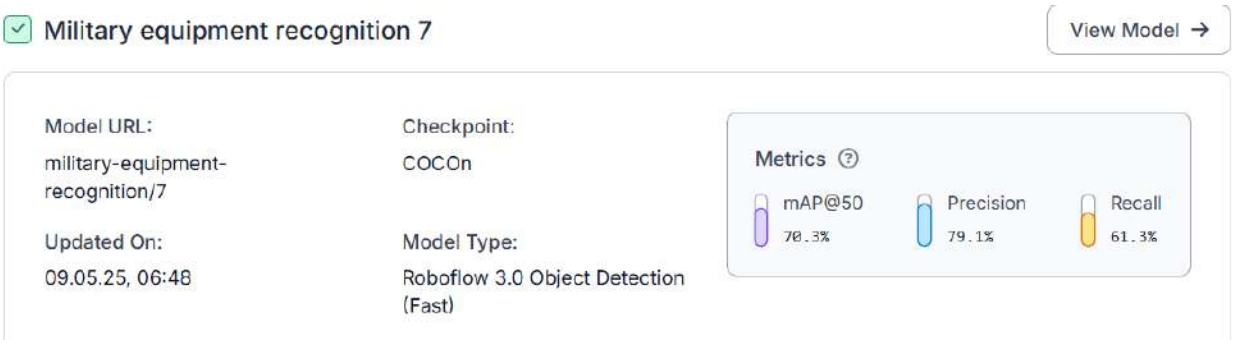
Using easy-to-use web interface Roboflow allows you saving dataset versions and train your model with several Model Architectures: Roboflow 3.0(YOLOv8 derived), YOLOv11, YOLOv12 and YOLO-NAS with a limited usage: estimated 3.8 credits per train with 50 credits a month available with free growth trial.

Trained three models(Split: 70/20/10):

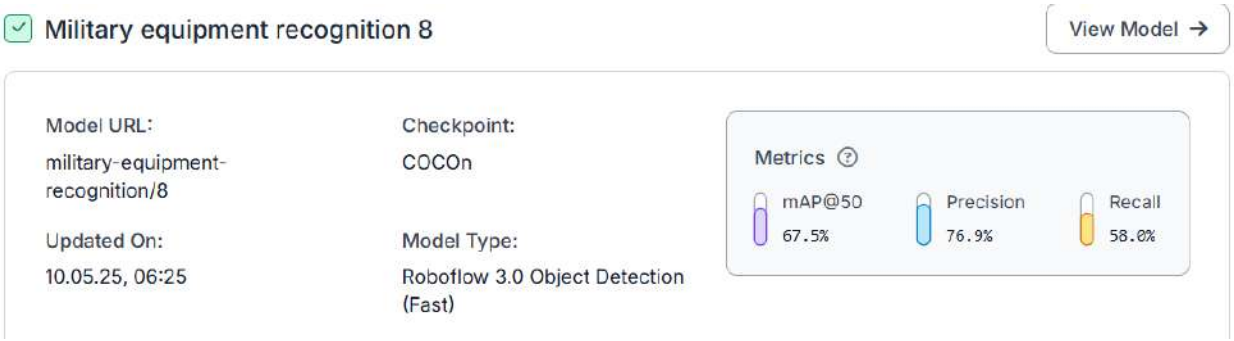
1. **Dataset V5** with 3299 images, preprocessing: auto-orient and resize 640x640 on Roboflow 3.0 Object Detection (Accurate).



2. **Dataset V7** with 7987 images(as a result of augmentation), same preprocessing: auto-orient and resize 640x640, but with augmentations: Blur(2.4px) Noise(1.84%) Grayscale(15%) Brightness (+-28%) and on Roboflow 3.0 Object Detection model ("Fast") instead of "Accurate"– referring to model size.



3. **Dataset V8** with 3299 images no preprocessing and no augmentation trained on Fast model size of Roboflow 3.0 .



Figures 3.2.10 - 3.2.12 – Models metrics trained with Roboflow

According to the [18] documentation: Roboflow Train offers various model architectures including "Roboflow 3.0 Object Detection (YOLOv8 Derived)" specifically for object detection tasks. The term "derived" suggests that while the

model uses YOLOv8 as its foundation, Roboflow has made their own adjustments to the architecture.

4 EXPERIMENTS & RESULTS

4.1 Evaluation Metrics

Model	*Dataset size*	*Augmentation*	*Preprocessing*	*Precision*	*Recall*	*mAP@50*
YOLO v8n	3299	-	Resize 640x640	0.686	0.605	0.655
YOLO v8s	3299	-	Resize 640x640	0.639	0.690	0.695
YOLO v8m	3299	-	Resize 640x640	0.707	0.570	0.659
YOLO v8s 100 epochs	3299	-	Resize 640x640	0.762	0.618	0.682
YOLO v8s no resize	3299	-	-	0.734	0.622	0.687
Roboflow 3.0 Accurate	3299 V5	-	Resize 640x640	0.791	0.631	0.703
Roboflow 3.0 Fast	7987 V7	Yes [list below]	Resize 640x640	0.775	0.630	0.724
Roboflow 3.0 Fast	3299 V8	-	-	0.769	0.580	0.675

Table 4.1 – evaluation metrics of trained models

[List below] Naming used augmentations: Blur(2.4px) Noise(1.84%) Grayscale(15%) Brightness (+-28%).

Augmentation of data with cropping parts for YOLO model is bad practice, because the model by default [23] include such augmentations. This is why we made sure no augmented data were left from initial dataset.

Taking into account YOLOv8s's advance over v8m model further research was decided to focus on three groups: 1) v8 nano, small and medium size models comparison 2) v8s group comparison of different hyperparameters influence as it was best performing stock model 3) Separately look at Custom Roboflow models as they performed the best overall.

V8s:

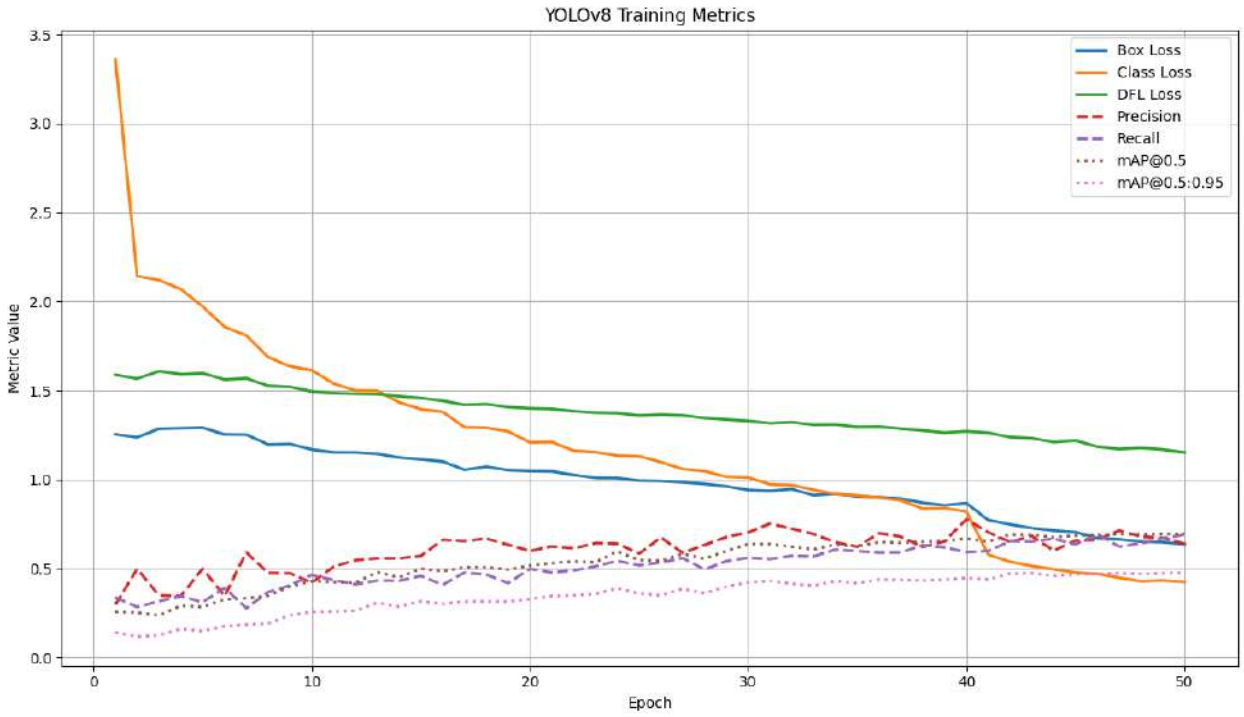


Figure 4.1.1 – v8s training metrics

V8s 100epochs:

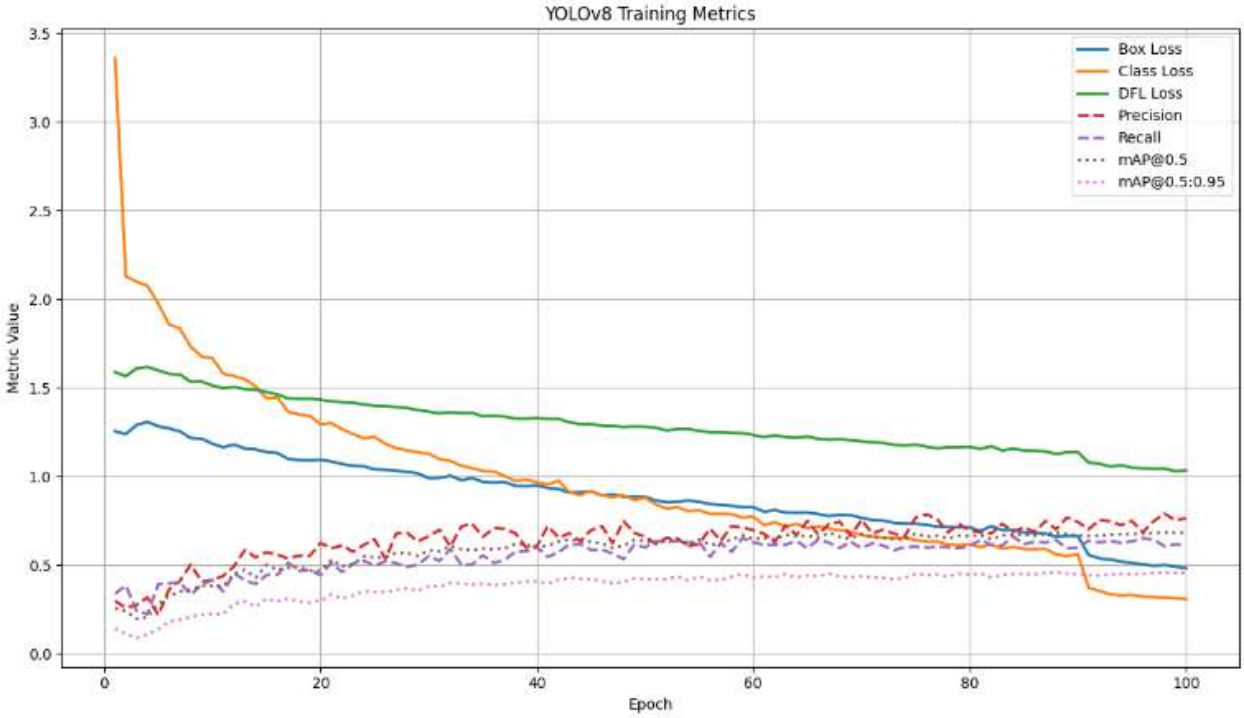


Figure 4.1.2 – v8s_100epochs training metrics

V8s_NoResize (without augmentations):

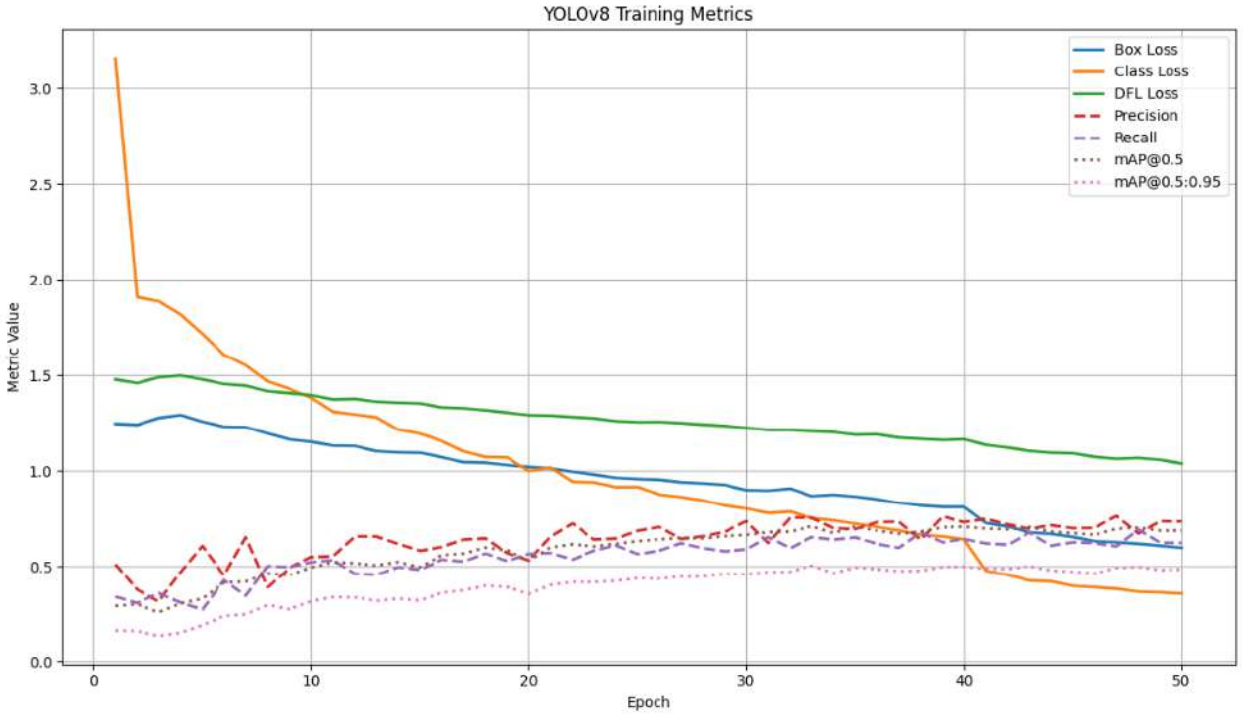


Figure 4.1.3 – v8s_NoResize training metrics

4.2 Per-Class results

YOLOv8n

	class	TP	FP	FN	Precision	Recall
0	APC	9	11	10	0.450	0.474
1	Aircraft	101	41	40	0.711	0.716
2	Anti-Aircraft System	22	7	21	0.759	0.512
3	Engineer vehicle	0	10	6	0.000	0.000
4	Helicopter	172	19	42	0.901	0.804
5	IFV	12	22	13	0.353	0.480
6	Logistics vehicle	111	43	79	0.721	0.584
7	MBT	132	23	57	0.852	0.698
8	MLRS	8	12	24	0.400	0.250
9	MRAP	14	12	15	0.538	0.483
10	RADS	6	5	8	0.545	0.429
11	SPA	6	8	19	0.429	0.240
12	UAV	0	0	0	0.000	0.000
13	non-military	86	20	23	0.811	0.789

YOLOv8s

	class	TP	FP	FN	Precision	Recall
0	APC	11	9	5	0.550	0.688
1	Aircraft	114	28	34	0.803	0.770
2	Anti-Aircraft System	25	4	9	0.862	0.735
3	Engineer vehicle	2	8	8	0.200	0.200
4	Helicopter	174	17	27	0.911	0.866
5	IFV	15	19	12	0.441	0.556
6	Logistics vehicle	119	35	44	0.773	0.730
7	MBT	135	20	48	0.871	0.738
8	MLRS	12	8	16	0.600	0.429
9	MRAP	13	13	16	0.500	0.448
10	RADS	8	3	2	0.727	0.800
11	SPA	5	9	12	0.357	0.294
12	UAV	0	0	0	0.000	0.000
13	non-military	86	20	21	0.811	0.804

YOLOv8m

	class	TP	FP	FN	Precision	Recall
0	APC	11	9	12	0.550	0.478
1	Aircraft	108	34	45	0.761	0.706
2	Anti-Aircraft System	24	5	11	0.828	0.686
3	Engineer vehicle	1	9	8	0.100	0.111
4	Helicopter	170	21	40	0.890	0.810
5	IFV	13	21	14	0.382	0.481
6	Logistics vehicle	119	35	47	0.773	0.717
7	MBT	140	15	45	0.903	0.757
8	MLRS	10	10	7	0.500	0.588
9	MRAP	13	13	11	0.500	0.542
10	RADS	8	3	6	0.727	0.571
11	SPA	7	7	23	0.500	0.233
12	UAV	0	0	0	0.000	0.000
13	non-military	82	24	14	0.774	0.854

Figures 4.2.1 - 4.2.3 – Per-class metrics for v8 type models (n-m)

YOLOv8s with different techniques:

v8s no resize

	class	TP	FP	FN	Precision	Recall
0	APC	11	9	7	0.550	0.611
1	Aircraft	107	35	42	0.754	0.718
2	Anti-Aircraft System	24	5	8	0.828	0.750
3	Engineer vehicle	4	6	8	0.400	0.333
4	Helicopter	172	19	35	0.901	0.831
5	IFV	16	18	11	0.471	0.593
6	Logistics vehicle	114	40	44	0.740	0.722
7	MBT	138	17	29	0.890	0.826
8	MLRS	11	9	15	0.550	0.423
9	MRAP	16	10	10	0.615	0.615
10	RADS	5	6	6	0.455	0.455
11	SPA	6	8	9	0.429	0.400
12	non-military	90	16	36	0.849	0.714

v8s resize 640x640 100 epochs

	class	TP	FP	FN	Precision	Recall
0	APC	10	10	13	0.500	0.435
1	Aircraft	108	34	42	0.761	0.720
2	Anti-Aircraft System	24	5	14	0.828	0.632
3	Engineer vehicle	3	7	7	0.300	0.300
4	Helicopter	167	24	29	0.874	0.852
5	IFV	12	22	6	0.353	0.667
6	Logistics vehicle	113	41	36	0.734	0.758
7	MBT	136	19	54	0.877	0.716
8	MLRS	12	8	12	0.600	0.500
9	MRAP	17	9	13	0.654	0.567
10	RADS	9	2	5	0.818	0.643
11	SPA	6	8	25	0.429	0.194
12	UAV	0	0	0	0.000	0.000
13	non-military	84	22	19	0.792	0.816

v8s resize 640x640

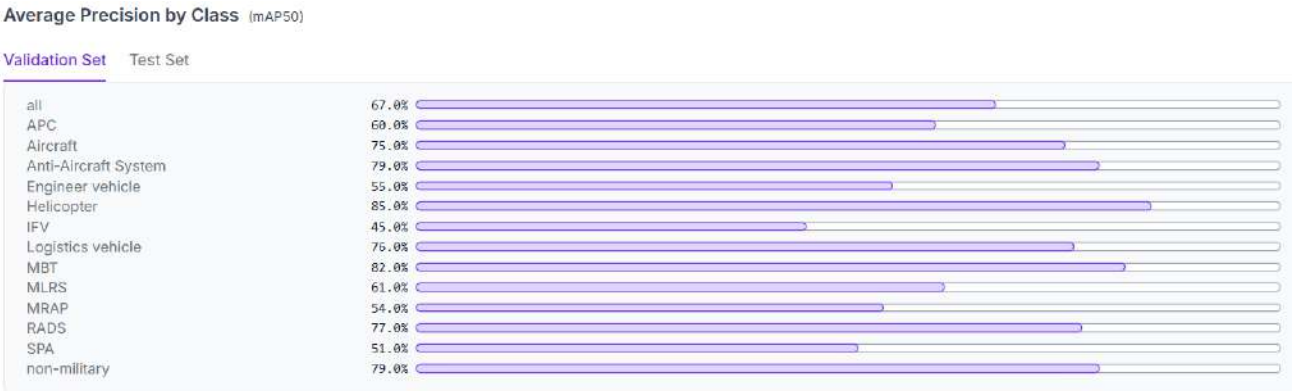
	class	TP	FP	FN	Precision	Recall
0	APC	11	9	5	0.550	0.688
1	Aircraft	114	28	34	0.803	0.770
2	Anti-Aircraft System	25	4	9	0.862	0.735
3	Engineer vehicle	2	8	8	0.200	0.200
4	Helicopter	174	17	27	0.911	0.866
5	IFV	15	19	12	0.441	0.556
6	Logistics vehicle	119	35	44	0.773	0.730
7	MBT	135	20	48	0.871	0.738
8	MLRS	12	8	16	0.600	0.429
9	MRAP	13	13	16	0.500	0.448
10	RADS	8	3	2	0.727	0.800
11	SPA	5	9	12	0.357	0.294
12	UAV	0	0	0	0.000	0.000
13	non-military	86	20	21	0.811	0.804

Figures 4.2.4 - 4.2.6 – Per-class metrics for v8s type models

Roboflow 3.0 (Accurate) resize 640x640

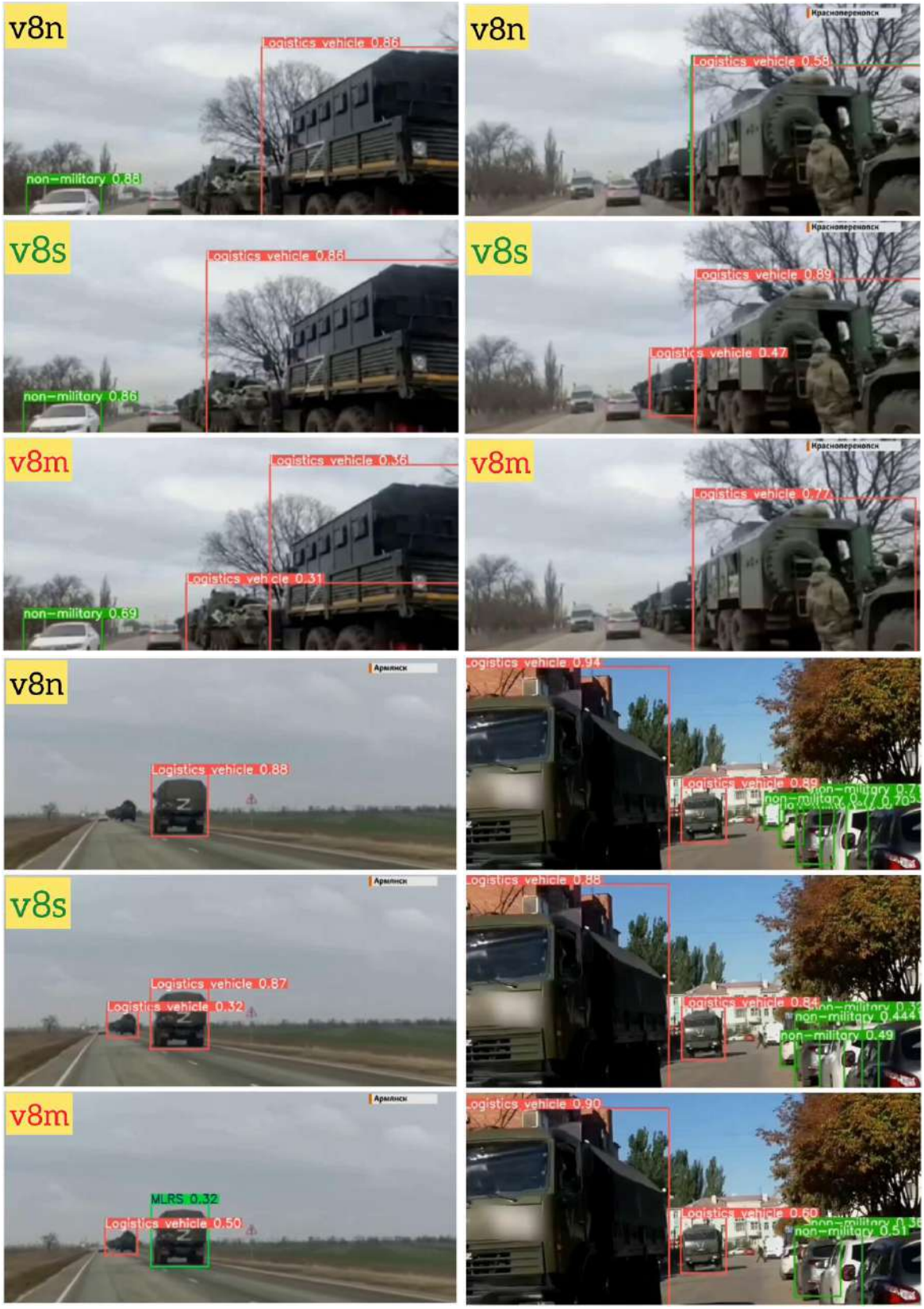
Roboflow 3.0 (Fast) Resize 640x640 Augmentation to 7987 images

Roboflow 3.0 (Fast) no Resize no Augmentations

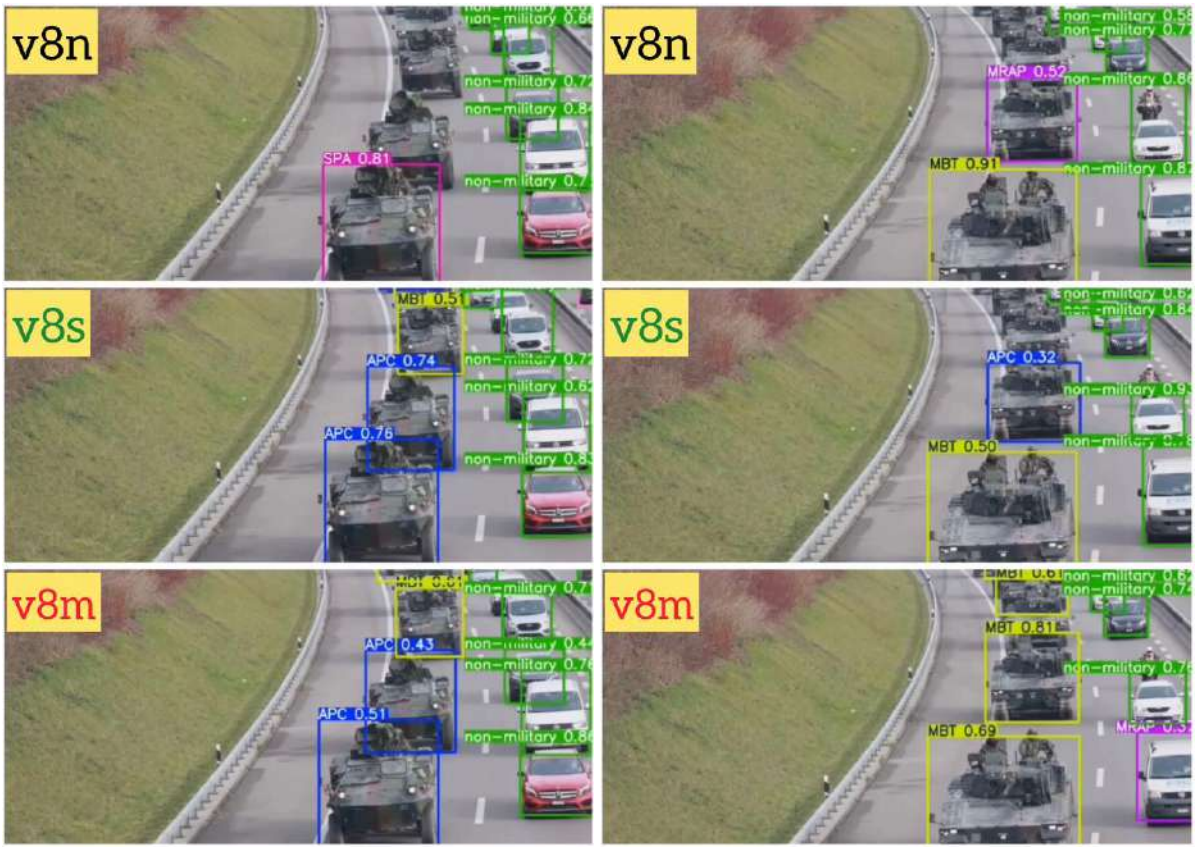


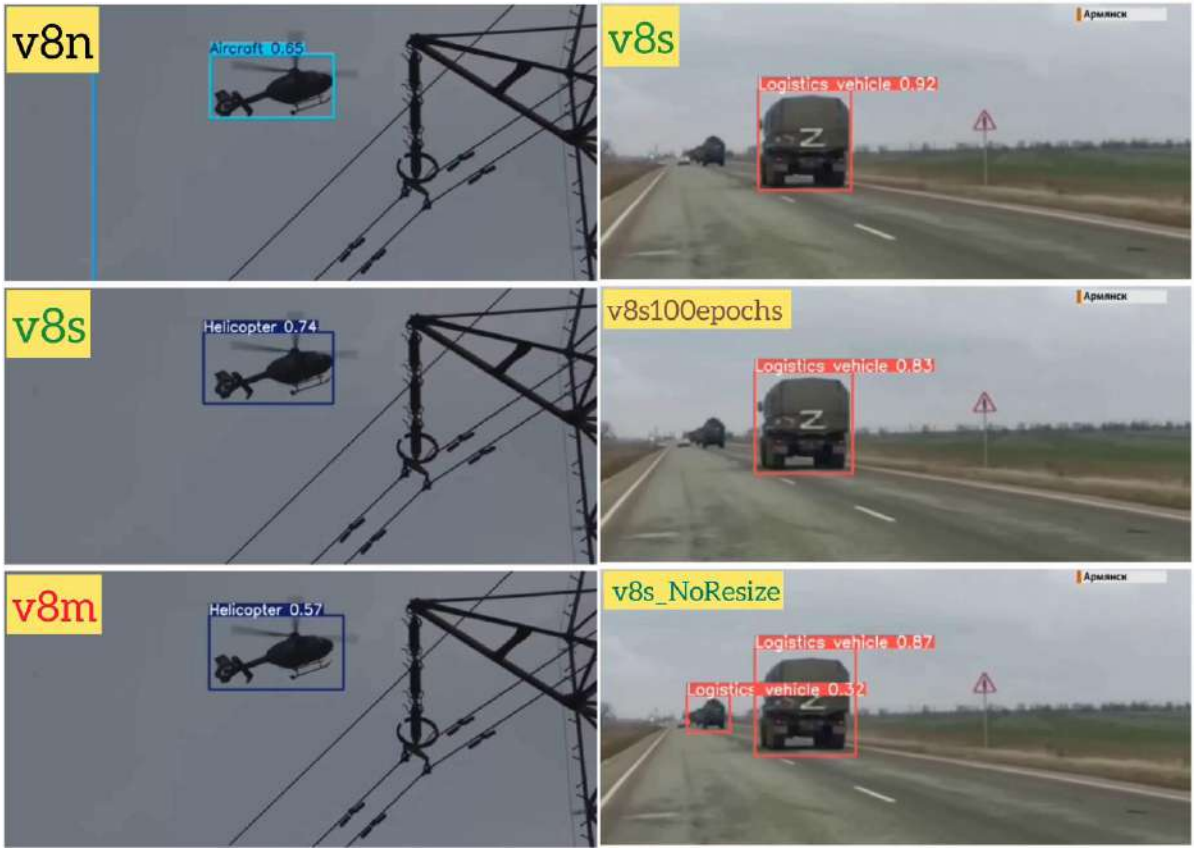
Figures 4.2.7 - 4.2.9 – Per-class metrics for Roboflow trained models

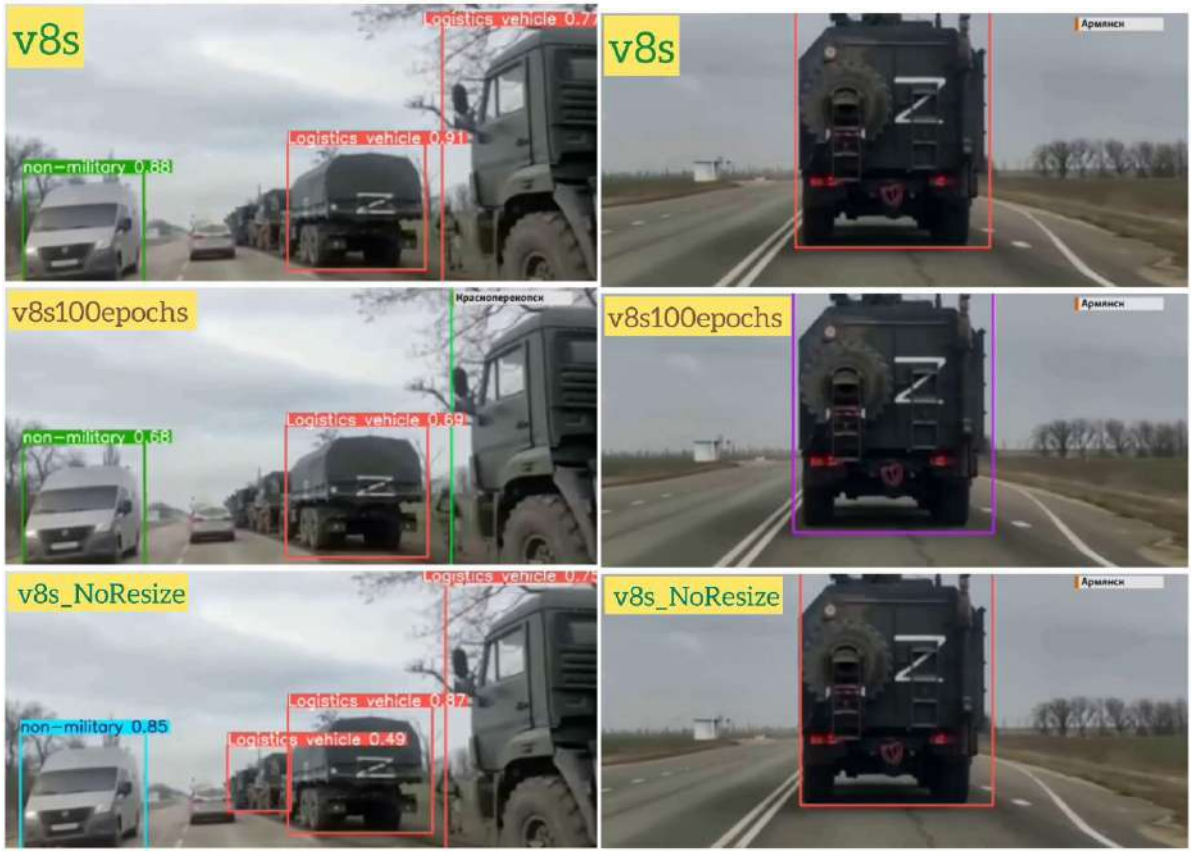
4.3 Video Inference output




v8n
Logistics vehicle 0.93
Logistics vehicle 0.92
non-military 0.91
Logistics vehicle 0.52
v8s
Logistics vehicle 0.91
Logistics vehicle 0.63
non-military 0.85
Logistics vehicle 0.73
non-military 0.78
v8m
Logistics vehicle 0.89
Logistics vehicle 0.53
MRAP 0.81
Logistics vehicle 0.54


v8n
SPA 0.81
v8n
MRAP 0.52
MBT 0.91
v8s
MBT 0.51
APC 0.74
APC 0.76
v8s
APC 0.32
MBT 0.50
v8m
APC 0.43
APC 0.51
v8m
MBT 0.81
MBT 0.69


v8n
Aircraft 0.65
v8s
Logistics vehicle 0.92
v8s
Helicopter 0.74
v8s100epochs
Logistics vehicle 0.83
v8m
Helicopter 0.57
v8s_NoResize
Logistics vehicle 0.87
Logistics vehicle 0.52


v8s
Logistics vehicle 0.91
non-military 0.88
v8s
v8s100epochs
Logistics vehicle 0.69
non-military 0.68
v8s100epochs
v8s_NoResize
Logistics vehicle 0.49
non-military 0.85
v8s_NoResize

Figures 4.3.1 - 4.3.12 – video output frames processed by corresponding model based on input videos by [22] Crimea Reality and [32] Planet First

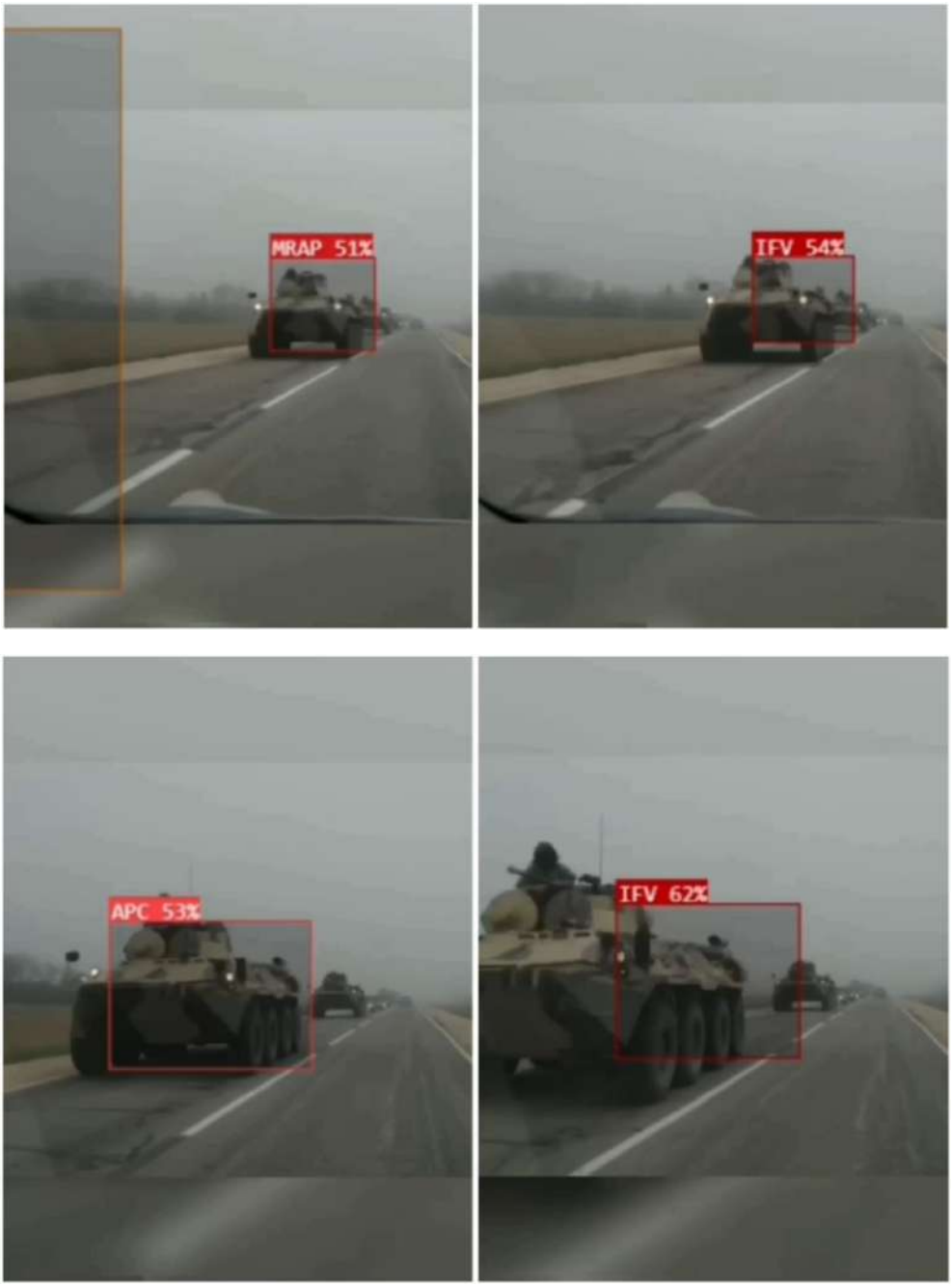
YOLO v8m's higher precision together with lower recall and its tendency to assign high confidence to incorrect detections suggests underlying issues in our training data. To check this I ran two additional v8s experiments on the same V5 dataset(as a result creating Version 8 dataset):

-v8s_E100 trained for 100 epochs (vs. 50 on other trainings) with the standard 640x640 stretch-resize preprocessing.

-v8s_NoResize trained for 50 epochs but without any stretch-resize (preserving original aspect ratios).

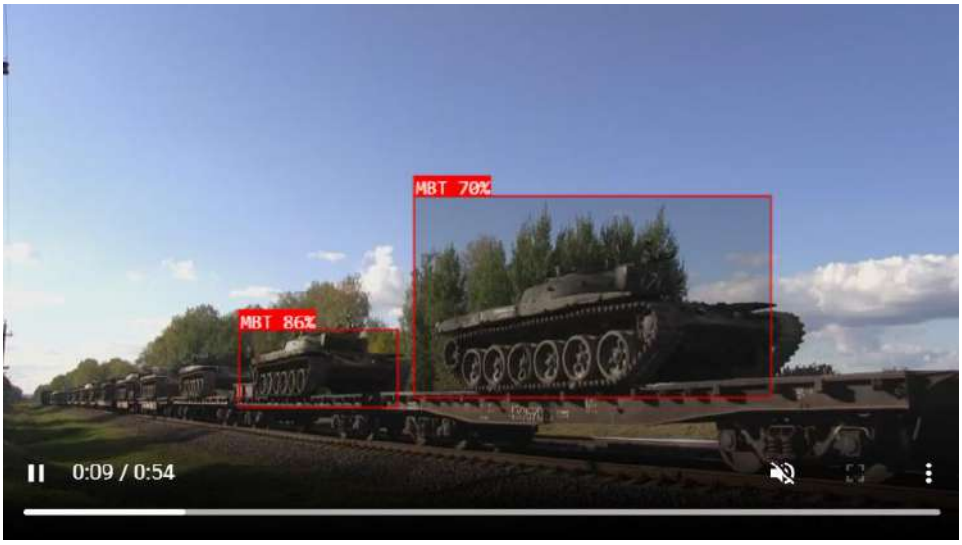
Both the metrics and the annotated video outputs confirm the hypothesis: forced image stretching amplifies overconfidence errors - especially in larger models - while removing that distortion helps reduce wrong predictions at high confidence.

RF 3.0 Fast on V7 data RF 3.0 Accurate on V5 data

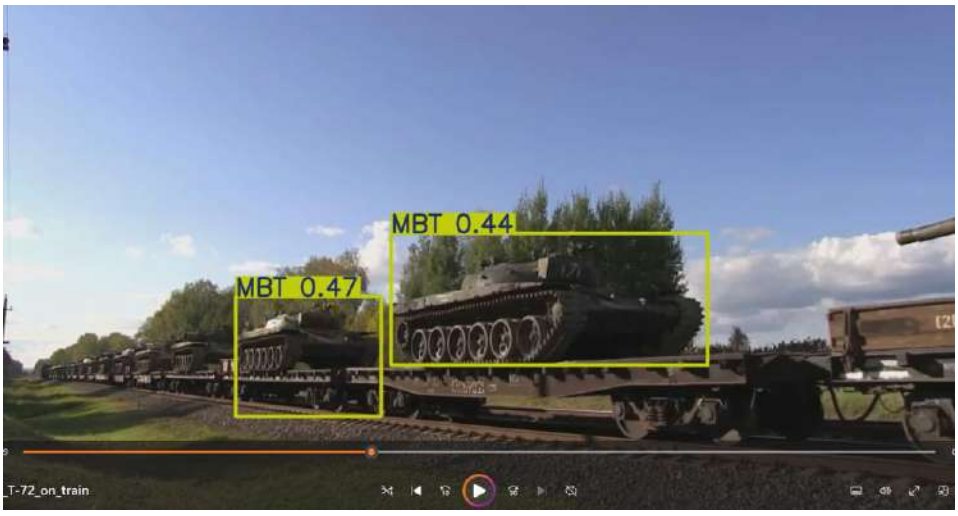


Figures 4.3.13 - 4.3.14 – video output frames processed by Roboflow 3.0 models based on input videos by [22] Crimea Reality

RF 3.0 Accurate



RF 3.0 Fast



YOLO v8s

Figures 4.3.15 - 4.3.17 – screenshots of video output processed by corresponding models based on input videos by [33] Baltic railways

Although the Roboflow 3.0 Accurate model (trained on V5, 3 299 images) and the Roboflow 3.0 Fast model (trained on the augmented V5 set -> V8, 7 987 instances) deliver almost identical detection metrics, their real-time throughput diverges sharply. According to my observations from video outputs Accurate model is better in cropping precise boxes, but it is offset by freezes and frame drops. On a machine with only 512 MB of integrated AMD Radeon graphics, the Accurate model manages about 5 fps at 100% GPU utilization, whereas the Fast model achieves around 11 fps at roughly 80% GPU load.

5 CONCLUSION

This study demonstrates that two models YOLOv8s and Roboflow 3.0 Fast with augmentation meet the core requirements for real-time reconnaissance: they run efficiently on constrained hardware, maintain a low false-positive rate on civilian vehicles, and achieve sufficient accuracy mAP@50 over 0.720.

However, inspection of batch predictions uncovered instances of 1.0 confidence on several detections - tracing back to duplicate images shared between training and validation sets. This highlights the necessity of rigorous deduplication and a more disciplined dataset curation process.

Figure 5.1 - RADS 1.0 – impossible value for fair trained model

As looking on this confusion matrix Aircraft , Helicopter, Logistics vehicle, MBT and non-military classes are clearly predicted while others are poorly distinguished:

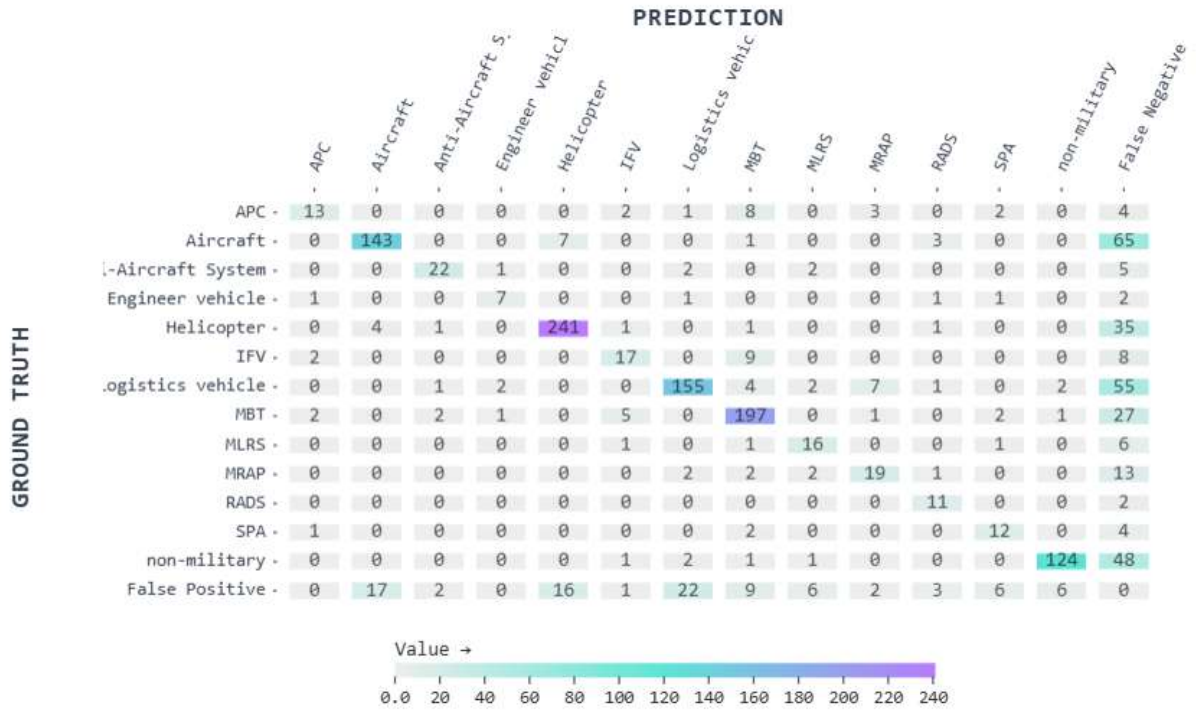


Figure 5.2 – confusion matrix of RF 3.0 Fast model trained on 7987 images

Future work should prioritize reducing inter-class confusion among visually similar categories by: expanding certain classes sample count, enforcing high qualitet of annotations; [25] Exploring targeted data augmentations & class-weighted loss functions - expand augmentation strategies.

Despite some downsides I would still insist on end-to-end workflow method I described in a Approach's [*Figure 2.1*] diagram: data gathering from [24] Kaggle, [7] RF Universe, [1] DataMandley, Roboflow CLI/SDK integration, preprocessing and versioning, Roboflow and [4] Colab training, and local video inference - is a good foundation for ongoing development.

References:

[1] Mendeley Data, “Military Equipment Recognition Dataset” (version 1) Dataset, Mendeley Data, https://data.mendeley.com/datasets/njdjkbxdpn/1 (accessed May 2025)

[2] Roboflow, “Platform for building computer vision projects”, Roboflow, https://roboflow.com (accessed May 2025)

[3] Ultralytics, “YOLO Documentation” Webpage, Ultralytics, https://docs.ultralytics.com/ (accessed May 2025)

[4] Google, “Google Colaboratory”, https://colab.google/ (accessed May 2025)

[5] Roboflow, “Annotation Tools”, Roboflow Docs, https://docs.roboflow.com/annotate/annotation-tools (accessed May 2025)

[6] D. Schultz, “dedupe.py” Script, dataset-tools (MIT License), GitHub, https://github.com/dvschultz/dataset-tools/blob/main/dedupe.py (accessed May 2025)

[7] Roboflow Universe, “Roboflow Open Database”, https://universe.roboflow.com/ (accessed May 2025)

[8] Old-Skydefender, “kek-osawl” (SPA & Anti-Aircraft) Dataset, Roboflow Universe, https://universe.roboflow.com/old-skydefender/kek-osawl (accessed May 2025)

[9] CapstoneProject, “Russian-Military-Annotated” Dataset, Roboflow Universe, https://universe.roboflow.com/capstoneproject/russian-military-annotated (accessed May 2025)

[10] ZST, “BTR” (APC) Dataset, Roboflow Universe, https://universe.roboflow.com/zst-whmmk/btr-e5iyf (accessed May 2025)

[11] TankIdentifier, “BTR80_reference; BMP3_reference; T90A_reference” Dataset, Roboflow Universe,

https://universe.roboflow.com/tankidentifier/btr80_reference
https://universe.roboflow.com/tankidentifier/bmp3_reference
https://universe.roboflow.com/tankidentifier/t90a_reference (accessed May 2025)

[12] YOLOTank, "Multi_Rocket_Launcher_ODv1" (MLRS) Dataset, Roboflow Universe, https://universe.roboflow.com/yolotank/multi_rocket_launcher_odv1 (accessed May 2025)

[13] Laura-Tsanaullailla, "Jalan-Rusak" (Roads) Dataset, Roboflow Universe, https://universe.roboflow.com/laura-tsanaullailla/jalan-rusak-y05ab (accessed May 2025)

[14] BTL, "CaseStudy3" (Radar Systems) Dataset, Roboflow Universe, https://universe.roboflow.com/btl-o3c5r/casestudy3 (accessed May 2025)

[15] Google, "Open Images Dataset V6" Dataset, Google Storage, https://storage.googleapis.com/openimages (accessed May 2025)

[16] Commander-in-Chief of the Armed Forces of Ukraine, "Order No. 140: NATO Symbology Standards", sprotyvg7, https://sprotyvg7.com.ua/wp-content/uploads/2024/12/%D0%9F%D0%BE%D1%80%D1%8F%D0%B4%D0%BE%D0%BA-%D0%BE%D1%84%D0%BE%D1%80%D0%BC%D0%BB%D0%B5%D0%BD%D0%BD%D1%8F-%D0%91%D0%A0.pdf (accessed May 2025)

[17] Fotor, "Crop Tool", Fotor, https://www.fotor.com/features/crop (accessed May 2025)

[18] Roboflow, "Train a Model Docs", Roboflow Docs, https://docs.roboflow.com/train/train (accessed May 2025)

[19] Ultralytics, "ultralytics" (Ultralitics GitHub page including YOLOv8 and metrics for whole ultralitics product family) Repository, GitHub, https://github.com/ultralytics/ultralytics (accessed May 2025)

[20] Ultralytics, “YOLO Performance Comparison” Image, Ultralytics assets, https://raw.githubusercontent.com/ultralytics/assets/refs/heads/main/yolo/performance-comparison.png (accessed May 2025)

[21] Edjie Electronics, “Comparison of YOLO V5, V8, V11 Variants”, 2024, https://www.youtube.com/watch?v=_WKS4E9SmkA (accessed May 2025)

[22] RFE/RL Ukrainian Service, “Crimea Reality” News Site, https://ua.krymr.com/ (accessed May 2025)

[23] Ultralytics, “Data Augmentations in YOLOv8” (augmentation docs), GitHub, https://github.com/ultralytics/ultralytics/blob/main/ultralytics/data/augment.py (accessed May 2025)

[24] Kaggle, “Datasets” Webpage, https://www.kaggle.com/datasets (accessed May 2025)

[25] S. Chen et al., “Scale-Aware Automatic Augmentation for Object Detection” Paper, *Proc. CVPR*, 2021, https://openaccess.thecvf.com/content/CVPR2021/papers/Chen_Scale-Aware_Automatic_Augmentation_for_Object_Detection_CVPR_2021_paper.pdf (accessed May 2025)

[26] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, “You Only Look Once: Unified, Real-Time Object Detection” Paper, *Proc. CVPR*, 2016

[27] G. Jocher et al., “YOLO by Ultralytics” Repository, GitHub, 2020 https://github.com/ultralytics/yolov5 (accessed May 2025)

[28] C.-Y. Wang, A. Bochkovskiy, and H.-Y. Liao, “YOLOv6: A Single-Stage Object Detection Framework for Industrial Applications” Paper, arXiv, 2022

[29] C.-Y. Wang, H.-Y. Liao et al., “YOLOv7: Trainable Bag-of-Freebies Sets New State-of-the-Art for Real-Time Object Detectors” Paper, *Proc. CVPR*, 2022

[30] PyTorch Team, “PyTorch”, https://pytorch.org/ (accessed May 2025)

[31] A. Géron, *Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow* 2nd ed., O'Reilly, 2019

[32] Planet First, "Large column of Swiss military Leopard Tanks on Highway in Switzerland | Pilum 22", YouTube, https://youtu.be/LG6evizFak8?si=QyvMFevtBps51cCY (accessed May 2025)

[33] Baltic railways, "Russian Soviet tanks type T-72", YouTube, https://www.youtube.com/watch?v=OIx4T18y6Dc (accessed May 2025)

[34] Drawio, "diagram on Figure 2.1.1", WebPage, https://app.diagrams.net/ (accessed May 2025)

Appendix A

Script localDeploy.py input video inference

```
import os
import cv2
from ultralytics import YOLO
import torchvision
print("torchvision version:", torchvision.__version__)

# Weights
weights = "PATH_TO\weights\best.pt"
if not os.path.exists(weights):
    raise FileNotFoundError(f"Weights not found at {weights}")

model = YOLO(weights)
model.model.cpu()

# input
video_in = "PATH_TO\input_video.mp4"
cap = cv2.VideoCapture(video_in)
if not cap.isOpened():
    raise RuntimeError(f"Failed to open video {video_in}")

fps = cap.get(cv2.CAP_PROP_FPS)
w = int(cap.get(cv2.CAP_PROP_FRAME_WIDTH))
h = int(cap.get(cv2.CAP_PROP_FRAME_HEIGHT))

# output
out_path = "PATH_TO\output_video.mp4"
fourcc = cv2.VideoWriter_fourcc(*"mp4v")
out = cv2.VideoWriter(out_path, fourcc, fps, (w, h))


frame_idx = 0
while True:
    ret, frame = cap.read()
    if not ret:
        break

 # run inference on CPU
    res = model(frame, device="cpu")[0]
    annotated = res.plot()

    out.write(annotated)
    frame_idx += 1

cap.release()
out.release()
print(f"Done—annotated video saved to {out_path}")
```