Ministry of Education and Science of Ukraine

National University of "Kyiv-Mohyla Academy"

Department of Informatics of the Faculty of Informatics

Real-Time Object Detection Algorithms for UAV Companion Computers

Text part to the thesis in the specialty "Software Engineering" - 121

Thesis Supervisor:

Senior Lecturer

Andrew Kurochkin

_____ (Signature)

"___" _____ 2025

Done by Student:

Bohdan Prokhorov

"___" _____ 2025

Kyiv, 2025

Ministry of Education and Science of Ukraine
National University of "Kyiv-Mohyla Academy"
Department of Informatics of the Faculty of Informatics

Approved
Head of Department of Informatics,
Associate Professor, Ph.D.
S. S. Gorokhovsky
_____ (Signature)
"___" _____ 2025

INDIVIDUAL TASK for thesis
of the student Bohdan Prokhorov
3rd year of the Faculty of Informatics
topic: Real-Time Object Detection Algorithms for UAV Companion Computers

The content of the PM to the thesis:
Abstract
Introduction
Related work
Approach
Solution
Evaluation
Further work and Conclusions
Bibliography

Date of issue: "___" _____ 2025

Supervisor: _____ (Signature)

Task received: _____ (Signature)

Abstract

Nowadays, various robots perform a wide range of tasks. For basic functions, it is sufficient to program the system to perform simple routine operations. However, solving more complex problems requires the development of intelligent systems with adaptive algorithms that can function effectively in changing conditions and appear in a variety of areas, from industry to everyday life.

In this work, we developed a pipeline that combines a large amount of data, selects only the necessary photos, normalizes them, and trains the YOLOv8n model on them.

As the result we got a model that can perform object detection on domain-specific data and could be deployed on a UAV companion computer to do the inference in real-time.

Contents

1 Introduction **1**
1.1 Background . . . 1
1.2 Motivation . . . 2

2 Overview of existing solutions **3**
2.1 Comparison of object detection algorithms . . . 3
2.2 Data augmentations . . . 4
2.3 UAV implementation solutions and constraints . . . 5

3 Approach **6**
3.1 Edge platform . . . 6
3.2 Architecture of the proposed algorithm . . . 6
3.3 Data collection . . . 7
3.4 Data augmentation . . . 10
3.5 Model training and optimization methodology . . . 11

4 Solution **12**
4.1 Experimental setup . . . 12
4.2 Performance metrics . . . 12

5 Evaluation **14**
5.1 Methodology . . . 14
5.2 Comparison to baseline . . . 14
5.3 Evaluation results . . . 16

6 Conclusions and Future Work **17**
6.1 Key findings . . . 17
6.2 Future work . . . 17

A YOLOv8 architecture **19**

Bibliography **20**

Chapter 1

Introduction

1.1 Background

This paper is related to the creating an object detection system for UAV companion computer to do the real-time inference to build autonomous drones. Understanding the environment is one of the most important and challenging parts of this task, as detection is at the initial stage of decision-making in autonomous systems. Low-quality inference can lead to greater uncertainty in the following steps, making these systems less reliable.

Drones looks perspective in different fields and can be applicable solve as various problems as possible. Here are some key applications and use cases that demonstrate the significant impact of drone technology:

- Agricultural sector for crop control, effective enrichment of plants with fertilizers, detection of suspicious objects in the field.
- Advanced logistics systems for rapid delivery of goods, medicines, donor organs, especially to hard-to-reach areas without significant human intervention.
- Integration of drones into urban infrastructure for monitoring, traffic control, security.
- Rescue operations where the question is about saving other people's lives and understanding the situation in hard-to-reach places can be very useful.
- Military use for reconnaissance, rescue operations, transporting supplies, and striking targets.

As we can see drones have become a universal technological solution due to their cost-effectiveness and ability to perform tasks in places inaccessible or dangerous to humans.

1.2 Motivation

Today, there exists a massive number of different solutions based on drones that are controlled by pilots, and the next logical step in their evolution is to remove the human factor.

UAVs are easily scalable technology that requires human skills to operate. The training process takes months for a pilot to learn how to perform UAV missions efficiently. Even the most experienced pilots can fail in extreme conditions, and it is a matter of mission success rate, which is considered the proportion of successfully completed missions to all missions. In an autonomous UAV, the problem of training humans may disappear if the system is sufficiently accurate and efficient. This step will significantly expand the use of these systems in all possible areas, increasing the success rates of achieving operational objectives. All you need is a domain-specific dataset and an engineer who can rerun the training pipeline to adapt the drone for a new type of usage.

Chapter 2

Overview of existing solutions

2.1 Comparison of object detection algorithms

Nowadays, object detection algorithms are widely spread across many fields and can be applied to solve various problems. Additionally, over the past few years, numerous researchers have developed various approaches, and the selection of the right one depends primarily on the task to be solved. Object detection methodologies can be categorized into distinct paradigms: multi-stage detectors, one-stage systems, and transformer-based models. Each of these approaches has its pros and cons, striking a balance between accuracy and computational demands. These factors are crucial to our problem of developing a real-time UAV on-board system because there are several limitations on inference time, consumption efficiency, and computational complexity.

The R-CNN family of algorithms, which includes R-CNN[1], Fast R-CNN[2] and Faster R-CNN[3], are a prime example of **multi-stage detectors**. They are an approach that splits the inference process into two parts, each with its model. The first part serves as the backbone, a feature extractor that generates hierarchical patterns of varying levels of abstraction. This information is fed into the RPN block for generating bounding boxes where the searched objects should be located. After this, an NMS operation is applied to eliminate redundant and overlapping proposals, leaving only the most likely candidates. The final step is to pass the filtered regions of interest to the classifier, which determines the object categories and refines the bounding box coordinates. Multi-stage detectors are known to be relatively slow, but the results are better than those of single-stage detectors in terms of accuracy and precision.

One-stage detectors represent another view on the object detection task, because of a different architecture, where inference is obtained in a single pass through the neural network, which significantly exceeds the processing speed. Prime examples of the one-stage approach include models such as YOLO[4], SSD[5], and RetinaNet[6]. These models achieve a compromise between speed and accuracy, which makes them attractive

for real-time production. The latest versions of YOLO models have come remarkably close to the accuracy of multi-stage models, while offering significantly better efficiency. Additionally, single-stage detectors struggle to detect small objects in images, which can be a disadvantage for real-time video captured from high altitudes.

Transformer-based models are an approach developed within the Facebook AI Research and represent a significant breakthrough in object detection called DERT[7]. They abandoned many of the approaches that single-stage and multi-stage models had, such as NMS and anchor boxes, in favor of the transformer architecture that Google originally proposed for natural language processing tasks in its paper Attention is All You Need[8]. This pipeline uses extracted features from the backbone and passes them as a sequence to the encoder-decoder layer. The decoder utilizes learnable object queries, which are a fixed set of embeddings that can be interpreted as potential objects to be detected. These object queries interact with the encoded image features through cross-attention mechanisms. The final decoded information is then processed by a feed-forward network to directly predict bounding box coordinates and class probabilities for each object query.

In further research, we decided to proceed with the YOLO family algorithms due to the inability to launch multi-stage and transformer-based models on board the UAV drone for real-time inference. This is because, in our task, efficiency is important and can significantly impact the overall performance of the system.

2.2 Data augmentations

Data augmentation is one of the most effective ways to improve the performance of neural networks and their ability to generalize. They can be applied to any domain or type of data. In image processing, the core idea of this approach is to take an image and make it as different as it can be, depending on the specific task. It can be useful to consider the common image degradation caused by low-quality drone cameras and apply them. This allows models to avoid overfitting on training data and exposes models to a wide range of variations they may encounter in real life.

Color space augmentations allow us to manipulate images on a pixel-wise level, performing transformations on the color distributions. Geometric augmentations are typically applied to modify existing photos, simulating different camera angles, viewing distances, and perspectives, thereby enhancing the model's ability to recognize objects regardless of their orientation or position within an image. In [9] the authors introduced new geometric augmentation called mosaic. This technique combines four training images into one during the training process, allowing the network to learn to detect objects at different scales and in varied contexts simultaneously.

2.3 UAV implementation solutions and constraints

The choice of a platform for detecting objects from a drone is critically important, as it determines the accuracy of target recognition, data processing speed and the ability to operate in real time, which can be critical when building autonomous drone-based systems. A properly selected system can significantly increase mission efficiency and minimize risks for the best value. The latter factor is also important due to the further possibility of scaling the number of drones to perform various tasks such as logistical, military, and rescue operations.

In [10] the researchers introduced the MSPP-FPN module and proposed a four-scale detector based on Darknet59 to improve the detection of small objects. The authors of [11] based their solution on the YOLOv3 architecture and developed a lightweight algorithm that analyzes actual flight data to determine the best parameters for better efficiency. In [12], the authors proposed the EdgeYOLO architecture, which was developed for the autonomous driving problem, but due to its optimizations for edge computing devices, it can be effectively applied to various resource-constrained scenarios, including unmanned aerial vehicles. Proposed light-weight design with a new decoupled head, which helps to improve inference speed with little precision loss. Later in [13] EdgeYOLO architecture was dramatically compressed to run it on GAP8 processor with 8 RISC-V cores. They applied hardware-aware approach to optimize the already existing architecture and improved efficiency up to 76%.

Most of these papers utilize NVIDIA Jetson[14] platforms for edge deployment due to their GPU acceleration and compatibility with frameworks like TensorRT[15]. These chipsets are one of the best solutions, though their high performance often comes with high price you should pay for them. There are other solutions that can be based on a companion computer. We can use for object detection processors such as Rockchip[16] that offer decent performance at a lower price, specially designed Hailo AI[17] accelerators, Raspberry Pi[18] combined with neural compute bars, and Google Coral Edge TPU[19] devices that provide efficient machine learning with optimized TensorFlow Lite[20] models.

Chapter 3

Approach

Our research aims to develop a system that can identify various military and civilian objects on a companion computer in real-time. To achieve this, we need to select state-of-the-art models and collect a comprehensive dataset for our task. As a result, we have developed a fully retrainable pipeline with all the required steps to train models using a large number of diverse datasets with different object classes.

3.1 Edge platform

After weighing factors such as the price of the device and the ease of integrating machine learning models into a peripheral, we chose Rockchip chips as the platform for our companion computer. We selected the Rockchip board processor as the main chipset. The dedicated NPU module can deliver up to 6 TOPS (Tera Operations Per Second), making it ideal for real-time object detection applications. When this work was written, the latest version of YOLOv8 with Rockchip integration was available.

3.2 Architecture of the proposed algorithm

Due to the selected companion computer platform, we decided to move on with the YOLOv8 nano model. It is the lightest-weight architecture from the YOLOv8 family and can be easily deployed in edge environments. Ultralytics released YOLOv8[21] on January 10th, 2023. Its developers did not provide the original paper, but we can observe the key concepts in [22].

The architecture can be divided into three logical blocks, each corresponding to a specific task. The backbone module is a feature extractor based on CSPDarknet53, which uses C2f (Cross-Stage Feature Fusion) connections to improve information transfer between layers and feature learning. This block is responsible for extracting multi-level image features and is designed to reduce computational costs while still having good

accuracy results. SPPF enhances the network's receptive field and captures objects of various scales.

The neck module uses PANet (Path Aggregation Network) to refine the information obtained from the previous block and improve multi-scale feature detection using information from different layers of the model. C2f blocks are also used here because of their advantages in multi-level feature extraction quality and efficiency.

The last module is the head, and its idea is to generate final predictions about the bounding boxes and the classes within them. YOLOv8 uses an anchor-free approach to bounding box prediction, which simplifies the prediction process by eliminating the need for predefined anchor boxes.

In A.1, we can see the holistic model architecture. The smallest nano and small versions may still have difficulty detecting small objects, but they offer a better balance between efficiency and accuracy than the larger YOLOv8 variants, which cannot run on UAVs in real time.

3.3 Data collection

Our approach is to create our own dataset from publicly available data collections that contain military objects captured from aerial views on the Internet and run it through a pipeline to extract only the data we need for specific tasks. The primary challenge in this task is to collect as much qualitative data as possible, as such data is usually not publicly available, being a valuable asset that can be harmful in the wrong hands.

This pipeline consisted of the following steps:

1. Creating a knowledge base with all the information we want to have about our datasets. This list consists of general data like links to data, publishers, size, median image ratio, classes, images, size of data, and short descriptions.

   Also, it has functional columns - "use", "is_rejected", and "is_ignored" classes. The "use" column determines whether we are taking the dataset for further processing in this training run. The "is_rejected" class marks images that are always discarded from the dataset, regardless of other classes present. The "is_ignored" class can only discard a label from an image if there are other classes in the image. If image contains only ignored class in the image, it will be discarded. All datasets information is stored in datasets.csv file which is read by filtering scripts. This approach to storing information about datasets may not be the most convenient, but we decided to move with it at this MVP stage of our project.

2. The next step is merging all datasets where the column "use" equals TRUE into one datasets.csv file with information about each image.

3. Performing filtering by "is_rejected" and "is_ignored" classes

4. For each image in the dataset, a unique hash will be generated to capture its visual features. Perceptual hash was calculated with ImageHash[23] Python library, and it presents unique fingerprints of selected images. These hashes will be nearly identical for visually similar images, allowing us to find photo duplicates. It is an important step in our work because this step can reduce the use of computing resources during training and prevent our model from introducing bias by overrepresenting certain examples.

   Subsequently, we will use a brute force approach to calculate the Hamming distance[24] between every possible pair of images based on their respective hash values. This comparison will allow us to identify similarities and relationships among all images within the collections and mark them in the "is_duplicate" column and remove them from the dataset. Based on the histogram 3.1, the threshold was set at 10, so any image pairs with a Hamming distance below 10 can be considered duplicates in our dataset.

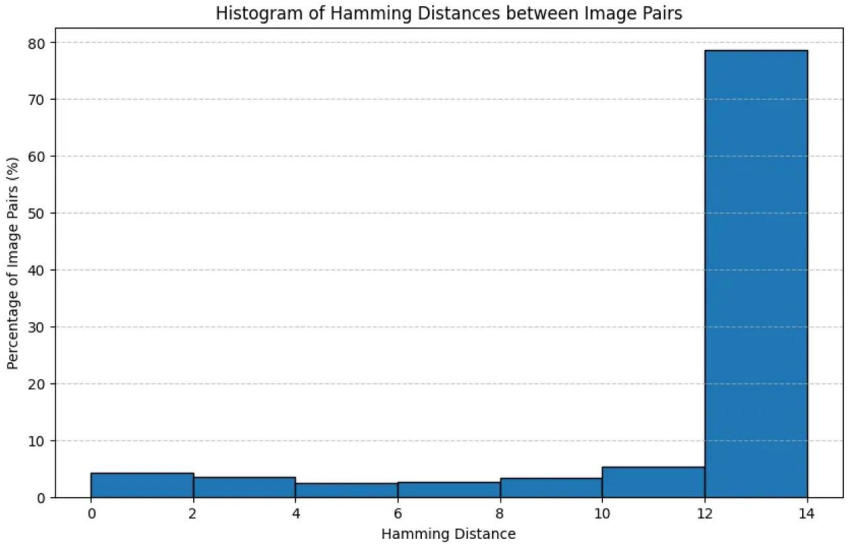

Figure 3.1: Histogram of Hamming distances between Image Pairs.

   To approve the threshold decision, we manually reviewed several sample image pairs with different Hamming distances to confirm the algorithm's accuracy. Here 3.2 we see an example of how the algorithm can find similar images despite any running lines, labels, and logos that are superimposed on top of the original image.

5. On this step, we need to change labels from the original to ours to reduce the number of classes, because there is a problem that in different datasets, the same objects can have different names and IDs. To solve this problem we created 2 .csv files - classes_mapping.csv and classes.csv. First file corresponds for knowledge base where

Figure 3.2: Example of two duplicates found by the algorithm.

we store original class name, dataset from which this image is and new name which we use in next steps.

After removing duplicates, unwanted classes, and selecting only our target datasets, the final dataset became 4 times smaller. Additionally, we reduced the number of unique classes from 92 to 8 by implementing a class remapping process.

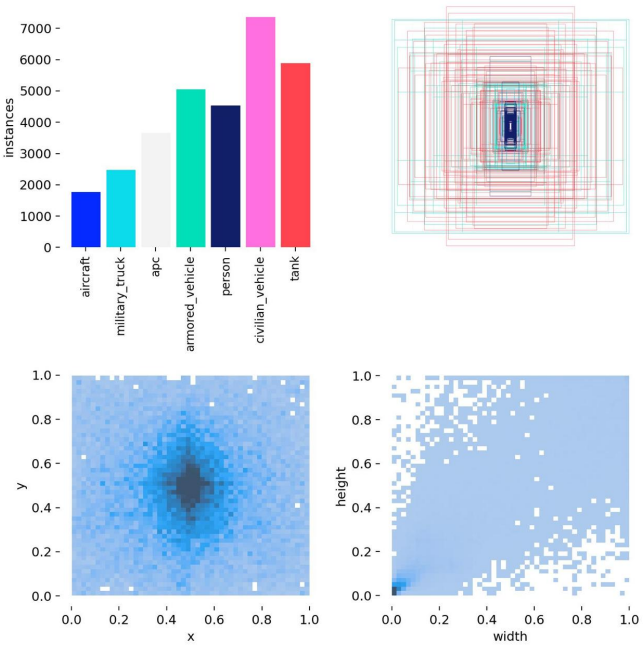

Figure 3.3: Class Distribution and Bounding Box Analysis.

Our detection model focuses on detecting military objects, but it is extremely important to create a highly reliable system based on this solution for the detection of both civilian and military targets in order to better understand the state of affairs during the execution of various missions.

We can see four plots 3.3 where the distribution of instances by class and their spatial characteristics in our dataset are shown. A histogram represents the number of instances

across eight classes, where civilian_vehicle has the highest count and aircraft the lowest. Although the civilian classes have the most instances, in the top right visualization, we see that they have the smallest bounding boxes because they were taken from the VisDrone[25] dataset. It is characterized by having many small objects in the images, while most military images do not have a large number of instances per image. The heatmap at the bottom left shows where bounding boxes are typically located, and we can see that they are usually concentrated in the center of the image. The heatmap at the bottom right shows that we have a strong concentration in the lower left corner, indicating that most objects in the dataset are relatively small compared to the image size.

3.4 Data augmentation

To build a reliable and effective object detection system for UAVs, we need to understand the potential problems and difficulties we face in this task in order to choose the most suitable augmentations for our task.

Aerial imagery datasets, usually captured from high-end drones that allow obtaining images in high resolution, represent the best possible quality of images we can obtain for now. Installing an expensive camera on a UAV board can lead to difficulties in scaling up and easily deploying autonomous drone technologies to perform various tasks.

Augmentation methods for different types of image degradation allow us to simulate real-world conditions where perfect images are not available. To apply them we use open-source image augmentation library called Albumentations[26]. It uses the OpenCV library under the hood, which allows it to perform complex transformations without much complexity through a simple API. It works with all popular deep learning frameworks and has a direct integration into the Ultralytics[27] library to create our object detection model.

We discovered different image transformations in [28] and tried to find the most effective techniques for our specific research task and conditions.

- **Downscale** - changes the quality of the image by downscaling and upscaling back.
- **ImageCompression** - changes the quality of the image by applying conversions into different file types. Here we can can adjust the quality range and choose between JPEG or WebP compression formats.
- **ShotNoise** - adds random noise to the image, which is common when taking photos in low light or on very low-cost sensors.
- **GaussianBlur** - simulates out-of-focus or motion-blurred images, improving the model's ability to generalize to blurry inputs.

Figure 3.4: An example of using all of the listed augmentations on a photo of a dog

3.5 Model training and optimization methodology

To train our object detection algorithm, we utilized YOLOv8 with pre-trained weights. YOLOv8 model can be used to perform different tasks in computer vision, it can be used in detection, segmentation and determination.

We should set it to detection mode and then set the training process to 100 epochs. For our model optimization, we used SGD (Stochastic gradient descent) with a learning rate of 0.01 and momentum of 0.9, configuring three distinct parameter groups: 57 weights with no decay, 64 weights with 0.0005 decay, and 63 bias parameters with no decay. These parameters are set during training by default in the standard version of the YOLOv8 model.

Transfer learning techniques allow us to adapt the base model to new tasks faster and reduce the need for large data. Also, the warm-up is set to the first 3 epochs to stabilize the training at the beginning of training.

IoU measures the overlap between the true label and the predicted bounding box. It is used to measure the performance of the model's prediction and its ability to find objects in the image correctly. In YOLOv8, the IoU threshold for NMS (Non-Maximum Suppression) is set to 0.7. It it also used from IoU loss computation during training.

Focal loss is used for unbalanced datasets because of its ability to give more attention to classes that are difficult to classify, improving overall performance.

Chapter 4

Solution

4.1 Experimental setup

We train all iterations of YOLOv8 on Lighting AI[29] - platform that provide a wide range of different CPU and GPU environments for computing needs. Under the hood it has the entire list of modern cloud platforms. Our model was trained on an NVIDIA L4 Tensor Core GPU[30], which is based on AWS and available to us through Lighting AI Studio. To do all data preprocessing steps we used a machine with 4 CPUs to effectively pass large datasets through our pipeline.

Tracking all model metrics and losses for the trains and validation sets is done using TensorBoard[31]. This is a visualization tool developed by TensorFlow that allows to track the model training process using various intuitive graphs in real time.

As a result of each YOLOv8 training run, we obtain a last and best trained models along with detailed evaluation metrics per each epochs, evaluation metrics for each class and basic EDA (exploratory data analysis) for the dataset on which the model was trained.

4.2 Performance metrics

Understanding of metrics always is an important step because in any machine learning task, it helps us to better understand strengths and weaknesses of our trained model to do the analysis that helps to improve model performance. The metric we track in object detection task is mAP (Mean Average Precision). The metric calculation consists of the following steps:

1. Run inference on test or validation dataset to get model predictions. As s result we obtain bounding boxes, confidence scores, and class labels.

2. Calculate IoU for each predicted box. A prediction is considered a True Positive if the IoU score is above the defined threshold and the predicted class matches

the ground truth. Otherwise we can get True Negative, False Positive and False Negative predictions

3. Calculate precision and recall scores and build curve made of these metrics with different thresholds. Precision can be calculated by dividing the True Positive value by all positive predictions. Recall can be calculated by dividing the True Positive value by all actual positive instances.

4. Find the average precision (AP) by calculating the area under the Precision-Recall curve. AP should be calculated for each class separately.

5. Calculate mAP by finding the mean across all AP for different classes in dataset. In YOLOv8 we have two types of mAP metrics - mAP@0.50 and mAP@0.50:0.95. mAP@0.50:0.95 computes the metric by averaging over multiple IoU thresholds ranging from 0.50 to 0.95, while mAP@0.50 calculates the metric using a fixed IoU threshold of 0.50. Since mAP@0.50:0.95 includes stricter overlap criteria, it is usually lower than mAP@0.50 and provides a more reliable indicator of the model's overall detection performance. In the next step, our main metric is mAP@0.50:0.95.

Chapter 5

Evaluation

5.1 Methodology

The evaluation step is crucial for comparing the performance of different YOLOv8 versions. We decided to compare two models to understand their strengths and weaknesses, and how our training and augmentation techniques improved the model's ability to make the correct inferences.

A deep understanding of results may be the key to improvements and the next iterations of the object detection model will take into account the problems. The first is our base model without any domain-specific additions, while the second contains image distortion methods. Also, we compare our solutions to out-of-the-box YOLOv8 which comes with the Utralytics library and trains on the COCO[32] dataset, which is commonly used to compare models in computer vision tasks.

mAP is the main metric, but other metrics such as precision, recall, and F1-score provide a more comprehensive evaluation of model performance.

For your practical verification, we also ran all three models on test video footage and conducted a qualitative evaluation of the detection results, which gave us an understanding of how these models can perform real-world tasks.

5.2 Comparison to baseline

Starting with a model trained on COCO dataset we have following results

Starting with a model trained on the COCO dataset, we obtained the following results, which established our baseline performance metrics. The pre-trained YOLOv8n model achieved the following results for all 30 classes included in the COCO dataset. In the original documentation mentioned that YOLOv8n has mAP@0.50:0.95 score of 37.3% on validation data.

Class	Images	Instances	Box(P	R	mAP50	mAP50-95):
all	3906	8128	0.783	0.649	0.72	0.504
aircraft	323	427	0.844	0.835	0.896	0.667
military_truck	396	632	0.741	0.666	0.74	0.541
apc	698	905	0.743	0.767	0.814	0.612
armored_vehicle	1131	1318	0.881	0.745	0.83	0.603
person	132	1327	0.619	0.145	0.232	0.103
civilian_vehicle	389	2016	0.833	0.563	0.652	0.371
tank	1149	1503	0.822	0.82	0.879	0.631

Speed: 0.2ms preprocess, 1.9ms inference, 0.0ms loss, 0.5ms postprocess per image

Figure 5.1: Model performance metrics.

Our first model iteration has the following metrics on validation data. This grid 5.1 of different metrics can give us valuable insights into our model's performance across different object classes and detection capabilities.

Model demonstrates strong overall performance with an mAP@0.50:0.95 score of 50.4% and this result is above our baseline obtained on COCO dataset. Military objects are detected very well and drive the metric up, while people and civilian vehicles are the weakest classes, having the worst results in inference. The letter R in the plot corresponds to the completeness value, and low values opposite people and civilian vehicles mean that the model cannot see them. This is probably because most instances of this class are small, and the nano version of YOLO struggles with finding small objects in the photo. We can also observe confirmation of this hypothesis on the normalized and non-normalized confusion matrices 5.2.

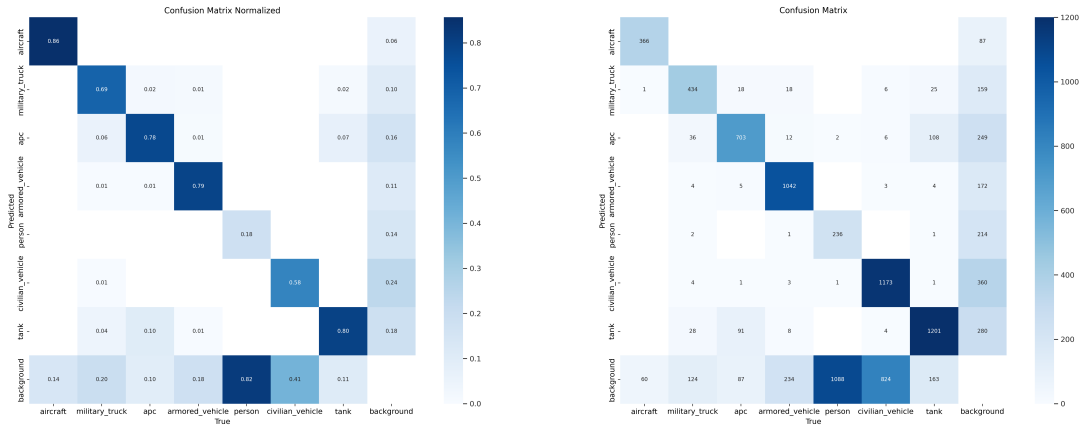

Figure 5.2: Confusion matrices for the model without domain-specific augmentations

Over 80% of people were not recognized in the verification data. A similar but less critical situation with civilian cars where the model does not detect 40% of instances.

In our second model with augmentations that simulates different distortions and noises the bast model has mAP@0.50:0.95 score of 48.9%. This is not what we expected to get, as we applied augmentation to improve the model's generalization ability, but this did not happen, and the model performed worse.

Class	Images	Instances	Box(P	R	mAP50	mAP50-95):
all	3906	8128	0.745	0.643	0.7	0.49
aircraft	323	427	0.824	0.81	0.873	0.643
military_truck	396	632	0.702	0.671	0.727	0.531
apc	698	905	0.694	0.779	0.798	0.599
armored_vehicle	1131	1318	0.848	0.755	0.83	0.605
person	132	1327	0.553	0.137	0.191	0.0888
civilian_vehicle	389	2016	0.796	0.544	0.619	0.352
tank	1149	1503	0.8	0.806	0.862	0.61

Speed: 0.2ms preprocess, 2.0ms inference, 0.0ms loss, 0.6ms postprocess per image

Figure 5.3: Model with domain domain-specific augmentations and its performance metrics.

In this grid 5.3 of metrics, we see that the augmented version of the model has all the same problems as the first model without augmentations. Confusion matrices 5.4 also show a similar result to the first model, but the metrics for all classes have dropped slightly.

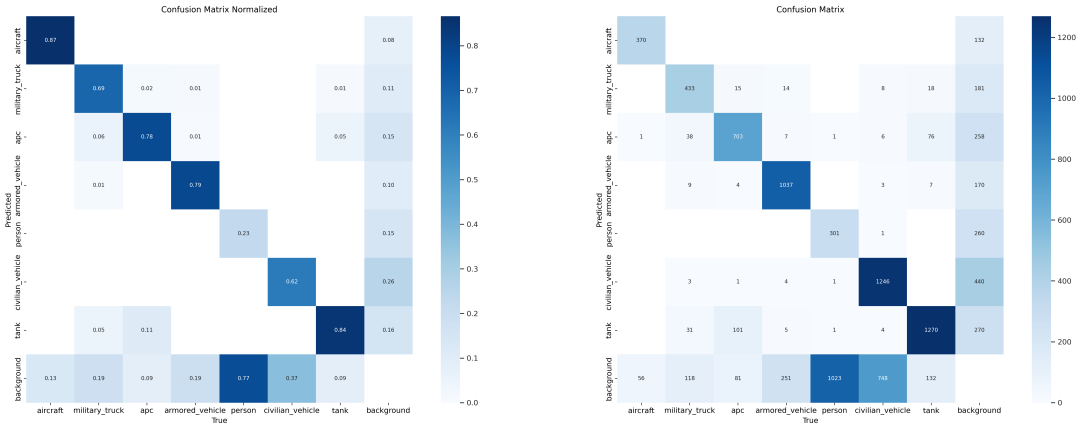

Figure 5.4: Confusion matrices for the model without domain-specific augmentations

5.3 Evaluation results

The performance dropped as we degraded the quality of the photos in the dataset, and the model started getting fewer features, making it harder to find objects in images. Perhaps this model would work better on a real-time video stream from UAV.

For the two models, we got better performance than on the COCO dataset. Although this is an incorrect comparison because datasets are different, it gives us a basic understanding that the model works well on our data.

Chapter 6

Conclusions and Future Work

6.1 Key findings

The research conducted in this study led to the development of a model that can effectively perform object detection tasks. The system is designed as an MVP of a YOLO family training pipeline for performing domain-specific tasks, which can then be used to analyze information entering the drone camera in real time. The model is successfully trained and exported to Rockchip format for running inference on the edge devices.

There is a problem with detecting small objects that is being actively addressed by the research community, and in subsequent iterations of YOLO this problem is being solved in small steps.

A pipeline was also created to merge different datasets and select only the instances we needed, removing all duplicates or redundant images.

6.2 Future work

The list of improvements can be applied and researched after receiving the results of the evaluation stages.

1. Small object detection is a general problem of YOLO family object detection algorithms. As we decided to build system that should be implemented into autonomous UAV systems, we should find different solutions to improve small size object detection performance. It can be architectural changes of the base YOLOv8 model that have a better ability to find small targets in an aerial real-time video stream. Also, custom loss functions can be implemented that will penalize classes with the smallest average size.

2. Approaches to finding the best possible additions to solve the problem can be from the device side. The technical characteristics of the devices can be taken as a basis,

and the data can be adapted to them as much as possible to reduce the gap between the dataset and the real-time video stream. It can also have the problem that the metric with augmentations is lower, but our goal is to build a robust system that performs well in real-world conditions rather than optimizing for specific custom dataset.

3. Deploy our model to the companion computer. The model inference is performed entirely on an NVIDIA L4 Tensor Core GPU, and in the future we want to track all performance and efficiency metrics of the models on the edge device to better understand the quality of our object detection model and its potential complexity in real time.

Appendix A

YOLOv8 architecture

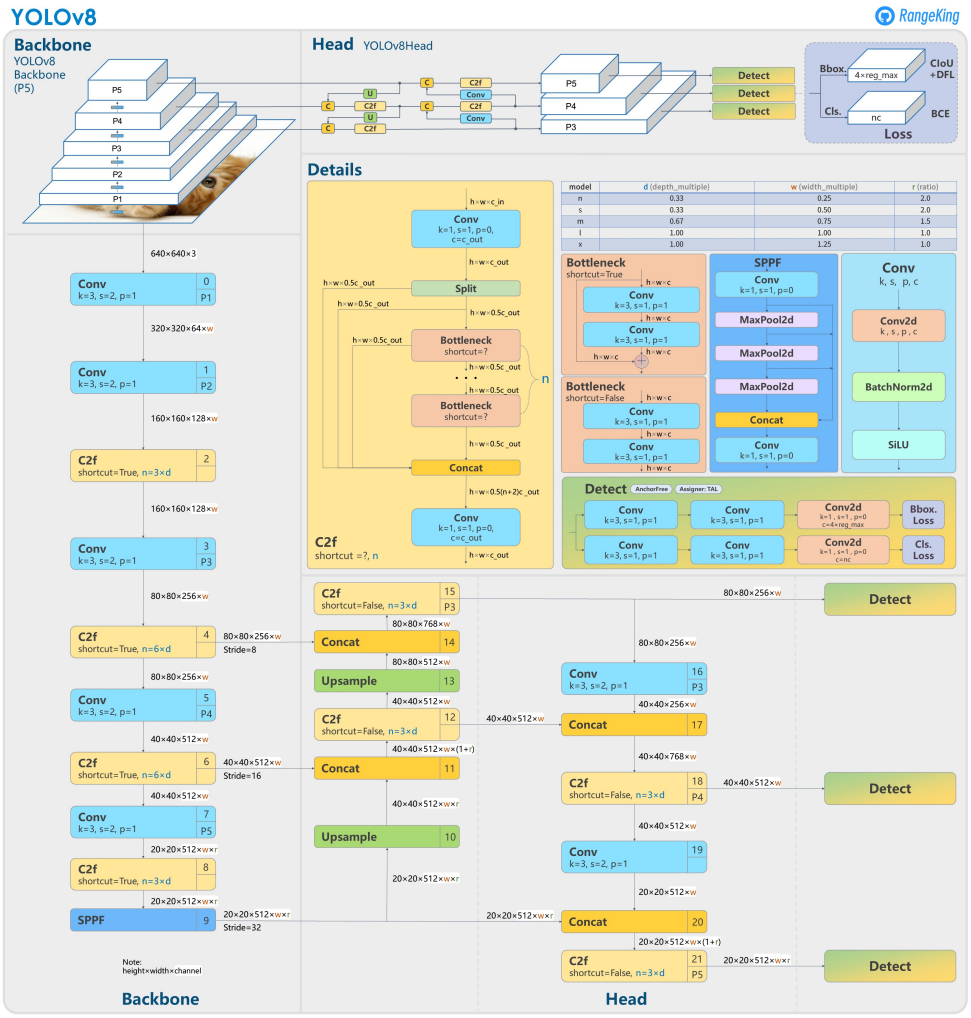

Figure A.1: Model structure of YOLOv8 detection models[33]

Bibliography

[1] R. Girshick, J. Donahue, T. Darrell, and J. Malik. "Rich feature hierarchies for accurate object detection and semantic segmentation". In: *arXiv [cs.CV]* (Nov. 2013).

[2] R. Girshick. "Fast R-CNN". In: *arXiv [cs.CV]* (Apr. 2015).

[3] S. Ren, K. He, R. Girshick, and J. Sun. "Faster R-CNN: Towards real-time object detection with region proposal networks". In: *arXiv [cs.CV]* (June 2015).

[4] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi. "You only look once: Unified, real-time object detection". In: *arXiv [cs.CV]* (June 2015).

[5] W. Liu et al. "SSD: Single Shot MultiBox Detector". In: *arXiv [cs.CV]* (Dec. 2015).

[6] T.-Y. Lin et al. "Focal Loss for dense object detection". In: *arXiv [cs.CV]* (Aug. 2017).

[7] N. Carion et al. "End-to-end object detection with transformers". In: *arXiv [cs.CV]* (May 2020).

[8] A. Vaswani et al. "Attention is all you need". In: *arXiv [cs.CL]* (June 2017).

[9] A. Bochkovskiy, C.-Y. Wang, and H.-Y. M. Liao. "YOLOv4: Optimal speed and accuracy of object detection". In: *arXiv [cs.CV]* (Apr. 2020).

[10] L. Zhu et al. "YOLO-Drone:Airborne real-time detection of dense small objects from high-altitude perspective". In: *arXiv [cs.CV]* (Apr. 2023).

[11] J. Suo, X. Zhang, W. Shi, and W. Zhou. "E^3-UAV: An edge-based energy-efficient object detection system for unmanned aerial vehicles". In: *arXiv [cs.RO]* (Aug. 2023).

[12] S. Liu et al. "EdgeYOLO: An edge-real-time object detector". In: *arXiv [cs.CV]* (Feb. 2023).

[13] E. Humes, M. Navardi, and T. Mohsenin. "Squeezed Edge YOLO: Onboard object detection on edge devices". en. In: *arXiv [cs.CV]* (Dec. 2023).

[14] *NVIDIA embedded systems for next-gen autonomous machines*. en. https://www.nvidia.com/en-us/autonomous-machines/embedded-systems/. Accessed: 2025-5-2.

[15] *NVIDIA TensorRT*. https://developer.nvidia.com/tensorrt. Accessed: 2025-5-2.

[16] *Rockchip-*. https://www.rock-chips.com/a/en/. Accessed: 2025-5-2.

[17] *The world's top performing edge AI processor for edge devices*. en. https://hailo.ai/. Accessed: 2025-5-2. July 2023.

[18] https://www.raspberrypi.com/. Accessed: 2025-5-2.

[19] *Coral*. en. https://coral.ai/. Accessed: 2025-5-2.

[20] *LiteRT for microcontrollers*. en. https://ai.google.dev/edge/litert/microcontrollers/overview. Accessed: 2025-5-2.

[21] G. Jocher, A. Chaurasia, and J. Qiu. *Ultralytics YOLOv8*. Version 8.0.0. 2023. URL: https://github.com/ultralytics/ultralytics.

[22] M. Yaseen. "What is YOLOv8: An in-depth exploration of the internal features of the next-generation object detector". en. In: *arXiv [cs.CV]* (Aug. 2024).

[23] *ImageHash*. en. https://pypi.org/project/ImageHash/. Accessed: 2025-5-4.

[24] R. W. Hamming. "Error detecting and error correcting codes". In: *Bell Syst. Tech. J.* 29.2 (Apr. 1950), pp. 147–160.

[25] VisDrone. *VisDrone-Dataset: The dataset for drone based detection and tracking is released, including both image/video, and annotations*. en.

[26] *Example notebooks*. en. https://albumentations.ai/docs/examples/. Accessed: 2025-5-4.

[27] *Ultralytics*. en. https://www.ultralytics.com/. Accessed: 2025-5-4.

[28] *Albumentations transforms*. en. https://explore.albumentations.ai/. Accessed: 2025-5-4.

[29] *Lightning AI*. en. https://lightning.ai/. Accessed: 2025-5-5.

[30] *NVIDIA L4 GPU datasheet*. en. https://resources.nvidia.com/en-us-gpu-resources/l4-tensor-datasheet. Accessed: 2025-5-5.

[31] *Get started with TensorBoard*. en. https://www.tensorflow.org/tensorboard/get_started. Accessed: 2025-5-5.

[32] *COCO - common objects in context*. https://cocodataset.org/. Accessed: 2025-5-5.

[33] *Ultralytics*. en.