UNIVERSITY OF PÉCS

FACULTY OF SCIENCES

INSTITUTE OF MATHEMATICS AND INFORMATICS

OBJECT DETECTION FOR SECURTIY CAMERA SYSTEMS IN INDOOR AND OUTDOOR ENVIRONMENTS

s

Supervisor: Ildikó Jenák

Assistant lecturer

Author: Abimbola Mohammed Ogunsakin (DKM9BE)

Computer Science BSc

PÉCS, 2025

Table of Contents

I. ABSTRACT

This thesis investigates the optimization of object detection models for security camera systems operating in diverse indoor and outdoor environments. The research focuses on addressing challenges posed by varying lightning conditions, weather effects, and occlusions, which often degrade detection accuracy.

Two State-of-the-art models, YOLOv8 AND SSDLite MobileNetV3 Large were evaluated and enhanced through a multi-phase methodology involving initial training, fine-tuning, and hyperparameter optimization using Optuna. The unified dataset, comprises MSCOCO128, DAWN, Objects365, and CCTV footage, was standardized to five classes: bus, car, motorcycle, person, and truck.

On a held-out test set, the optimized YOLOv8 model demonstrated remarkable performance, with mAP50 reaching 0.914 and mAP50-95 at 0.792, significantly outperforming the fine-tuned baseline. SSDLite MobileNetV3 Large, tailored for efficiency, improved from near-zero metrics to mAP50 of 0.471 and mAP of 0.318 after optimization.

A Streamlit web application, SecureDetect, was developed to enable real-time testing and comparison, demonstrating practical applicability.

This work advances computer vision by delivering robust, efficient object detection solutions for security surveillance, with broader implications for fields like autonomous vehicles and healthcare.

II. INTRODUCTION

As we continue to advance in the digital age, there has been an increase in demand for enhanced security surveillance in public and private spaces. Security cameras have been present in monitoring restricted areas like homes and offices and in the public like parking lots, streets, and malls. Although the cumbersome amount of data generated by these surveillance systems (security cameras) creates a challenge for humans, making it difficult to monitor all the video feeds in real time. To solve this, an aspect of computer vision used in identifying and classifying objects in an image or video feed, an automated object detection system has been developed, used in autonomous vehicles, facial recognition and robotics also adapted into security cameras to improve security surveillance by detecting potential threats or suspicious activity thereby reducing the reliance on humans.

However, in the context of security systems, object detection can automatically recognize people, vehicles, and objects of interest and trigger alerts in case of suspicious activity. Even with this level of advancement, there are challenges faced when using this technology, to environmental factors with various lighting conditions, occlusions, and weather effects. These factors tend to reduce the accuracy of detection thereby compromising the effectiveness of object detection in the security system.

The aim of this thesis is to analyze a range of factors that affect the algorithms and optimize existing algorithms used in object detection for security cameras that can work effectively in both indoor and outdoor environments. While exploring various models such as YOLO (You only look once) and SSD (Single Shot Detector), Faster R- CNN and so on , this research will explore how they perform in a real-world setting and evaluate possible optimization techniques like data augmentation, transfer learning and model tuning to improve them, thereby creating a more robust system that can overcome environmental limitations and improving the overall reliability of security cameras

This research aims to show practical relevance in the various applications of object detection in vehicles, healthcare, agriculture, industry, and anomaly detection.

III. THEORETICAL BACKGROUND

III.1. Object Detection

III.1.1. What Is Object Detection?

Object detection is the process of recognizing things of interest inside an image or video. According to the researchers at IBM, “It is an instance of artificial intelligence that consists of training computers to see as humans do, specifically by recognizing and classifying objects according to semantic categories” (Murel & Kavlakoglu, 2024).

An Object Detection System uses object detection models to identify position of objects in an image or video by returning the coordinates of the objects it has been trained to recognize. These systems are essential in various applications in security surveillance, and their performance depends on the algorithm used to identify and classify objects. Over the years there have been several ways of creating an object detection system; the most recent are machine learning and deep learning techniques.

III.1.2. Machine Learning In Object Detection

Traditional approaches to creating object detection relied on template matching and feature-based methods by comparing new images to predefined templates or shapes however this struggled with variations in object scale, lighting conditions, and weather (Parti, 2024).The constraint of the traditional approach brought about the use of machine learning, a subset of artificial intelligence for object detection that involves training models and using algorithms to learn patterns from data. With the use of manually engineered features to detect objects based on specific characteristics of the image like edges, gradients and so on.

Some of the early machines learning algorithms in object detection:

The Haar Cascades : Haar Cascades, an object detection algorithm from the early 2000s, detects objects by extracting Haar features (lines, edges, rectangles). To speed up feature extraction in large images, it uses integral images, creating sub-rectangles for efficient computation. Relevant features are selected using the Adaboost algorithm, which trains weak classifiers iteratively, assigning higher weights to misclassified samples to focus on challenging cases. This refines the classifiers, selecting the most relevant Haar features and improving detection accuracy (Data Science Wizards, 2023). (Viola & Jones, 2001)

Histogram of Oriented Gradients (HOG): It was proposed by Dalal and Triggs in 2005, is a traditional machine learning technique for object detection, initially popularized for pedestrian detection. It categorizes objects by their edge and gradient distributions, extracting shape features to train a binary classifier. HOG, a global descriptor, divides an image into small cells, calculates gradients, and groups them into orientation-based histograms representing edge structures. These histograms are normalized using block normalization to handle varying lighting and contrast, ensuring robust performance. HOG descriptors are then classified using a linear Support Vector Machine to detect objects. (Dalal & Triggs, 2005)

Scale Invariant Feature Transform (SIFT): It introduced a novel feature generation method to address limitations of earlier techniques, which lacked scale invariance and were sensitive to projective distortion and illumination changes (Lowe, 1999) .Designed to detect and describe objects regardless of orientation, size, or rotation, SIFT is valuable in surveillance systems where objects vary in distance. The algorithm identifies key locations (e.g., high-contrast corners, edges) using the Difference of Gaussian function across scales, assigning each a rotation-invariant orientation based on local image gradients. These form unique feature vectors for matching across images, enabling object identification despite scale or lighting variations, though it requires optimization for highly variable environments (Lowe, 1999).

These Traditional Machine Learning approaches have formed the foundational pillars of object detection techniques, the methods used in them have come with several limitations which are challenging e.g. struggles with accurately detecting objects in real world environments where there's lighting occlusion, scale variation and orientation constantly change and are hard to account for. The core principles that drove these traditional Machine learning approaches like feature extraction and pattern recognition remain the backbone and have directly made way for advanced technologies and models. The need for improved flexibility in varying conditions, improved detection accuracy, and automated feature extraction prompted the development of advanced machine learning techniques.

III.2. Deep Learning

III.2.1. What Is Deep Learning?

"Deep Learning is a subdivision of machine learning that uses multilayered neural networks, deep neural networks, to simulate the complex decision-making power of the human brain" (Holdsworth & Scapicchio, 2024).

According to the researchers at Google Cloud, Neural networks, which draw inspiration from the human brain, are the foundation of deep learning it is made up of layers of interconnected nodes in a layered like structure, with each node responsible for learning a specific feature of the data. Each layer of nodes has its own functions like a layer learning to identify the edges, another layer learning to find shapes and another layer to learn to recognize objects. As the layers of network learn, "the weights on the connections between the nodes are adjusted so the network can better classify the data, this process called training and can be done using a variety of techniques like supervised learning, unsupervised learning, and reinforcement learning". Neural networks are categorized based on their architecture, training methods and uses (Google, 2024).

As it has been established that neural networks is the core of deep learning, that learns to make decisions similar to the human brain by mimicking the way biological neurons work together to identify phenomena, weigh options and arrive at conclusions (IBM, 2024). It made of nodes, which include an input layer, one, two or more hidden layers, and an output layer. From the input layer through the hidden layers to the output layer, every node is connected to every other node. Each node has its own associated weight and threshold, if the output layer of any individual node is above the specified value, the node is activated and sends data to the next layer of network. Else no data is passed along to the next layers of network.

III.2.2. How Do Neural Networks Work?

The Researchers at IBM describes each individual node having its own linear regression model, which consists of the input data, weights, a bias, and an output.

when the input layer is determined, the weights are added to reduce error and to determine the importance of input data, with larger ones contributing to the impact of the output compared to other inputs. Each input is multiplied by the added weights and then summed up, then the output is determined by passing it through an activation function. There is a threshold given in which if the output is greater than, activates the node, passing data to the next layer in the network. “As a result, the output of one node becomes the input of the next node. This process is called feedforward network” (IBM, 2024). This is how a basic neural network works also known as Artificial Neural Network, however for more specialised use like image classification, speech recognition, object recognition and detection there are other methods to access its precision with a cost function. There are different ways this neural networks learn which can be seen in Table 1 below

Table 1: Various Training methods for deep learning (GeeksForGeeks, 2024)

Training Method	**Description**	**Key Features and Algorithms**
Supervised Learning	“The neural network learns to make predictions or classify data based on the labelled datasets, both the input features and target variables are fed to the neural network, and it learns to make predictions based on the cost or error that comes from the difference between the predicted and actual target”.	It utilizes algorithms like Convolutional Neural Networks and Recurrent Neural Networks. Learning is based on cost/error minimization.
Unsupervised Learning	In unlabeled datasets without target variables, the neural network learns to recognize patterns or clusters.	Algorithms like autoencoders and generative models are used
Reinforcement Learning	This technique uses agents to learn how to make decisions in an environment to maximize a reward signal. The agents interact with the environment by acting and observing the resulting rewards. It can be used to learn policies or a set of actions that maximizes the cumulative reward overtime.	It uses reinforcements learning algorithms like Deep Q networks and Deep Deterministic Policy Gradient (DDPG).

III.2.3. Convolutional Neural Network

There are distinct types of neural networks, varying on the purpose it serves.

The Convolutional Neural Network is used in image identfication, pattern recognition, and computer vision to discover patterns in images using 3-dimensional data. (IBM, 2024). As stated earlier, prior to the advancement of machine learning the traditional way explored manual, time-consuming feature extraction methods to identify objects in an image, but the use of Convolutional networks has offered a more scalable method for image classification and object detection tasks by automating this process and utilizing linear algebra concepts to find patterns in a picture, although they require huge computational power, using graphical processing units (GPU) to train models. (IBM, 2024)

How does Convolutional Neural Network work? Just like every other type of neural network, the convolutional neural networks share the fundamental principles, however they use advanced architectures specifically designed for image recognition and classification, by leveraging convolutional layers to capture patterns effectively. It features three main types of layers, convolutional later, pooling layer and the Fully connected layer explained in Table 2 below (IBM, 2024)

Table 2: Types of CNN Layers (IBM, 2024)

Layer	Description
Convolutional Layer	The convolutional layer, the foundational block of a Convolutional Neural Network, processes a 3D colored image (height, width, depth for RGB) using input data, filters, and a feature map. A filter, a 2D array of weights, moves across receptive fields, applying dot product calculations to input pixels, with results fed into an output array, repeated across the image with a stride to form a feature map. Pre-training hyperparameters—number of filters (e.g., three for depth 3), stride, and padding—are set to adjust output volume. A Rectified Linear Unit (ReLU) activation (Glorot et al., 2011)adds nonlinearity, converting the image into numerical values for pattern extraction, passed to the next layer (IBM, 2024)
Pooling Layer	In the Pooling layer, after the data is received from the previous layer, a dimensionality reduction is conducted to reduce the number of parameters in the input. Just like the convolutional layer, a filter is swept across the entire filter, but instead of weights an aggregation function to the value within the receptive field is applied, populating the output array. An amount of information is lost here but it is to an advantage, reduction in complexity, improved efficiency and a limited risk of overfitting. (IBM, 2024)
Full Connected Layer	In the Full Connected Layer, the previous layer the pixel values of the input image are partially connected to the output layer, however in this layer each node in the output layer connects directly to a node in the previous layer. The fully connected layer classifies based on the extracted features from the pooling layer and the filters applied, it leverages a SoftMax activation function to classify its inputs by producing a binary probability (IBM, 2024)

III.2.4. CNN Based Models in Object Detection

R-CNN (Region-Based Convolutional Neural Networks): Introduced by Ross Girshick in 2014, it applies convolutional neural networks to region proposals, improving detection accuracy over traditional methods like HOG and SIFT, which struggled with handcrafted features and inefficiency. This multistage model combines region proposals with CNNs, featuring three modules: Region Proposal, Feature Extraction, and Classification with Bounding Box Regression. In Region Proposal, selective search (Jasper , Van de Sande, Gevers, & Smeulders, 2013) groups similar regions based on color, texture, and brightness, generating high-recall ROIs despite false positives. Feature Extraction uses a C++-based Caffe framework to create 4096-dimensional vectors from 227x227 RGB images through five convolutional layers, reshaping ROIs for pretrained networks. Classification employs Support Vector Machines, with bounding box regression refining object localization. (Girshick, Donahue, Darrell, & Malik, 2014)

The R-CNN has proved to be significantly improved over traditional machine learning techniques, there has been some limitations due to its expensive computation, Each image has to pass through the network so many times during the region proposal that its slow during processing and became unsuitable for real time applications, which brought about the development of the new version Fast R-CNN by Ross Girshick in 2015.

Fast R-CNN: An Upgrade based on the previous R-CNN model, to address the slowed down processing that happens when the features are computed for each region proposal separately, this model applies the CNN to the entire image only once. The computation burden was reduced by making the region proposals project directly on the feature map (Girshick R. , 2015). Apart from this, the Region of interest in the pooling layer was used to extract fixed size feature vectors from regions of different sizes so that the structure of the image can be maintained during classification, the model also introduced multi task loss during training to improve the detection accuracy. (Girshick R. , 2015). Even though this upgrade improved the accuracy of detection and speed, the use of the selective search algorithm was noted to still limit the speed, and a new algorithm was used to replace it, leading to Faster R-CNN (Gavrilescu, Zet, Fosalau, Skoczylas, & Cotovanu, 2018)

Faster R-CNN: The most improved version of the RCNN replaces the Selective Search Algorithm with a small convolution network, Region Proposal Network, making it part of the neural network itself that generates regions of interest. The faster RCNN uses anchor boxes to scale region of interests, it combines two modules: a deep fully convolution network and a Fast R-CNN detector, to create a unified network for object detection that makes it faster than the RCNN by 99.6% without sacrificing detection accuracy. (Gavrilescu, Zet, Fosalau, Skoczylas, & Cotovanu, 2018). Faster R-CNN marked a huge leap forward and has been a relevant model in the context of security camera systems because of its ability to generate regional proposals and classify objects in a single pass.

YOLO: You only look once, a state-of-the-art model that was introduced by (Redmon et al., 2016) in the Facebook AI research back in 2016. The model has transformed object detection by framing it as a single regression problem to spatially separated bounding boxes and associated class probabilities. The paper mentions that YOLO views the whole image at once making it understand the context of objects within frame which leads to fewer false positives in cluttered scenes.

The model uses a grid-based detection in which it divides the input image into a grid and 0probability that the box contains an object and that the accuracy of the predicted bounding box, eliminating the need for generating region proposals to increase detection time. Every time an object is detected, the algorithm predicts class probabilities to know which category is present, then assigns a score based on how confident the prediction is. (Redmon, Divvala, Girshick, & Farhadi, 2016). Despite the algorithm being fast it struggles with detecting small objects accurately, this is a limitation of the algorithm which led to the transition of newer versions that has focused on improving both accuracy and performance.

Single Shot Multibox Detector: This model developed by (Liu et al., 2016), designed to balance between speed and accuracy in object detection. It uses feature maps from different layers of network to detect objects at various scales which allows it to capture small and large objects well.(Liu et al., 2016) The ability of the model to detect objects at various scales and aspect ratios has made it relevant in security camera systems where live detection is important and the system must be able to handle objects of varying sizes in different situations simultaneously. The model uses multi-scale feature map for detection which allows it to detect objects at various scales, convolutional predictors for detection in different resolutions at multiple layers and default boxes and aspect ratios. (Liu, et al., 2016). Despite the balance between speed and accuracy it struggles when it comes to precision and detecting tiny objects, and the use of fixed default boxes do not always align perfectly with the objects in the images (Annan, 2024).

III.3. Factors That Affect Models in Object Detection

Despite significant improvements over traditional methods in object classification, localization, and detection, deep learning models face challenges in security camera systems due to diverse environmental conditions. In low light, standard RGB cameras without infrared struggle, and models like YOLO and SSD lose accuracy due to reliance on well-lit training data (Redmon et al., 2016). Daytime overexposure from bright sunlight or glare obscures objects with shadows or reflections. Weather conditions like rain, fog, snow, or dust reduce image quality, hindering feature extraction and detection, especially for models needing clear boundaries (Mehra, Mandal, Narang, & Chamola, 2021). Monitoring large areas, such as warehouses, introduces scale variations, which Faster R-CNN handles with region proposals but at the cost of speed (Ren, He, Girshick, & Sun, 2015).

IV. MODELS ARCHITECTURE

IV.1 YOLOv8

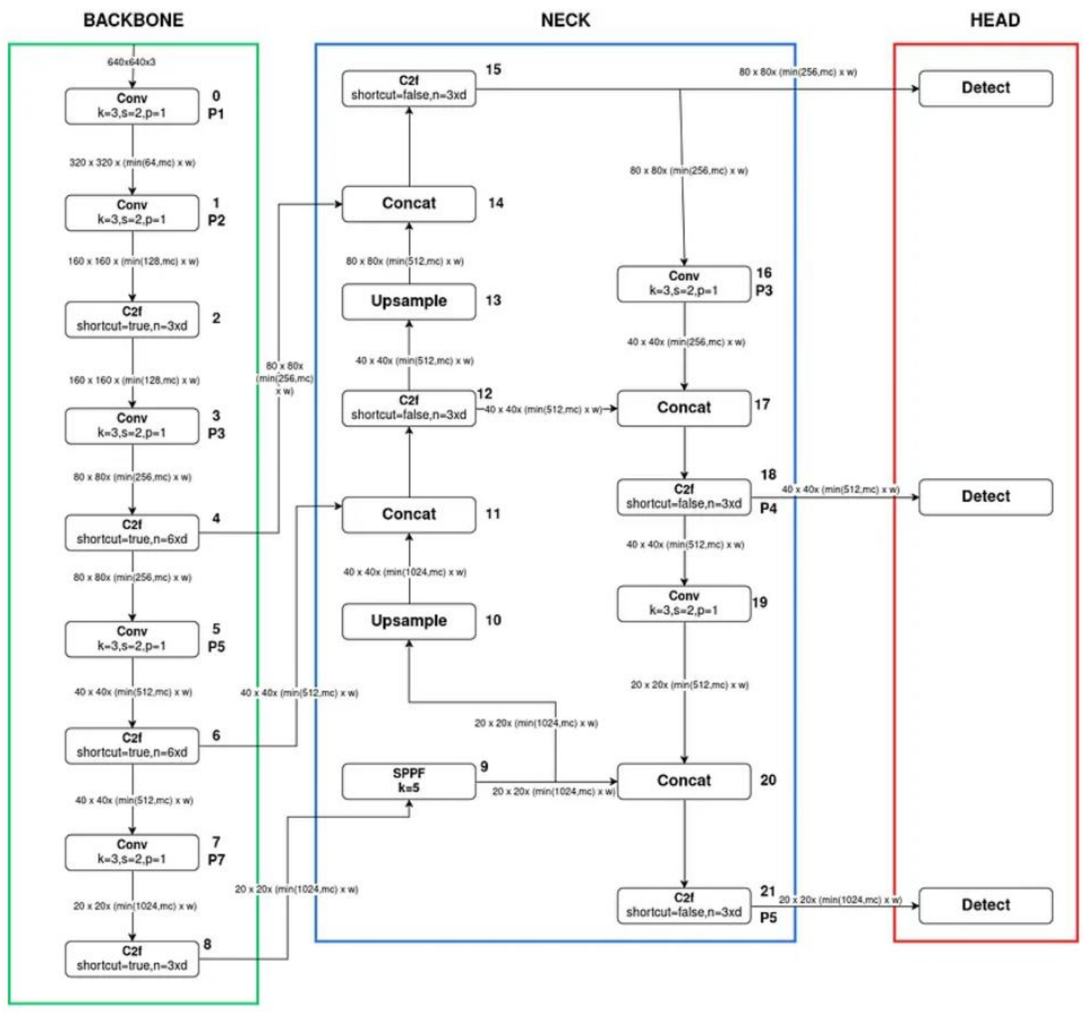

BACKBONE
NECK
HEAD
640x640x3
Conv k=3,s=2,p=1 0 P1
320 x 320 x (min(64,mc) x w)
Conv k=3,s=2,p=1 1 P2
160 x 160 x (min(128,mc) x w)
C2f shortcut=true,n=3xd 2
160 x 160 x (min(128,mc) x w)
Conv k=3,s=2,p=1 3 P3
80 x 80x (min(256,mc) x w)
C2f shortcut=true,n=6xd 4
80 x 80x (min(256,mc) x w)
Conv k=3,s=2,p=1 5 P5
40 x 40x (min(512,mc) x w)
C2f shortcut=true,n=6xd 6
40 x 40x (min(512,mc) x w)
Conv k=3,s=2,p=1 7 P7
20 x 20x (min(1024,mc) x w)
C2f shortcut=true,n=3xd 8
80 x 80x (min(256,mc) x w)
40 x 40x (min(512,mc) x w)
20 x 20x (min(1024,mc) x w)
SPPF k=5 9
Upsample 10
40 x 40x (min(1024,mc) x w)
Concat 11
C2f shortcut=false,n=3xd 12
40 x 40x (min(512,mc) x w)
Upsample 13
80 x 80x (min(512,mc) x w)
Concat 14
C2f shortcut=false,n=3xd 15
80 x 80x (min(256,mc) x w)
80 x 80x (min(256,mc) x w)
Conv k=3,s=2,p=1 16 P3
40 x 40x (min(256,mc) x w)
40 x 40x (min(512,mc) x w)
Concat 17
C2f shortcut=false,n=3xd 18 P4
40 x 40x (min(512,mc) x w)
40 x 40x (min(512,mc) x w)
Conv k=3,s=2,p=1 19
20 x 20x (min(512,mc) x w)
20 x 20x (min(1024,mc) x w)
20 x 20x (min(1024,mc) x w)
Concat 20
C2f shortcut=false,n=3xd 21 P5
20 x 20x (min(1024,mc) x w)
Detect
Detect
Detect

Figure 1: Full Architecture of YOLOv8

Introducing the YOLOv8 (Jocher Glenn et al., 2023), one of the most stable and well documented versions of the YOLO (you only look once) family, with a significant advancement in accurately and efficiently detecting objects, as well as in real-time situations, it features dynamic anchor boxes, optimized training strategies, and an improved generalization on diverse datasets. The features possessed by this model makes it a suitable choice for analysing and addressing the challenges faced by object detection models in security camera systems that operates in various environmental conditions both artificial and natural. The challenges introduced in the earlier paragraphs such as lighting conditions, weather conditions, camera limitations, hardware constraints and algorithmic bias, the adaption of these challenges to the YOLOv8 will be systematically evaluated in this thesis.

The YOLv8 (Jocher Glenn et al., 2023) architecture consists of three parts: backbone, neck, and the head as seen in Figure 1. Each layer leverages distinct blocks to process input images and produce accurate object detections.

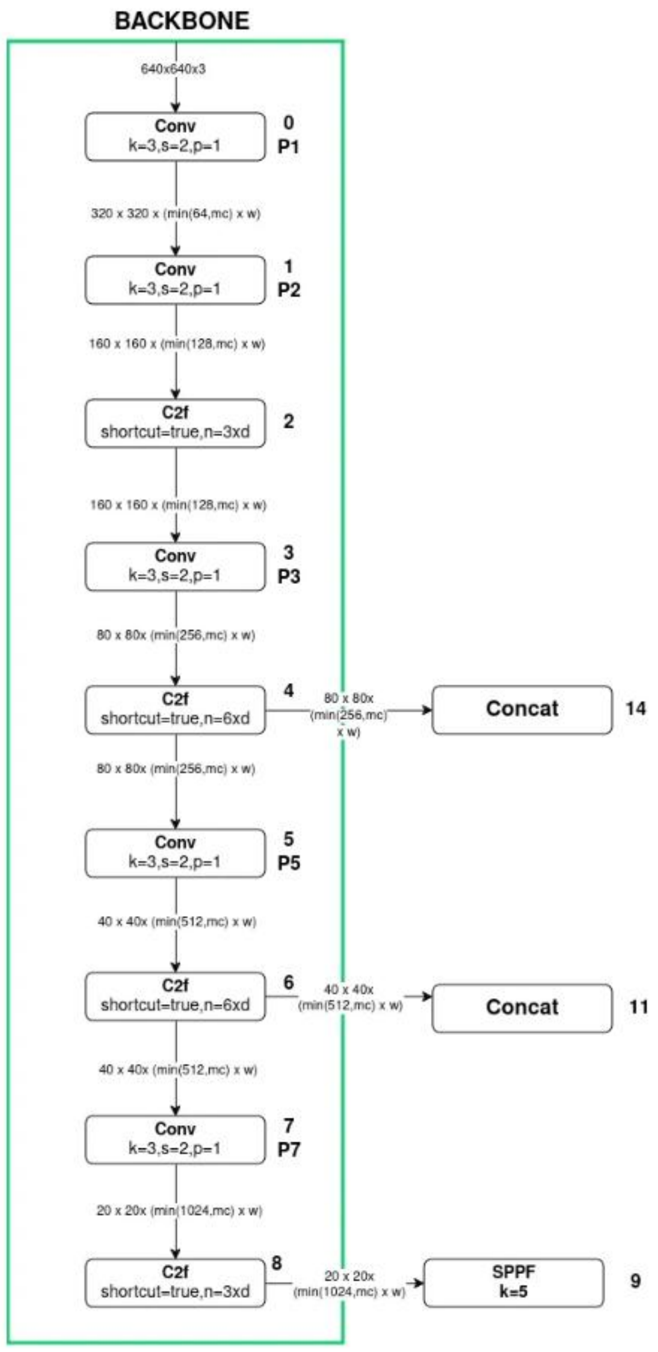

BACKBONE
640x640x3
Conv k=3,s=2,p=1
0 P1
320 x 320 x (min(64,mc) x w)
Conv k=3,s=2,p=1
1 P2
160 x 160 x (min(128,mc) x w)
C2f shortcut=true,n=3xd
2
160 x 160 x (min(128,mc) x w)
Conv k=3,s=2,p=1
3 P3
80 x 80x (min(256,mc) x w)
C2f shortcut=true,n=6xd
4
80 x 80x (min(256,mc) x w)
Concat
14
80 x 80x (min(256,mc) x w)
Conv k=3,s=2,p=1
5 P5
40 x 40x (min(512,mc) x w)
C2f shortcut=true,n=6xd
6
40 x 40x (min(512,mc) x w)
Concat
11
40 x 40x (min(512,mc) x w)
Conv k=3,s=2,p=1
7 P7
20 x 20x (min(1024,mc) x w)
C2f shortcut=true,n=3xd
8
20 x 20x (min(1024,mc) x w)
SPPF k=5
9

Figure 2: Backbone of YOLOv8

IV.1.1 Backbone

The backbone Figure 2 above extracts features from input images at multiple scales using 9 blocks, combining Convolutional Blocks (Conv Blocks), C2f Blocks, and a Spatial Pyramid

Pooling Fast (SPPF) (Jocher Glenn et al., 2023). The Conv Block, foundational to YOLOv8, includes a Conv2d layer, BatchNorm2d, and SiLU activation, processing a 640x640x3 input (e.g., to 320x320) with kernel size 3, stride 2, and padding 1, using min(64,mc)w for output channels to stabilize training and mitigate vanishing gradients (Jocher Glenn et al., 2023). The C2f Block splits the feature map, processing one part with Bottleneck Blocks (with optional shortcuts) and concatenating with the bypassed part into a final Conv Block, governed by depth_multiple (e.g., n=3d) to enhance feature extraction(Jocher Glenn et al., 2023). The SPPF Block, in Block 9, uses a Conv Block and three MaxPool2d layers to downsample and concatenate features, efficiently capturing multi-scale information without resizing the input (Jocher Glenn et al., 2023).

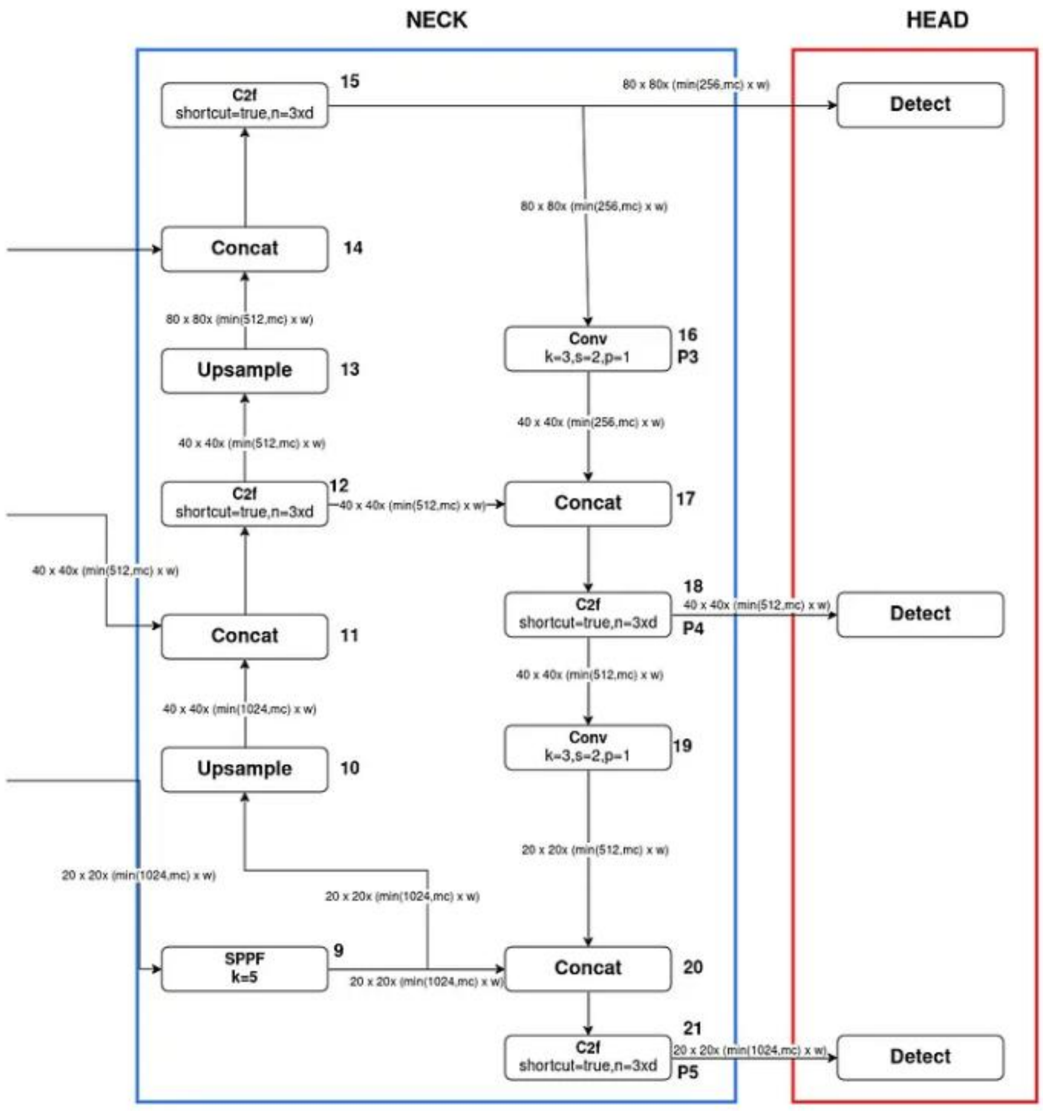

NECK
HEAD
C2f shortcut=true,n=3xd
15
80 x 80x (min(256,mc) x w)
Detect
80 x 80x (min(256,mc) x w)
Concat
14
80 x 80x (min(512,mc) x w)
Conv k=3,s=2,p=1
16
P3
Upsample
13
40 x 40x (min(512,mc) x w)
40 x 40x (min(256,mc) x w)
C2f shortcut=true,n=3xd
12
40 x 40x (min(512,mc) x w)
Concat
17
40 x 40x (min(512,mc) x w)
C2f shortcut=true,n=3xd
18
40 x 40x (min(512,mc) x w)
P4
Detect
Concat
11
40 x 40x (min(512,mc) x w)
40 x 40x (min(1024,mc) x w)
Conv k=3,s=2,p=1
19
Upsample
10
20 x 20x (min(512,mc) x w)
20 x 20x (min(1024,mc) x w)
20 x 20x (min(1024,mc) x w)
SPPF k=5
9
20 x 20x (min(1024,mc) x w)
Concat
20
C2f shortcut=true,n=3xd
21
20 x 20x (min(1024,mc) x w)
P5
Detect

Figure 3: Neck and Head Layer Architecture of YOLOv8

IV.1.2 Neck

The neck section as shown in Figure 3 above aggregates features from different backbone layers to create a multi-scale feature pyramid, essential for detecting objects of diverse sizes. It employs up sampling and concatenation operations (Jocher Glenn et al., 2023).

Upsample Layer: This layer doubles the spatial resolution of the feature map (e.g., from 80x80 to 160x160) without altering the output channels, facilitating the integration of deeper, semantically rich features with higher-resolution maps from earlier layers (Jocher Glenn et al., 2023).

Concat Block: The Concat Block combines feature maps from different backbone stages by summing their output channels, maintaining the resolution. This fusion enhances the model's ability to detect objects across scales by blending low-level details with high-level contextual information (Jocher Glenn et al., 2023).

The neck's structure resembles a Feature Pyramid Network (FPN), enabling effective multi-scale feature processing.

IV.1.3 Head

The head section shown in Figure 3 generates the final predictions for object bounding boxes and class labels. YOLOv8 adopts an anchor-free approach, directly predicting object centres rather than offsets from anchor boxes, which reduces post-processing complexity and speeds up inference (Jocher Glenn et al., 2023).

Detect Block: Multiple Detect Blocks operate at different scales, each tailored to detect small, medium, or large objects (sourced from C2f Blocks in Blocks 15, 18, and 21, respectively). Each Detect Block features two tracks: one for bounding box predictions and another for class predictions. Both tracks consist of two Conv Blocks followed by a Conv2d layer, outputting bounding box coordinates and class probabilities, respectively. This dual-track design ensures precise localization and classification (Jocher Glenn et al., 2023).

The backbone extracts multi-scale features, the neck integrates them into a cohesive pyramid, and the head delivers efficient, anchor-free predictions. This modular architecture underpins YOLOv8's effectiveness in real-time object detection, particularly for security camera systems under diverse conditions.

IV.2. Ssdlite Mobilenetv3 Large Architecture

The SSDLite320_MobileNetV3_Large model (Howard et al., 2019) is a lightweight, efficient variant of the Single Shot MultiBox Detector (Liu et al., 2016) framework, specifically optimized for resource-constrained devices like mobile and edge systems. It combines the core principles of SSD(Liu et al., 2016) with the MobileNetV3 Large backbone (Howard et al., 2019), enhancing its suitability for real-time object detection in applications such as security camera systems. This section explores the architecture, focusing on its backbone, detection mechanism, and efficiency, drawing from the original SSD paper and the MobileNetV3 paper.

IV.2.1. Backbone: Mobilenetv3 Large

Unlike the original SSD(Liu et al., 2016), which uses VGG16 as its backbone, SSDLite320_MobileNetV3_Large (Howard et al., 2019) employs MobileNetV3_Large, a highly efficient convolutional neural network (CNN) designed for mobile vision tasks. It leverages depth wise separable convolutions, inverted residuals, and squeeze-and-excitation (SE) blocks to reduce computational complexity while preserving accuracy.

Depthwise Separable Convolutions: These split standard convolutions into a depthwise convolution (applying a single filter per input channel) and a pointwise convolution (1x1 convolution to combine channels), significantly reducing parameters and computation.(Howard et al., 2019)

Inverted Residuals: Introduced in MobileNetV2, this structure expands the input channels with a 1x1 convolution, applies a depthwise convolution, and then projects back to a lower dimension, improving feature extraction efficiency.(Howard et al., 2019)

Squeeze-and-Excitation (SE): SE blocks enhance channel interdependencies by recalibrating feature maps, boosting representational power with minimal computational overhead.

Feature Map Extraction: MobileNetV3_Large(Howard et al., 2019) extracts feature maps at multiple layers (e.g., layer thirteen and the final layer), which are then used for multi-scale detection. For SSDLite320, the input resolution is 320x320, balancing speed and accuracy.

The MobileNetV3_Large backbone reduces the model's footprint compared to VGG16, making it ideal for real-time processing on edge devices while providing rich hierarchical features for detection.(Howard et al., 2019)

IV.2.2. Ssdlite Detection Mechanism

It adapts SSD's (Liu et al., 2016) detection pipeline by replacing standard convolutional predictors with depthwise separable convolutions, further optimizing for efficiency. It retains SSD's core innovations: multi-scale feature maps and default boxes.

Multi-Scale Feature Maps: it uses feature maps from different layers of MobileNetV3_Large (Howard et al., 2019) (e.g., 40x40, 20x20, 10x10, etc., depending on stride), enabling detection of objects at various scales. Earlier layers detect smaller objects, while deeper layers manage larger ones. At each feature map location, it defines a set of default boxes with pre-set aspect ratios (e.g., 1, 2, 1/2, 3) and scales. These function as priors for predicting object bounding boxes and are adjusted during inference.

Predictions: For each default box, it predicts four values (Δcx, Δcy, Δw, Δh) to refine the default box's position and size and confidence scores for each class (e.g., 91 for COCO, including background, 90+1).

The use of depthwise separable convolutions in the detection heads reduces the computational cost compared to SSD's original 3x3 convolutions, aligning with MobileNetV3's efficiency goals.

IV.2.3. Loss Function

It inherits SSD's (Howard et al., 2019) loss function, which combines localization and confidence losses; Smooth L1 loss measures the difference between predicted and ground-truth bounding boxes for positive matches ($IoU > 0.5$). Confidence Loss: Cross-entropy loss evaluates classification accuracy, with hard negative mining to balance the 3:1 ratio of negative to positive samples. Total Loss: A weighted sum of the two losses, optimized to refine predictions. This loss design ensures robust training despite class imbalance, critical for security camera applications with diverse object distributions.

Efficiency and Real-Time Suitability: SSDLite320_MobileNetV3_Large excels in efficiency, achieving real-time performance on edge devices. Key factors include Lightweight Backbone: it reduces FLOPs and parameters compared to VGG16, enabling deployment on resource-limited hardware. The Depthwise separable convolutions in detection heads lower the computational burden without sacrificing accuracy. The 320x320 input size strikes a balance between speed (higher FPS) and detection quality, suitable for security camera feeds.

Per (Howard et al., 2019), MobileNetV3_Large achieves top 1 accuracy of 75.2% on ImageNet with only 5.4M parameters, and when paired with the base model, it delivers competitive mAP (e.g., ~22-25% on COCO(Lin et al., 2014)) at over 30 FPS on mobile devices. This makes it an excellent choice for real-time object detection in security systems, where low latency and adaptability to varying conditions are paramount.

V. OPTIMIZATION TECHNIQUES

V.1. What are hyperparameters?

Hyperparameters are external configuration settings that control the learning process of the models. Unlike model parameters (weights in a neural network), which are learned during training, hyperparameters are set before the training begins and directly influences the model's performance and behaviour. E.g. learning rate, batch size, number of epochs and even architecture-specific settings like the number of layers. (GeeksforGeeks, 2025)

V.1.2. What is hyperparameter optimization?

This is the process of systematically selecting optimal values for a model's hyperparameter to reach the maximum performance on a given dataset in a reasonable amount of time. This is very crucial for deep learning models that have numerous hyperparameter that significantly impact accuracy, robustness, and training efficiency. (GeeksforGeeks, 2025)

There are various types of tools that can be used to extract more performance from deep learning models, especially in tasks like object detection where these models have a lot of hyperparameters, such as learning rate, batch size, epochs, and other neural network specific parameters that largely influences the accuracy and robustness. manually tuning these hyperparameters is unrealistic due to the huge search space and the computational cost, mostly for models deployed in security camera systems under adverse conditions like low or excessive light. Using an automated hyperparameter optimization technique addresses this issue by methodically investigating the parameter space to pick out configurations that will maximizes performance metrics, like mean Average Precision (mAP) in the case of computer vision models. In this thesis, two optimization tools will be evaluated; Hyperopt (Bergstra et al., 2015) and Optuna (Akiba et al., 2019), their methodology is discussed, followed by a justification for choosing Optuna (Akiba et al., 2019) as the most appropriate tool.

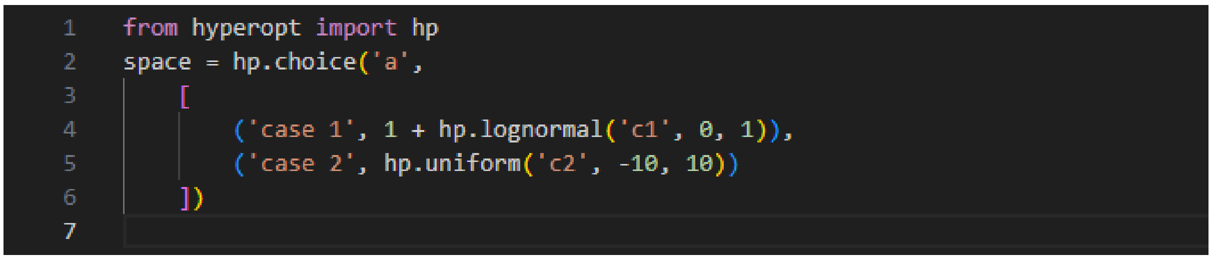
V.2. What Is Hyperopt?

"It is a python library for serial and parallel optimization over awkward search spaces, which may include real-valued, discrete, and conditional dimensions". (Bergstra, Komer, Eliasmith, Yamins, & Cox, 2013) It leverages a Bayesian optimization approach through the Tree-structured Parzen Estimator (TPE) algorithm. (Bergstra, Komer, Eliasmith, Yamins, & Cox, 2013) Hyperopt builds a probabilistic model of the objective function based on past trials and uses this to predict and sample potential hyperparameter combinations thereby reducing the number of evaluations needed which makes It more efficient than a random or manual tuning. (Bergstra, Komer, Eliasmith, Yamins, & Cox, 2013)

There are four primary features of Hyperopt required for optimization, discussed in the next subchapters.

V.2.1. Search Space

Hyperopt offers various functions to specify ranges for hyperparameters. "It consists of nested function expressions which includes stochastic expressions that are also hyperparameters" which can be seen in Figure 4, which specifies the random search algorithm. (Bergstra, Komer, Eliasmith, Yamins, & Cox, 2013).



```python
from hyperopt import hp
space = hp.choice('a',
    [
        ('case 1', 1 + hp.lognormal('c1', 0, 1)),
        ('case 2', hp.uniform('c2', -10, 10))
    ])

```

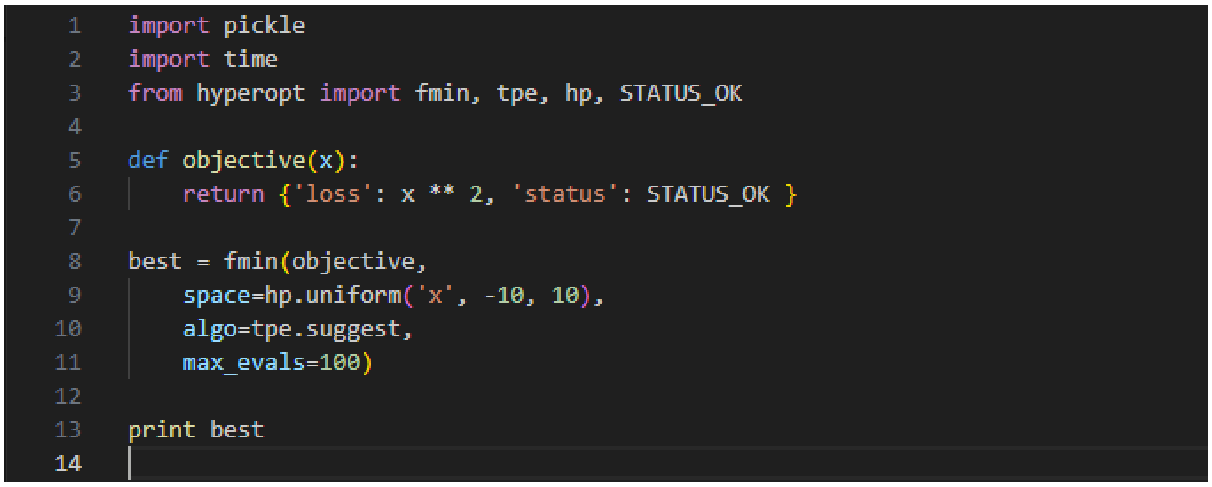
Figure 4: An Example of a search space

"This search space described by space has 3 parameters; 'a' - selects the case, 'c1' - a positive-valued parameter that is used in 'case 1', 'c2' - a bounded real-valued parameter that is used in 'case 2'". (Bergstra, Komer, Eliasmith, Yamins, & Cox, 2013)

There are other options to choose for a search space like hp.randint; in the case of an integer, and hp.normal; which returns a real value that's regularly distributed with a mean and standard deviation.(Bergstra et al., 2015)

V.2.2. Objective Function

A function that minimizes, accepts hyperparameter values as input from the search space and returns the loss. An implementation of the it is applied to minimize a quadratic objective function over one single variable, searching over a bounded range from -10 to +10 shown in Figure 5. (Bergstra, Komer, Eliasmith, Yamins, & Cox, 2013)

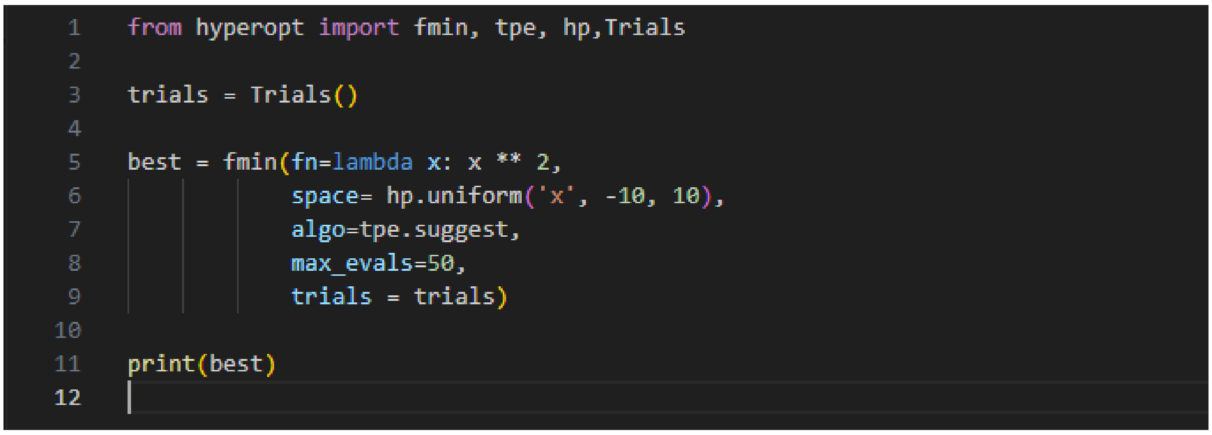


```python
import pickle
import time
from hyperopt import fmin, tpe, hp, STATUS_OK

def objective(x):
    return {'loss': x ** 2, 'status': STATUS_OK }

best = fmin(objective,
    space=hp.uniform('x', -10, 10),
    algo=tpe.suggest,
    max_evals=100)

print best
```

Figure 5: An Example of Objective Function

V.2.3. fmin

This is the optimization function that loops on different sets of algorithms and their hyperparameters, which minimizes the objective function. It takes 5 inputs as can be seen below in Figure 6; an objective function, a search space, a search algorithm, the maximum number of evaluations and a trials object with can be optional in some cases. (Bergstra, Komer, Eliasmith, Yamins, & Cox, 2013)

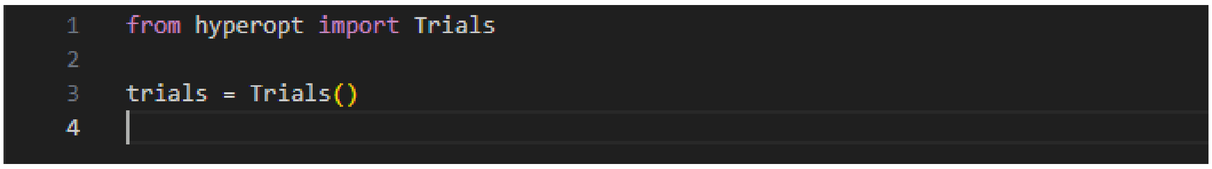


```python
from hyperopt import fmin, tpe, hp,Trials

trials = Trials()

best = fmin(fn=lambda x: x ** 2,
            space= hp.uniform('x', -10, 10),
            algo=tpe.suggest,
            max_evals=50,
            trials = trials)

print(best)
```

Figure 6: An Example of fmin.

V.2.4. Trial Object

As shown below in Figure 7, it keeps all hyperparameter, loss and other information so it can be accessed after running optimization. (Bergstra, Komer, Eliasmith, Yamins, & Cox, 2013)



```python
from hyperopt import Trials

trials = Trials()
```

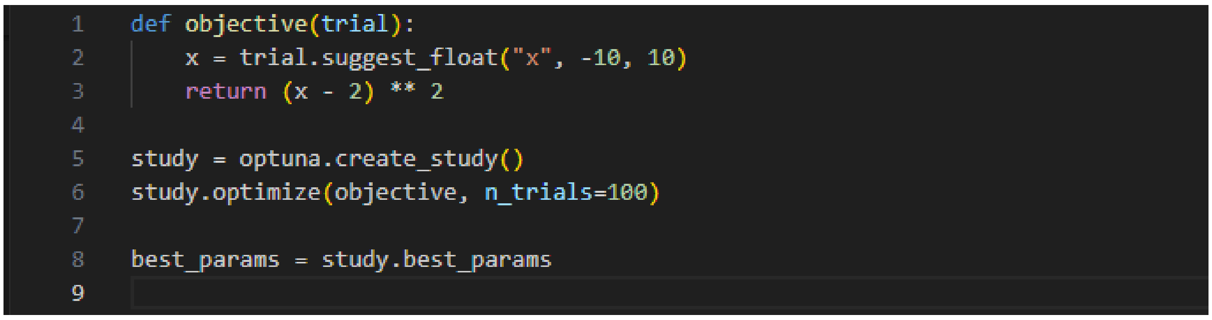
Figure 7: An Example of a trial object.

V.3. Optuna

Optuna (Akiba et al., 2019) is an open-source Python hyperparameter optimization framework developed by Akiba et al to automate hyperparameter search in machine learning or deep learning tasks. It scales efficiently across a single or multiple compute resources with little or no changes to code. It features a pythonic search space; uses a python syntax, an api; which allows construction of the parameter search space dynamically, an efficient implementation of searching and pruning design. Due to its lightweight and versatile architecture, it manages a wide variety of tasks, with simple installation and minimal dependencies. (Akiba, Sano, Yanase, Ohta, & Koyama, 2019)

Optuna Workflow Steps as shown below in Figure 8

1. An objective function to optimize; hyperparameter search space.
2. Create a *study* Object which is run by calling the *optimize* function of the study object. (Apache Software Foundation, 2024)



```python
def objective(trial):
    x = trial.suggest_float("x", -10, 10)
    return (x - 2) ** 2

study = optuna.create_study()
study.optimize(objective, n_trials=100)

best_params = study.best_params
```

Figure 8: An implementation of Optuna.

An *objective* function is defined, a suggest_float function is called to define the search space for the parameter x. A study is created and optimize the objective function with one hundred trials, in short calls the objective function one hundred times with different values of x. And the best parameter(s) is gotten from the study. (Apache Software Foundation, 2024)

V.3.1. Justification of choosing optuna over Hyperopt

Optuna supports pruning which improves Efficiency. One of the main reasons for choosing optuna is its pruning feature. In object detection tasks, training vision models takes a lot of time most especially with large datasets and complex architectures. Optuna's pruning allows the framework to stop unpromising trials early; in the case that a set of hyperparameters is not performing well after a few epochs. This saves a lot of computational resources, which is particularly important when I am working on a limited GPU time and need to iterate quickly to find the best set of hyperparameters. On the other hand, Hyperopt does not have this built-in pruning capability, so it would keep running all trials to completion, even ones that are clearly not going anywhere.

V.3.2. Dynamic Seach Space and Pythonic API

Optuna(Akiba et al., 2019) comes with built-in visualization tools that I really appreciate for documenting this thesis. I can generate plots of optimization history or parameter importance right out of the box which helps me to analyse what is working and not. This is most especially useful when I am trying to explain why certain hyperparameters matter more for mAP in object detection. Hyperopt (Bergstra et al., 2015) does not offer this kind of visualization natively, which would need to be built separately. Optuna's integration with tools like MLflow makes it easier to log and compare runs, keeping everything organized.

V.3.3. Justification for choosing Optuna over YOLOv8's in-built tuning.

Hyperparameter optimization is a critical step in the enhancement of the performance of the YOLOv8 object detection which is central to this thesis. While Ultralytics YOLOv8 library features an in-built hyperparameter tuning function, *model.tune ()*, this research opts for Optuna due to its superiority in flexibility, efficiency, and broader application. However, it is design imposes several constraints that limits its suitability for the scope of this thesis as shown in Table 3.

Table 3: YOLOv8m*(Jocher Glenn et al., 2023)* in-built tuning limitation and Optuna's*(Akiba et al., 2019)* Advantage

Aspect	**Yolov8's model.tune()**	**Optuna**
Model Compatibility	It is engineered exclusively for YOLOv8, rendering it incompatible with other architectures, which is a key focus of this thesis.	The model-agnostic Architecture in which Optuna operates independently of any specific model or framework which thus enables seamless hyperparameter optimization for various models.
Pruning Capability	It does not incorporate early stopping or pruning of trials that do not perform well	It is Pruning capability allows the early termination of unpromising trials based on intermediate performance metrics, which helps to reduce computational cost.

Search Strategy	It relies on a simpler search algorithm, which is less effective for complex spaces.	It supports sophisticated sampling techniques like the Tree-structured Parzen Estimator (TPE) and a Covariance Matrix Adaptation Evolution Strategy (CMA-ES).
Scalability	This function is not designed for large-scale or distributed optimization, which will limit its application in a more extensive experiments or future research expansions.	It supports parallel and distributed hyperparameter searches across multiple compute resources, which helps it to accommodate larger experiments in the future.
Dynamic Search Spaces	It has a less effective exploration of the hyperparameter space, resulting in sub optimal configurations.	Optuna's python-based API allows the creation of dynamic search spaces, where hyperparameters can depend conditionally on others (can be learning rates that are tied to optimizer types) very vital for tailoring optimization to model-specific requirements.

In addition of Optuna's immediate benefits for this project, It's model-agnostic design enhances its utility beyond the scope of YOLOv8, which ensures that the optimization strategy adopted in this thesis is not only effective for the current study but can also be transferred to future investigations that involves diverse architectures or tasks

VI. METHODOLOGY

VI.1. Model 1: YOLOv8m

As the goal of this thesis to develop an object detection model capable of performing reliably under diverse environmental conditions, such as those encountered in security camera systems. To create a model that can accommodate these conditions, two main approaches are considered for the YOLOv8 model: (1) loading a model with pretrained weights to leverage prior knowledge of patterns and features, or (2) initializing a new model based on a configuration file (a YAML) with pre-trained weights, which then initializes it with weights from the pre-trained model. This allows customization of the structure of the model to better suit a specific dataset, while also leveraging the pre-trained weights. Due to computational resource constraints, this study adopts the former approach using pre-trained weights applied to the YOLOv8. The process is structured into five phases, each designed to iteratively build and refine the model's performance across diverse datasets and conditions and tracked with MLFLOW (Zaharia et al., 2018)

Using the Medium version of the YOLOv8m model developed by (Jocher Glenn et al., 2023), processes input images at a resolution of 640 x 640 pixels. It consists of 25.9 million parameters, which allows it to have a balanced trade-off between accuracy and computational cost. It requires about 78.9 billion floating point operations per inference, showing its moderate complexity suited for real-time object detection tasks on mid-range hardware.

The dataset is a combination of images from MSCOCO128 (Lin et al., 2014), DAWN (Kenk & Hassaballah, 2020), objects365 (Shao et al., 2019) and publicly available CCTV footage dataset, standardized to five class labels; person, car, bus, motorcycle and truck. The combination of this dataset has been chosen due to the conditions being evaluated for.

A cloud-based GPU, NVIDIA A10G, provided by Lighting AI studio is used for all the stage processes and python is the main programming language used.

VI.1.1. Phase 1: Initial Training on MSCOCO128 Dataset

The Objective of this phase is to establish a baseline performance for the model by training it on the MSCOCO128 dataset using pre-trained weights which its performance can been seen in Figure 9 and Figure 10, assessing its generalization across diverse Object categories.

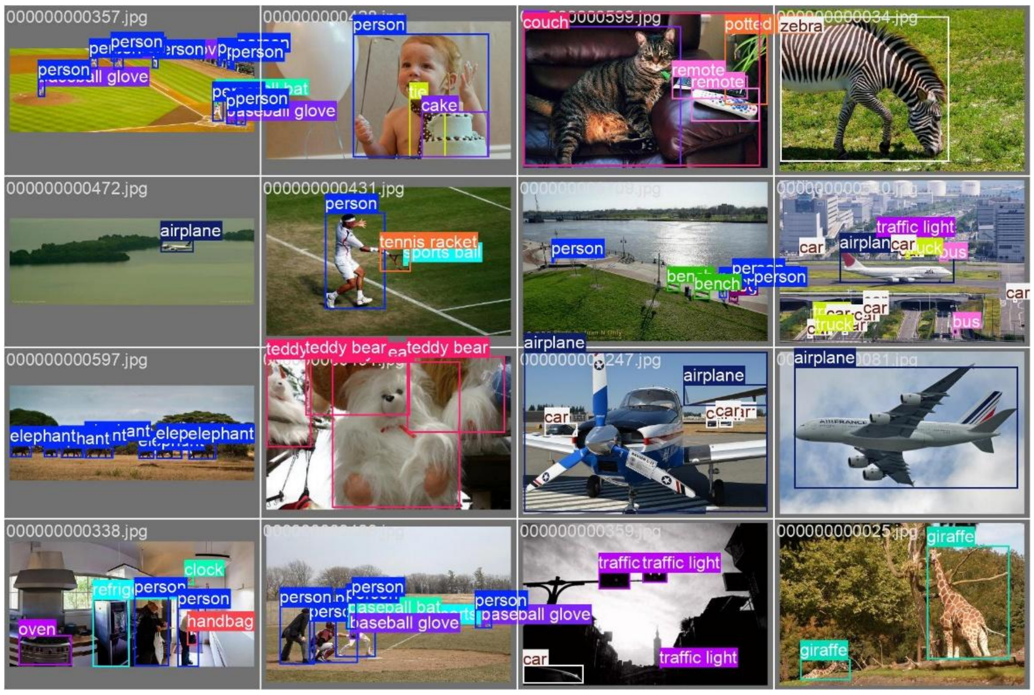


Figure 9: Sample images from coco128 dataset, with corresponding annotations

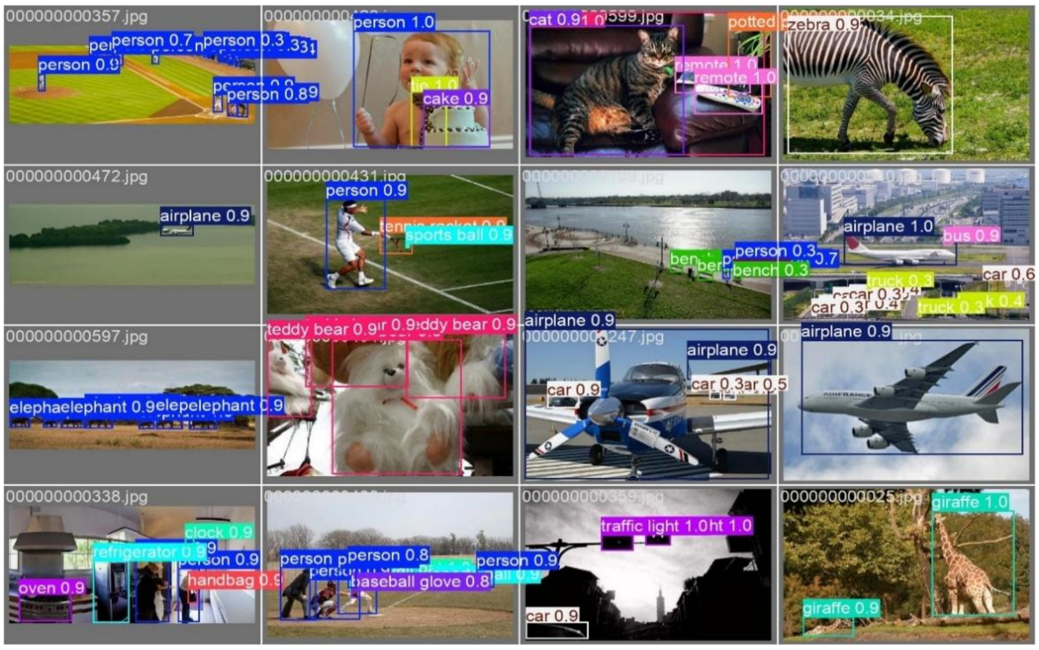


Figure 10: sample of model prediction during validation.

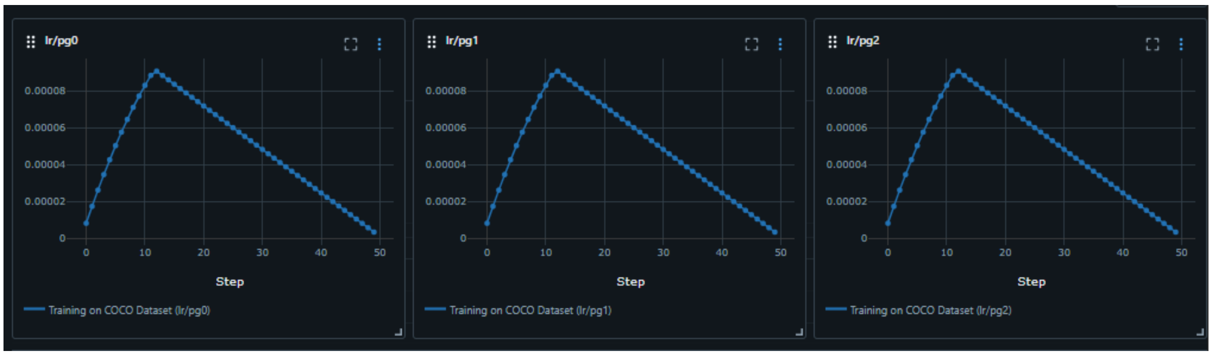


Figure 11: illustrates three key warmup loss components training dynamics on coco128 dataset.

Table 4: loss components values

lr/pg0	**lr/pg1**	**lr/pg2**
0.000003546	0.000003546	0.000003546

The horizontal axis represents the training step in Figure 11 and the corresponding values in Table 4, while the vertical axis denotes the loss magnitude for each component. Early into the training between (steps 0-10), there is a slight increase in each loss, which can be attributed to the learning rate warmup. Beyond step 10, all three losses begin to decline, which indicates that the model is increasingly accurate in localizing objects, identifying object regions, and classifying them into their correct categories. By the final steps, each loss converges to a stable lower value, showing that the model's network has effectively learned features relevant to COCO's diverse set of classes and is less likely to misclassify or mis localize.

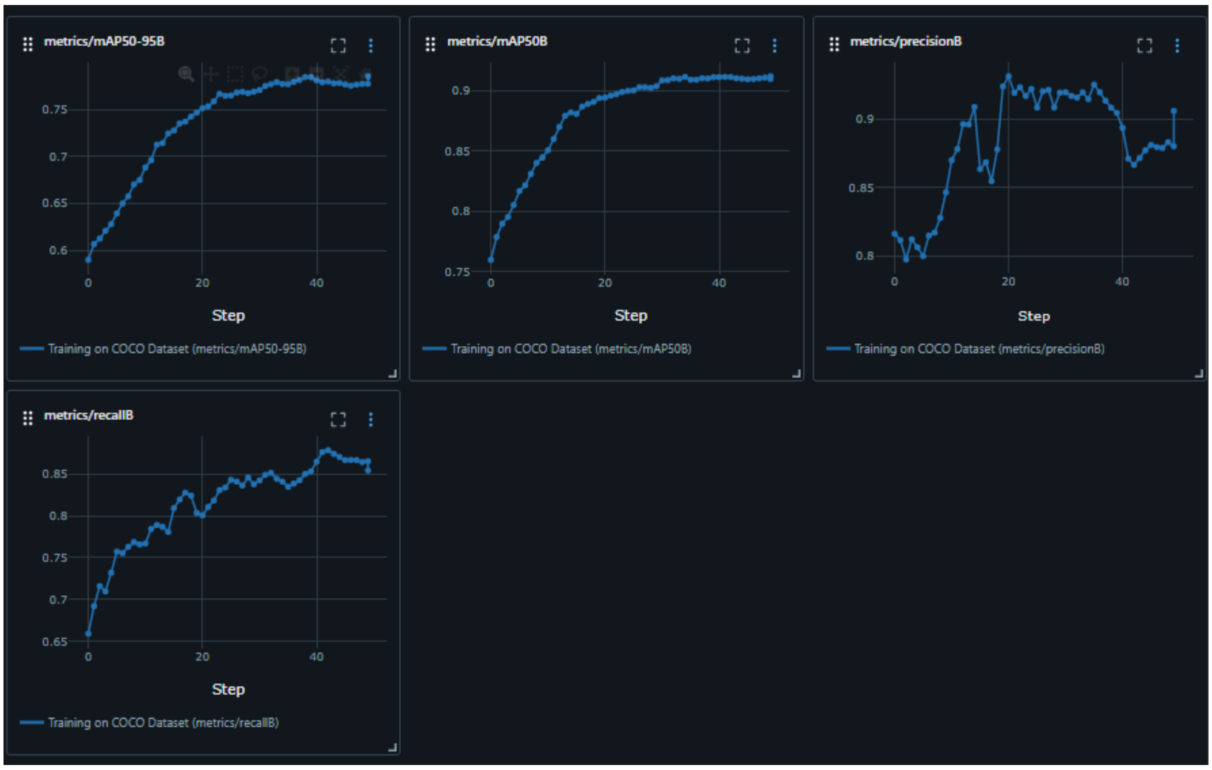


Figure 12: metrics from coco128 validation.

Table 5: metric values from coco128 validation

mAP50	**mAP50-95**	**Precision**	**Recall**
0.912	0.785	0.905	0.854

Figure 12 illustrates the progression of four core evaluation metrics: mAP50-95, mAP50, Precision and recall over fifty training steps and Table 5 shows the corresponding values. It can be observed that both mAP50-95 and mAP50 rise continuously, reflecting the model's growing ability to localize and classify objects across multiple intersection-over-union thresholds. Notably, the mAP50 tends to be higher than mAP50-95 because it demands a less stringent overlap requirement (IoU ≥ 0.5) to register a successful detection. The precision and recall show complementary behaviours; as the precision increases, the model reduces the proportion of false-positive detections, ensuring that most of its predicted bounding boxes indeed correspond to real objects. Meanwhile, the upward trend in recall indicates that the model is detecting a larger fraction of the ground-truth objects. At the end of the training, the model converges under the current training regimen.

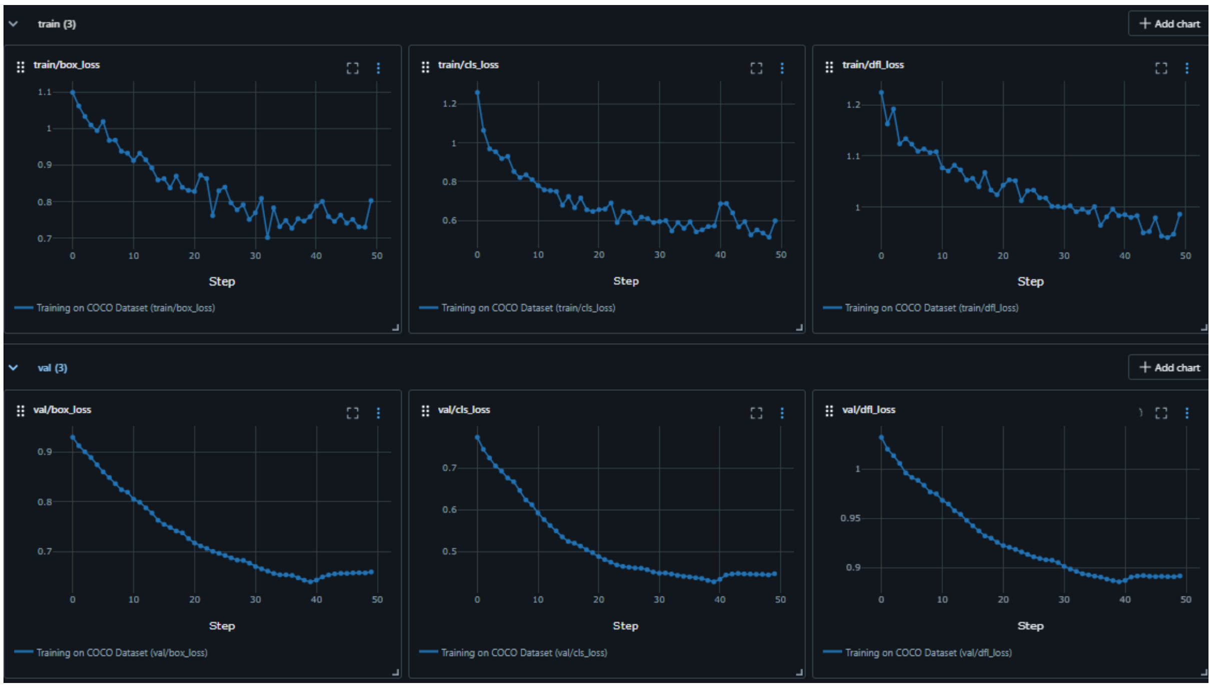


Figure 13: model's training and validation losses.

Table 6: Model training and validation loss values

train/box_loss	**train/cls_loss**	**train/dfl_loss**
0.803	0.599	0.986
val/box_loss	**val/cls_loss**	**val/dfl_loss**
0.659	0.447	0.892

The trends in the Figure 13 highlight the model's successful training on the dataset with Table 6 showing corresponding values. The steady decline in both training and validation losses over fifty steps reflect effective optimization of bounding box localization, object classification, and distribution focal loss components. The smoother validation curves indicate strong generalization, a critical attribute for real-world object detection applications. However, the fluctuations in training loss, most especially in cls_loss and dfl_loss, calls for further investigation. These irregularities could stem from class imbalances, complex object distributions in the dataset, or suboptimal hyperparameter settings. The model reached a convergence point for classification after step 20 in val/cls_loss.

Overall, this has set a baseline for the model, performance-wise and we can proceed to fine-tuning the model to our dataset.

VI.1.2. Phase 2 (checking performance on the dataset that contains the conditions)

In the phase, is the first decisive moment where the model will be put to the real test on the unified dataset.

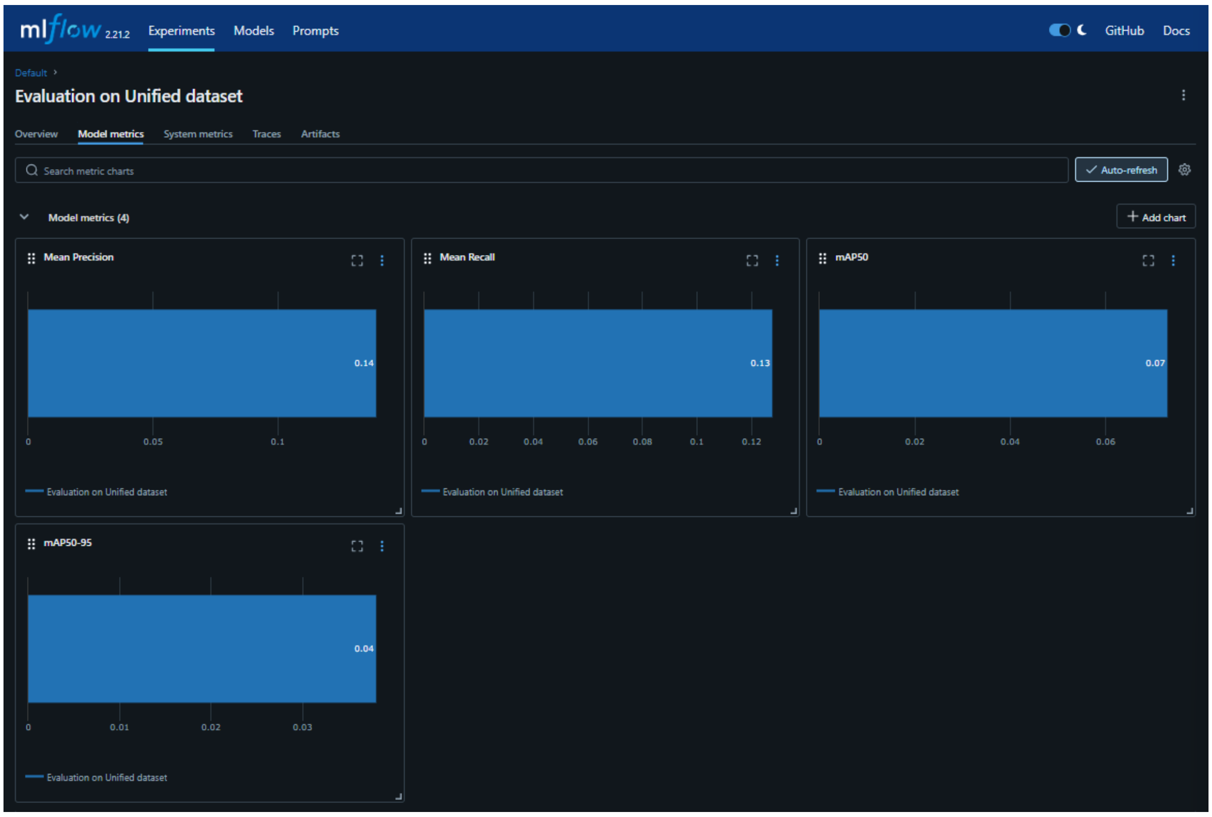


Figure 14: Evaluation of Unified Dataset

Evaluation on the unified dataset as seen in Figure 14 revealed that the model's detection performance drops significantly compared to COCO128 validation metrics. This outcome aligns with expectations, given the increased visual complexity and domain diversity present in the unified dataset, especially from the CCTV images, DAWN dataset variations (weather lighting) and Object365's long-tail distribution classes. The model's mAP50-95 of 0.038 and mAP50 of 0.073 show that while the model can detect some objects, its localization accuracy is still limited. Moreover, the low precision and recall further suggests that both false positives and false negatives are frequent. This calls the need for domain adaptation techniques, fine-tuning the model to the unified dataset.

VI.1.3. Phase 3 (fine-tuning on coco-trained-model) layers frozen)

The objective of this phase is to adapt the coco-trained model from phase 1 to the unified dataset by fine-tuning its detection head (classification and box-regression layers), while keeping the backbone weights frozen. This strategy preserves the general feature learned from the large-scale MSCOCO dataset (Lin et al., 2014) and focuses on the specific appearance distributions and classes present in the unified dataset, such as varying, weather, lighting, and scene complexity, encountered in security camera systems.

The coco-trained model was fined-tuned on the unified dataset with its backbone layers frozen, allowing only the detection head to be updated. The fine-tuning process utilized the following hyperparameters, logged via MLflow (Zaharia et al., 2018)

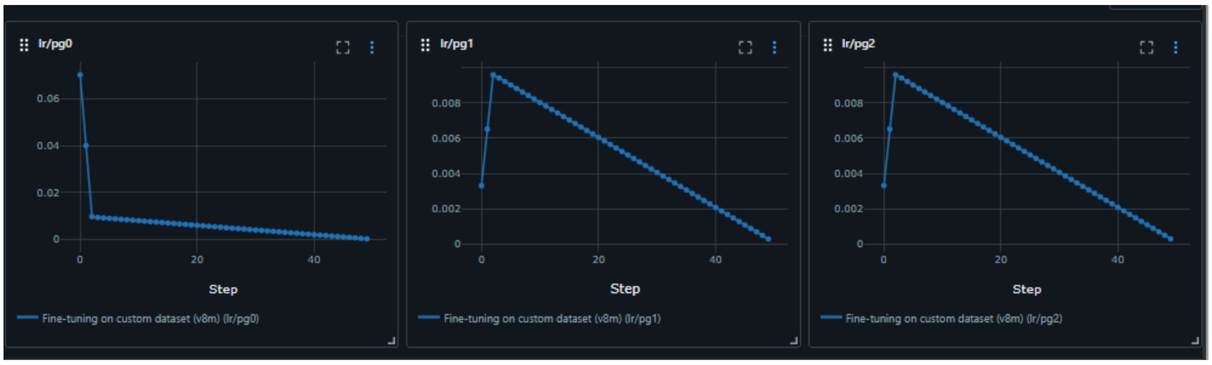


Figure 15: illustrates three key loss warmup components training on coco-trained-model after fine-tuning.

Table 7: loss component values

lr/pg0	lr/pg1	lr/pg2
0.000298	0.000298	0.000298

It can be observed in Figure 15 with corresponding values in Table 7 7, that in the learning-rate schedule, the three optimizer parameter groups (pg0, pg1, pg2), each warmed up in the first step and then linearly decayed over fifty steps. Although pg0 peaks at ~ 0.068 at step 0, and immediately drops to 0.0008 by step 1, then decays toward zero. While pg1 and pg2 warm up from ~ 0.004 to ~ 0.009 within the first step, then linearly decline to ~ 0 by step 50. Overall, the aggressive warmup prevents gradient explosions in the new head layers, while the gradual decay ensures stage convergence over fine-tuning steps.

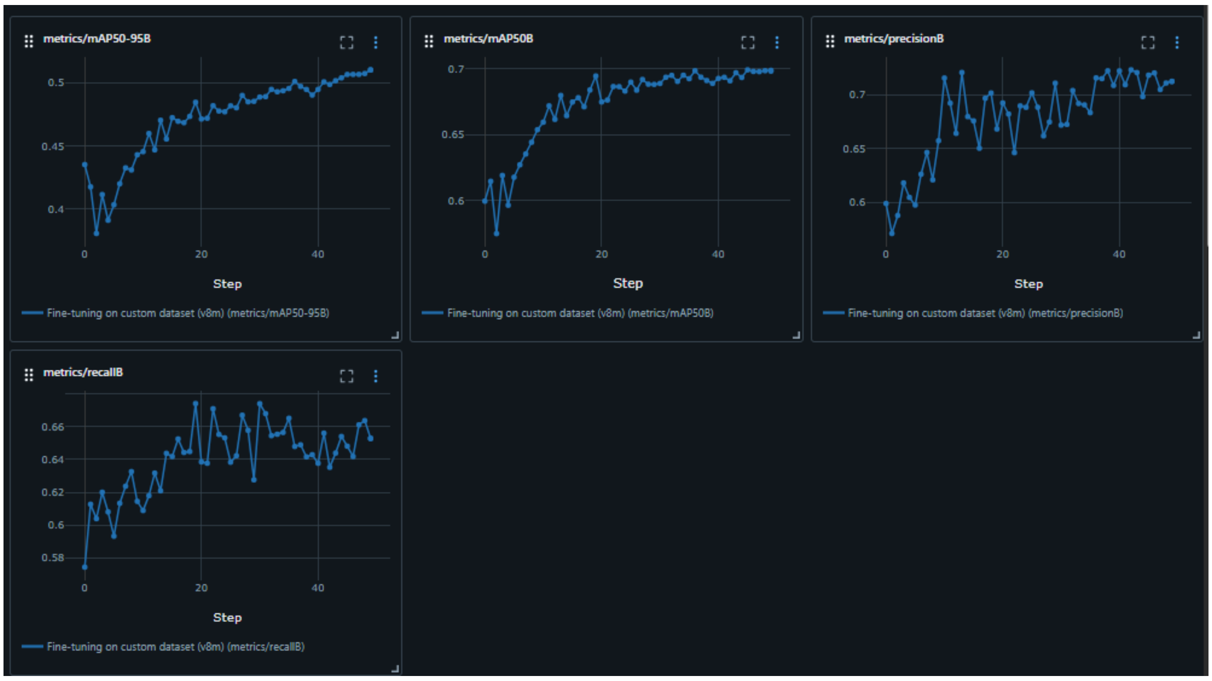


Figure 16: illustrates evaluation metrics on fine-tuned model.

Table 8: evaluation metrics values

mAP50	**mAP50-95**	**Precision**	**Recall**
~ 0.698	~ 0.510	~ 0.712	~ 0.653

It can be observed in Figure 16, that both mAP50-95 and mAP50 corresponding values in Table 8 show a similar trend jumping from ~ 0.38 to ~ 0.45 and ~ 0.57 to ~ 0.66 respectively within the first 10 steps, then climbs more gradually demonstrating the head quickly learns coarse localization before refining precision. Although the mAP50 tapered towards higher than mAP50-95 ending at ~ 0.51. Precision increases from ~0.58 to ~0.71 with minor fluctuations, reflects that the model is sensitive to hard negatives in CCTV and adverse weather images. Recall grows steadily from ~0.57 to ~0.65, with occasional spikes corresponding to more challenging validation batches. All metrics begin to plateau neat the step 40, suggesting the fine-tuning schedule has exhausted the head's capacity under a frozen backbone.

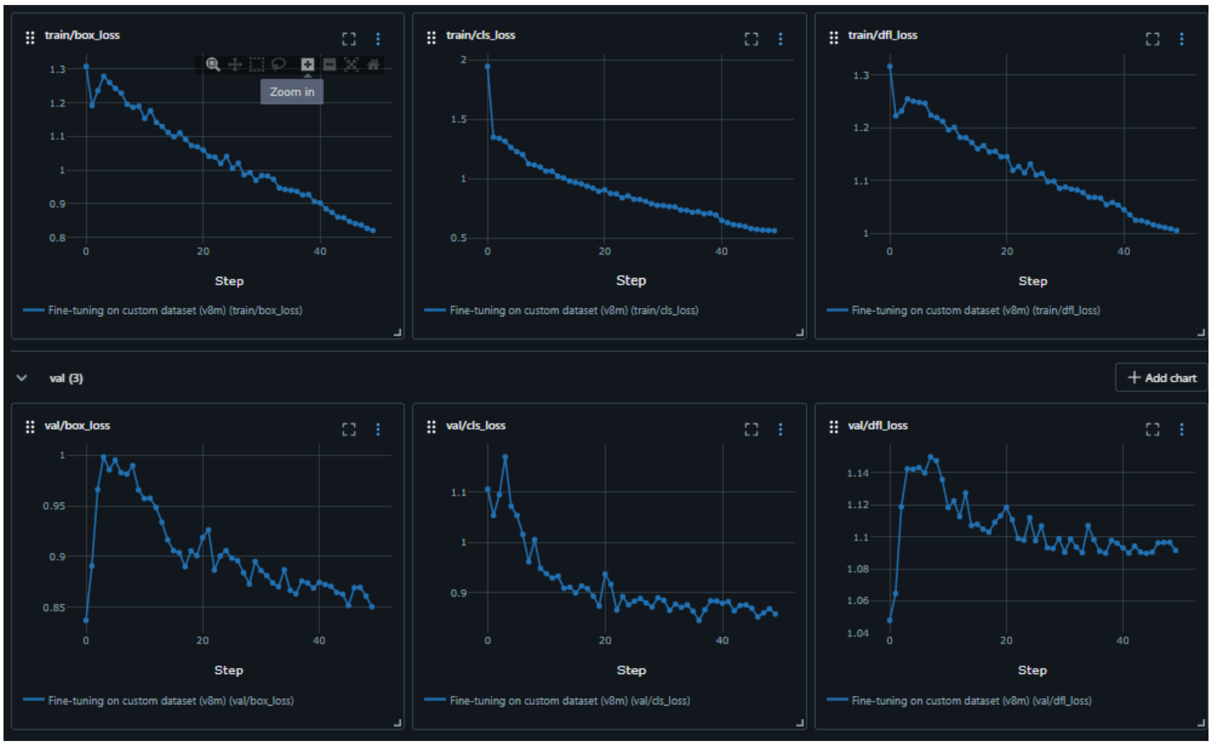


Figure 17: Model's training and validation losses.

Table 9: Model's training and validation losses values

train/box_loss	train/cls_loss	train/dfl_loss
~ 0.82	~ 0.56	~ 1.00
val/box_loss	**val/cls_loss**	**val/dfl_loss**
~ 0.85	~ 0.85	~ 1.09

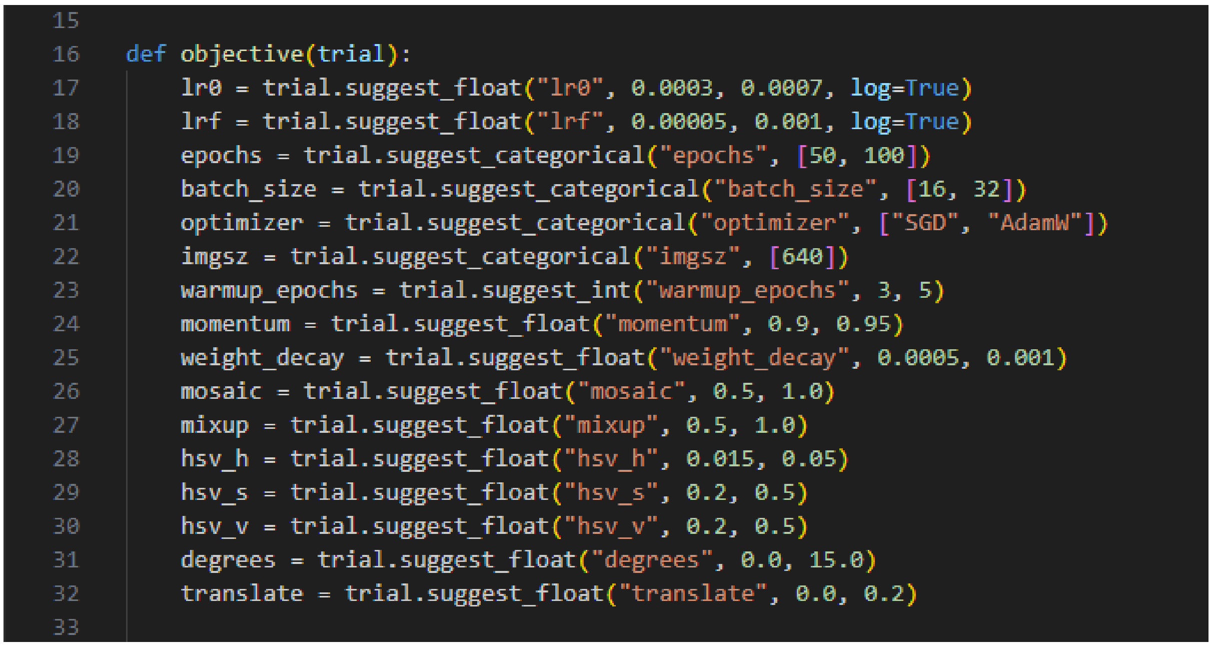
All three training losses observed in Figure 17 with corresponding value in Table 9, decline smoothly, demonstrating effective learning in localization (box_loss), classification (cls_loss) and boundingbox refinement (dfl_loss). Validation losses remain close to their training counterparts, confirming that the finetuned head generalizes well despite the frozen backbone.

Overall, Phase 3 successfully specialized the coco-trained YOLOv8m model's detection head to the unified dataset, achieving substantial performance gains. The mAP50 improved to ~0.698 and mAP50-95 to ~0.510, with precision and recall reaching ~0.712 and ~0.653, respectively. These improvements highlight the effectiveness of fine-tuning the detection head while retaining the general features from MSCOCO. The smooth decline in losses and close alignment between training and validation metrics further confirm the model's ability to generalize across the diverse conditions of the unified dataset.

Despite these advancements, the plateauing of metrics around step 40 indicates that the model has exhausted the potential of the detection head under a frozen backbone. The general features from the backbone, while valuable, may not fully capture dataset-specific nuances, such as unique lighting or weather conditions in CCTV footage. Additionally, minor fluctuations in precision suggest sensitivity to challenging samples, like occluded or small objects. To push performance further particularly in localization accuracy (mAP50-95) and robustness optimization is required. The next phase will involve unfreezing select backbone layers and exploring hyperparameter tuning to enable deep adaptation to the unified dataset's complexities.

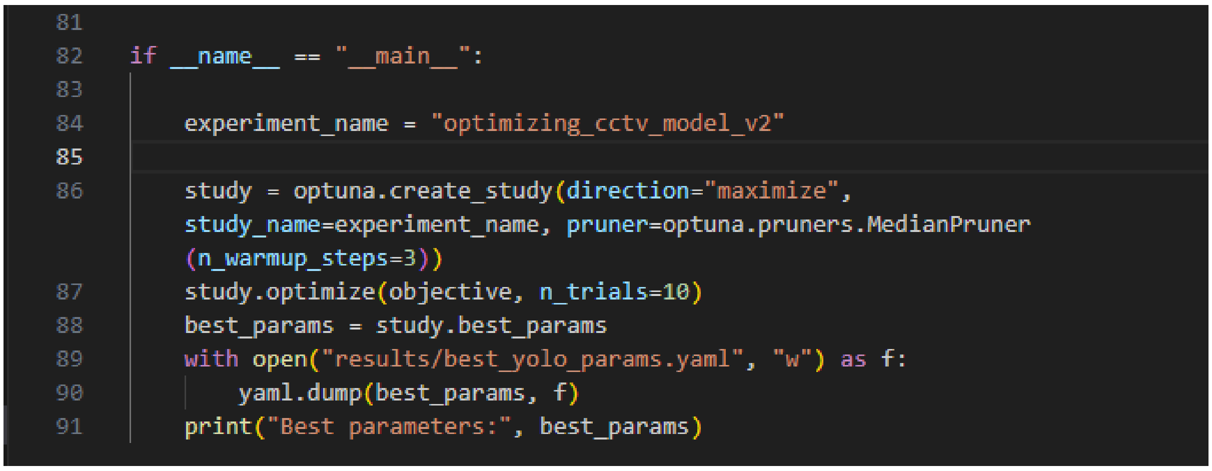
VI.1.4. Phase 4: Fine-Tuning with Unfrozen Backbone Layers and Hyperparameter Optimization.

The objective of this phase is to further enhance the YOLOv8m model's performance on the unified dataset by unfreezing select backbone layers and optimizing hyperparameters. This approach builds on Phase 3 by allowing the backbone to adapt to dataset-specific features such as unique lighting, weather conditions, and object occlusions in CCTV footage that were not fully captured with a frozen backbone. The goal is to improve localization accuracy (mAP50-95), robustness, and overall detection quality under diverse environmental conditions encountered in security camera systems with hyperparameters defined in Figure 18 and study created in Figure 19.



```python

def objective(trial):
    lr0 = trial.suggest_float("lr0", 0.0003, 0.0007, log=True)
    lrf = trial.suggest_float("lrf", 0.00005, 0.001, log=True)
    epochs = trial.suggest_categorical("epochs", [50, 100])
    batch_size = trial.suggest_categorical("batch_size", [16, 32])
    optimizer = trial.suggest_categorical("optimizer", ["SGD", "AdamW"])
    imgsz = trial.suggest_categorical("imgsz", [640])
    warmup_epochs = trial.suggest_int("warmup_epochs", 3, 5)
    momentum = trial.suggest_float("momentum", 0.9, 0.95)
    weight_decay = trial.suggest_float("weight_decay", 0.0005, 0.001)
    mosaic = trial.suggest_float("mosaic", 0.5, 1.0)
    mixup = trial.suggest_float("mixup", 0.5, 1.0)
    hsv_h = trial.suggest_float("hsv_h", 0.015, 0.05)
    hsv_s = trial.suggest_float("hsv_s", 0.2, 0.5)
    hsv_v = trial.suggest_float("hsv_v", 0.2, 0.5)
    degrees = trial.suggest_float("degrees", 0.0, 15.0)
    translate = trial.suggest_float("translate", 0.0, 0.2)

```

Figure 18: Hyperparameters optimized.



```python

if __name__ == "__main__":

    experiment_name = "optimizing_cctv_model_v2"

    study = optuna.create_study(direction="maximize",
    study_name=experiment_name, pruner=optuna.pruners.MedianPruner
    (n_warmup_steps=3))
    study.optimize(objective, n_trials=10)
    best_params = study.best_params
    with open("results/best_yolo_params.yaml", "w") as f:
        yaml.dump(best_params, f)
    print("Best parameters:", best_params)
```

Figure 19: Optuna trial implementation.

Building on the fine-tuned model from Phase 3, unfreezing the last ten convolutional layers of the backbone to enable their weights to be updated during training. This selective unfreezing balances the retention of general features from MSCOCO (Lin et al., 2014) with the need for domain-specific learning. The fine-tuning process utilized the following hyperparameters, logged in MLflow (Zaharia et al., 2018).

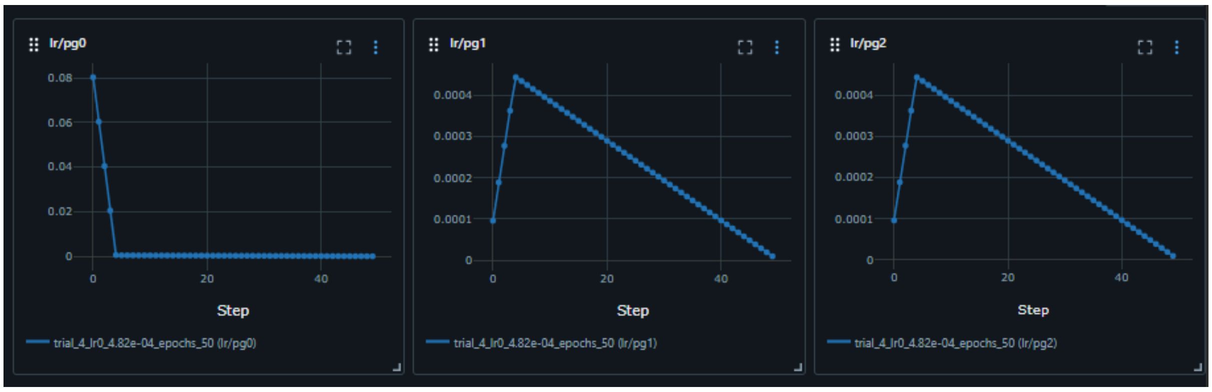


Figure 20: warmup loss components training dynamics on unified dataset.

Table 10: warmup loss components values

lr/pg0	**lr/pg1**	**lr/pg2**
~ 0.000009847	~ 0.000009847	~ 0.000009847

It can be observed in Figure 20, and corresponding values in Table 10 that in the learning-rate schedule, the three optimizer parameter groups (pg0, pg1, pg2) each warmed up in the initial steps and then linearly decayed over fifty steps. The learning rates peaked at ~0.001 at step 0, dropped to ~0.0001 by step 5, and decayed to ~0.0000098 by step 50. This conservative warmup and decay schedule prevented overfitting while allowing the unfrozen backbone layers to adapt gradually.

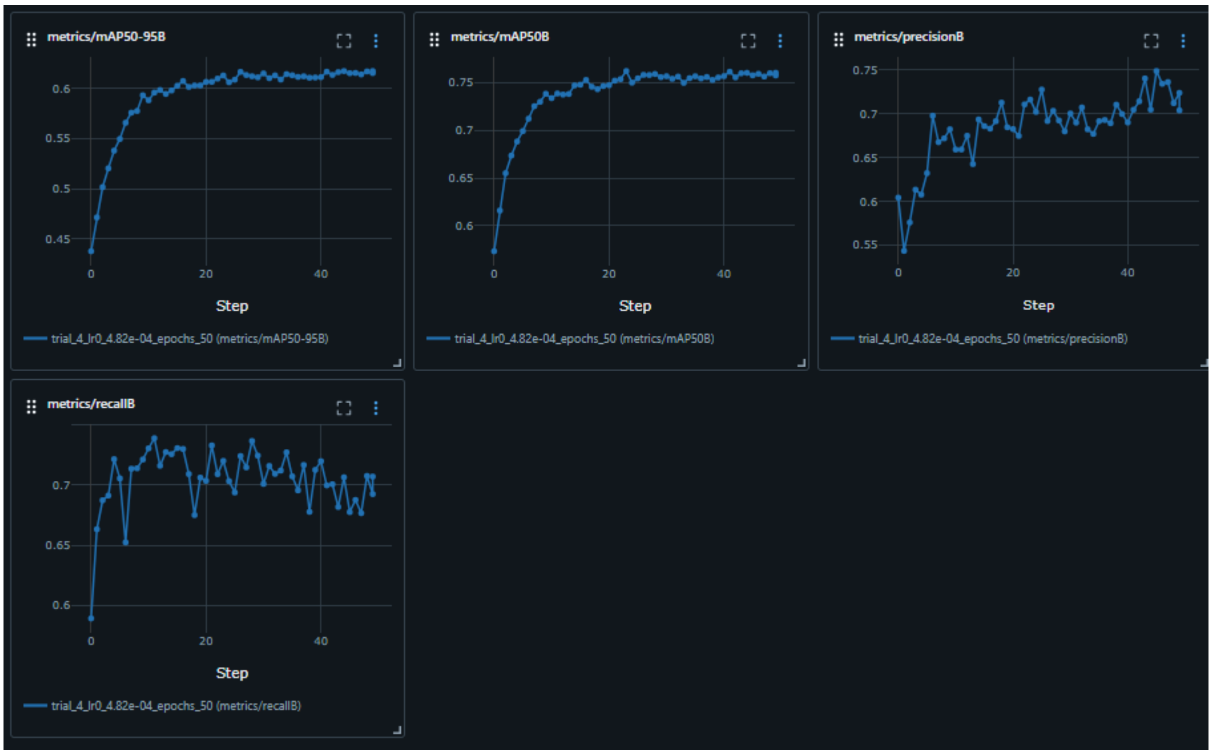


Figure 21: Evaluation metrics of fine-tuned model.

It can be observed in Figure 21 that both mAP50-95 and mAP50 jumped significantly within the first 10 steps (e.g., mAP50 from ~0.66 to ~0.74), then climbed more gradually, suggesting the unfrozen backbone quickly learned coarse dataset-specific features before refining them. Precision remained stable with minor fluctuations, indicating consistent sensitivity to hard negatives like occlusions or adverse weather. Recall increased steadily, with occasional spikes tied to challenging validation batches. Metrics showed no plateau by step 50, suggesting further potential for improvement.

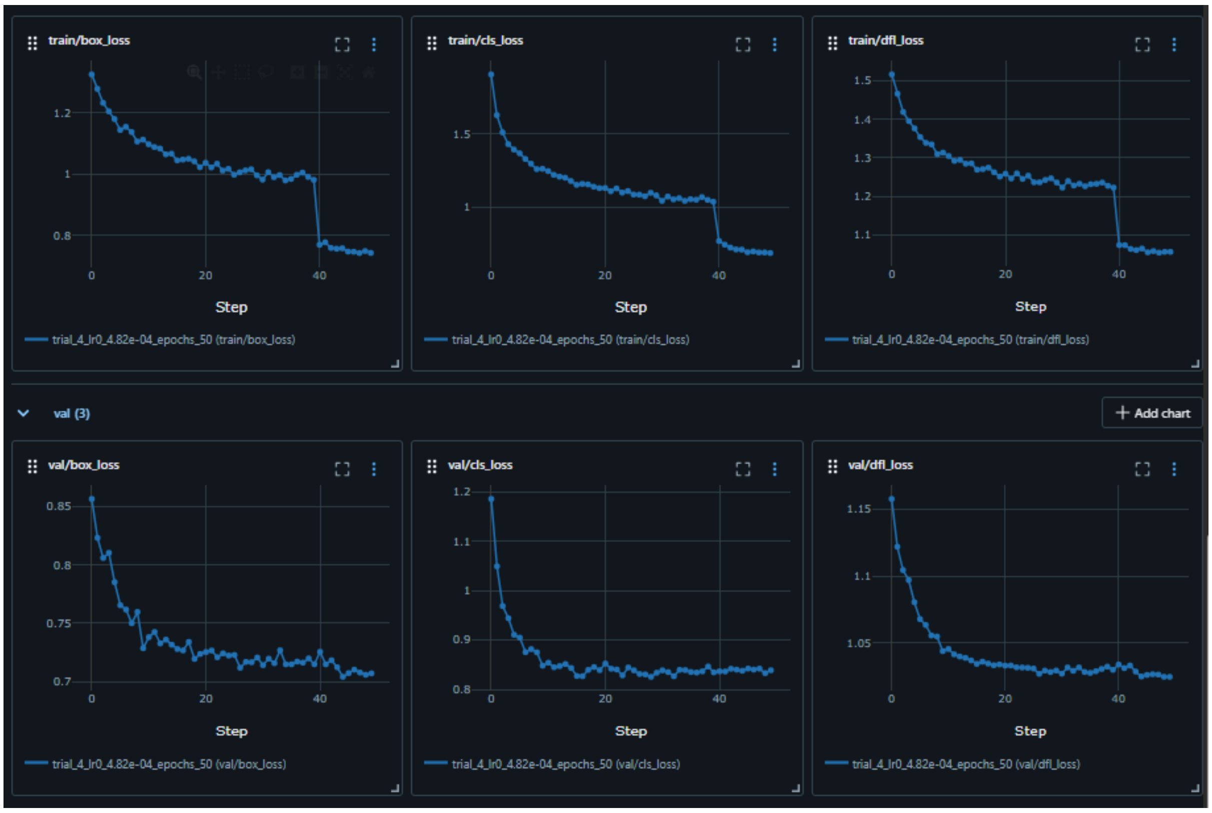


Figure 22: Model's training and validation losses.

All three training losses Observed in Figure 22 declined effectively: train/box_loss to 0.74288, train/cls_loss to 0.68562, and train/dfl_loss to 1.05581. Validation losses stayed close (val/box_loss at 0.70709, val/cls_loss at 0.8388, val/dfl_loss at 1.02443), confirming strong generalization despite the unfrozen backbone.

Compared to Phase 3, Phase 4 achieved significant gains in mAP50 (from ~0.698 to 0.76067) and mAP50-95 (from ~0.510 to 0.61728), highlighting improved localization accuracy and robustness. The increase in recall (from ~0.653 to 0.70718) indicates the model detected more objects, while the slight drop in precision (from ~0.712 to 0.70402) suggests a minor trade-off in false positives. The close alignment between training and validation losses demonstrates effective learning without overfitting, even with unfrozen layers.

The absence of plateauing in metrics by step 50, unlike Phase 3's plateau around step 40, suggests the unfrozen backbone unlocked additional learning capacity. This enabled the model to better capture dataset-specific nuances, such as CCTV lighting or weather conditions, previously limited by the frozen backbone. However, while these trial metrics are promising, they alone are not sufficient to fully confirm the model's performance in real-world security camera systems. To achieve this, the next phase will involve evaluating the model on a held-out test set. This test set, consisting of unseen data, will provide an unbiased assessment of the model's ability to generalize to new scenarios, which is critical for security applications where reliability under varying conditions is essential.

Thus, while phase 4 concludes the model development process with strong results, the upcoming test set evaluation will be key to validating its real-world applicability and robustness for deployment in security camera systems.

VI.1.5. Phase 5: Evaluation on Held-out test set.

The objective of this phase is to evaluate the fine-tined and optimized YOLOv8m models on a held-out test set to assess its real-world application capabilities and robustness for security camera systems. By testing on unseen data, this phase ensures that the model generalizes effectively to diverse environmental conditions encountered in practical deployment scenarios, concluding the development process. The Two models were evaluated: the fine-tuned model from Phase 3 (frozen backbone) in Table 11 and class wise metrics in Table 12 and the optimized model from Phase 4 (unfrozen backbone with best hyperparameters) in Table 13 and Table 14. The held-out test set comprised 203 images with 1,038 instances from the unified dataset.

Table 11: Fine-tuned Model metrics

Metric	**Value**
mAP50	~ 0.526
mAP50-95	~ 0.453
Precision	~ 0.647
Recall	~ 0.534

Table 12: Fine-tuned Model metrics

Class	**Instances**	**Precision**	**Recall**	**mAP50**	**Map50-95**
Bus	100	0.955	0.644	0.802	0.751
Car	403	0.580	0.760	0.661	0.613
Motorcycle	75	0.334	0.453	0.27	0.112
Person	403	0.743	0.164	0.244	0.178
Truck	57	0.623	0.649	0.653	0.612

Table 13: Optimized Model metrics

Metric	**Value**
mAP50	~ 0.914
mAP50-95	~ 0.792
Precision	~ 0.829
Recall	~ 0.867

Table 14: Optimized Model metrics

Class	**Instances**	**Precision**	**Recall**	**mAP50**	**Map50-95**
Bus	100	0.912	0.940	0.972	0.923
Car	403	0.836	0.933	0.953	0.884
Motorcycle	75	0.682	0.800	0.838	0.586
Person	403	0.841	0.717	0.835	0.644
Truck	57	0.874	0.947	0.970	0.925

The fine-tuned model's test set performance as shown in Table 11 was lower than its Phase 3 validation results shown in Table 8 indicating limited generalization due to the frozen backbone. In contrast, the optimized model achieved a significantly higher mAP50 of 0.914 and mAP50-95 of 0.792 in Table 13, surpassing its Phase 4 validation metrics (mAP50: 0.7607, mAP50-95: 0.6173) in Figure 22. This improvement reflects the effectiveness of unfreezing the backbone and optimizing hyperparameters, enabling better adaptation to the unified dataset's diversity.

Class-wise analysis shows the optimized model's balanced performance across all classes, with notable gains in challenging classes like 'motorcycle' (mAP50: 0.271 to 0.838, mAP50-95: 0.112 to 0.586) and 'person' (mAP50: 0.244 to 0.835, mAP50-95: 0.178 to 0.644) in Table 14 and Table 12. Larger classes like 'bus' and 'truck' achieved near-perfect scores (mAP50: 0.972, 0.970; mAP50-95: 0.923, 0.925), demonstrating robustness for distinct objects. The inference speed of 12.9 ms per image (approximately 78 FPS) for the optimized model remains suitable for real-time security applications, slightly slower than the fine-tuned model's 12.5 ms but still well above the 30 FPS threshold.

VI.1.6. Addressing Overfitting concerns

The optimized model's high test set metrics initially raised concerns about overfitting. However, several factors support that the model is not overfitted:

The test set is held out and unseen during training, ensuring an unbiased evaluation. Also, the consistent improvement from Phase 3 (validation mAP50: ~0.698) to Phase 4 (validation mAP50: 0.7607) to Phase 5 (test mAP50: 0.914) reflects genuine learning, as does the fine-tuned model's lower test performance (mAP50: 0.526), which confirms the optimized model's superior adaptation. In Phase 4, training and validation losses were closely aligned (e.g., train/box_loss: 0.7429 vs. val/box_loss: 0.7071), indicating good generalization during training.
Balanced improvements across all classes, including challenging ones like 'motorcycle' and 'person', suggest the model has not overfit to specific patterns but rather learned robust features. The high-test set performance may partly result from the test set being less challenging than the validation set (e.g., fewer extreme occlusions or lighting conditions), but this aligns with typical security camera scenarios where conditions may be more controlled.

VI.2. Model 2: SSDLite MOBILENETV3 LARGE

For the SSDLite MobileNetV3 large model (Howard et al., 2019), the two approaches will be considered and will follow the same chosen in the yolo. An SSDlite (Howard et al., 2019) with pre-trained weights is loaded to utilize prior knowledge of patterns and features, however this time one with the MSCOCO dataset(Lin et al., 2014). Therefore, the process is structured into three phases to iteratively enhance the model's performance across diverse datasets and conditions, with metrics tracked by Mlflow (Zaharia et al., 2018). The same dataset as the yolo will be used although it will be converted to a tensor format for it to be used by the SSDLite (Howard et al., 2019)

The SSDLite320 MobileNetV3 Large model (Howard et al., 2019) is a lightweight object detection framework processing input image at 320x320 pixels. With approximately 3.22 million parameters and 1.02 billion floating-point operations (FLOPs) per inference, it offers an efficient trade-off between accuracy and computational cost, ideal for real-time detection on low-power hardware.

VI.2.1. Phase 1: Evaluation on unified dataset

The objective here is to assess the baseline performance of the pre-trained SSDLite model on the unified dataset without additional training. The SSDLite320 MobileNetV3 large model, pre-trained on MSCOCO(Lin et al., 2014), was loaded and used to perform inference on the unified dataset. No further training was conducted in this phase, focusing solely on evaluation out-of-the-box performance. Due to the setup's limitation, only a few metrics were tracked which is sufficient for evaluation though MLflow (Zaharia et al., 2018)

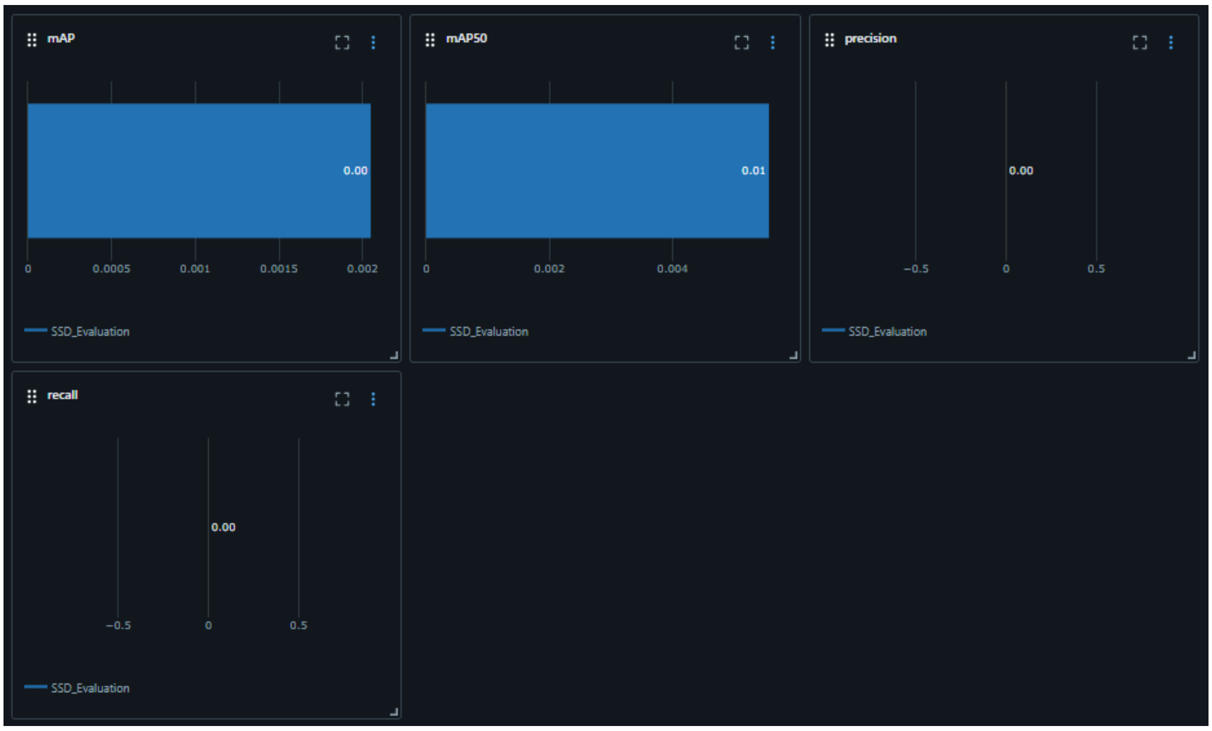


Figure 23: SSD model evaluation.

Table 15: SSD Evaluation metrics

mAP	mAP50	Precision	Recall
~ 0.0020	~ 0.0056	0	0

The model's performance on the unified dataset was significantly different from its performance on the COCO dataset (Lin et al., 2014), despite performing well on it. The extremely low mAPs along with zero precision and recall as seen in Figure 23 and corresponding values in Table 15 show that there is a substantial domain shift due to the diverse conditions in the unified dataset. To adjust the model to the new data, this necessitates fine-tuning.

VI.2.2. Phase 2: Fine-Tuning on Unified Dataset

The objective is to adapt the model to the unified dataset by fine-tuning the detection head while keeping the backbone frozen. The pre-trained model was fine-tuned for thirty steps, updating only the detection head with the backbone layers frozen. Hyperparameters included a learning rate of 0.001 which includes a warmup, a batch size of sixteen, the optimizer used was the default SGD (stochastic gradient decent) and a weight decay of 0.0005.

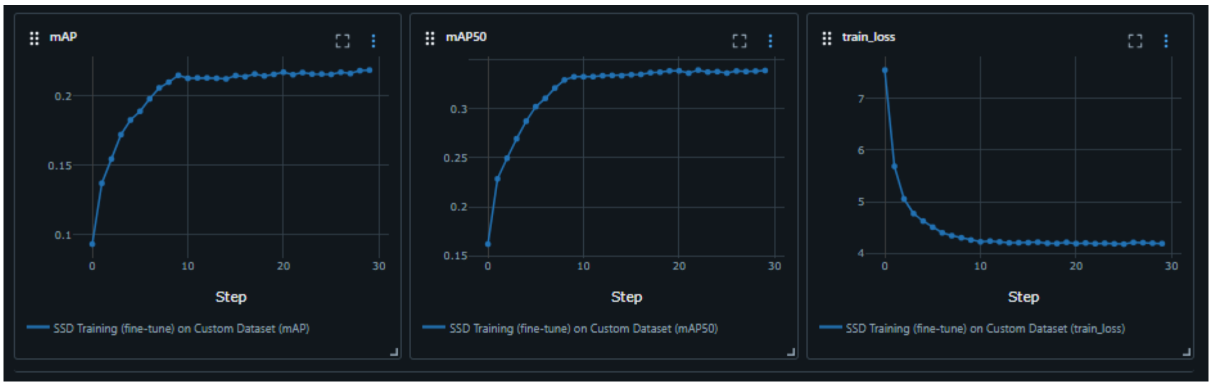


Figure 24: SSD model training dynamics.

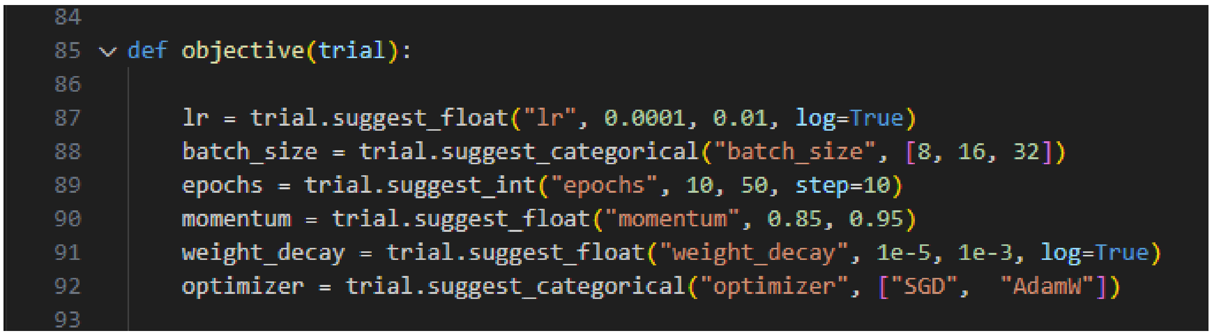
Table 16: SSD training dynamics values

Train loss	mAP	mAP50
~ 4.189	~ 0.218	~ 0.338

Fine-tuning improved performance, with mAP rising to 0.218 and mAP50 to 0.338 seen in Figure 24 and corresponding values in Table 16, although these values are still modest compared to the model's performance on COCO(Lin et al., 2014). The training loss suggests that further optimization is needed to bridge the gap between the unified dataset and COCO(Lin et al., 2014) , prompting optimization to get optimal hyperparameters for the model.

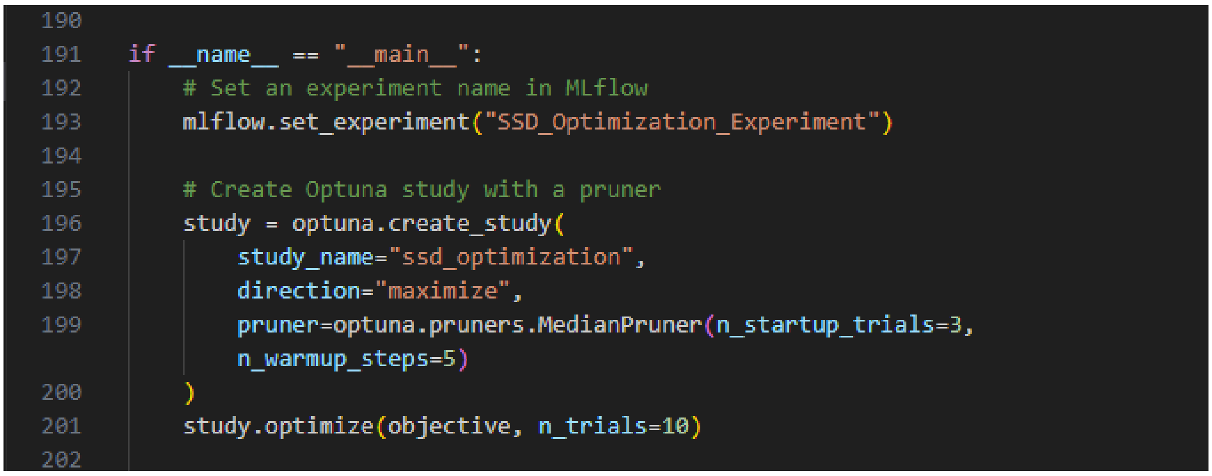
VI.2.3. Phase 3: Hyperparameter Optimization and Final Fine-Tuning

The objective here is to optimize the model's performance through hyperparameter tuning and further fine-tuning. To maximize the performance, ten experiments were conducted with varying hyperparameters. With Trial 1 yielding the best results, achieving optimized parameters after fine-tuning. The same hyperparameters from phase 2 were used as a baseline, with adjustments assessed across trials as seen in Figure 25 and Figure 26 respectively.



```python
def objective(trial):

    lr = trial.suggest_float("lr", 0.0001, 0.01, log=True)
    batch_size = trial.suggest_categorical("batch_size", [8, 16, 32])
    epochs = trial.suggest_int("epochs", 10, 50, step=10)
    momentum = trial.suggest_float("momentum", 0.85, 0.95)
    weight_decay = trial.suggest_float("weight_decay", 1e-5, 1e-3, log=True)
    optimizer = trial.suggest_categorical("optimizer", ["SGD",  "AdamW"])

```

Figure 25: Hyperparameters Optimized



```python

if __name__ == "__main__":
    # Set an experiment name in MLflow
    mlflow.set_experiment("SSD_Optimization_Experiment")

    # Create Optuna study with a pruner
    study = optuna.create_study(
        study_name="ssd_optimization",
        direction="maximize",
        pruner=optuna.pruners.MedianPruner(n_startup_trials=3,
        n_warmup_steps=5)
    )
    study.optimize(objective, n_trials=10)

```

Figure 26: Optuna Trial implementation.

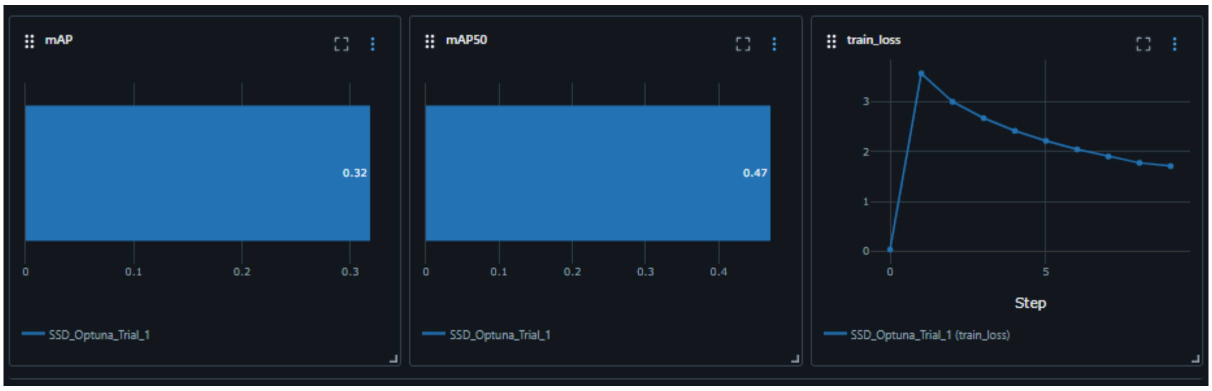


Figure 27: Optimized model evaluation.

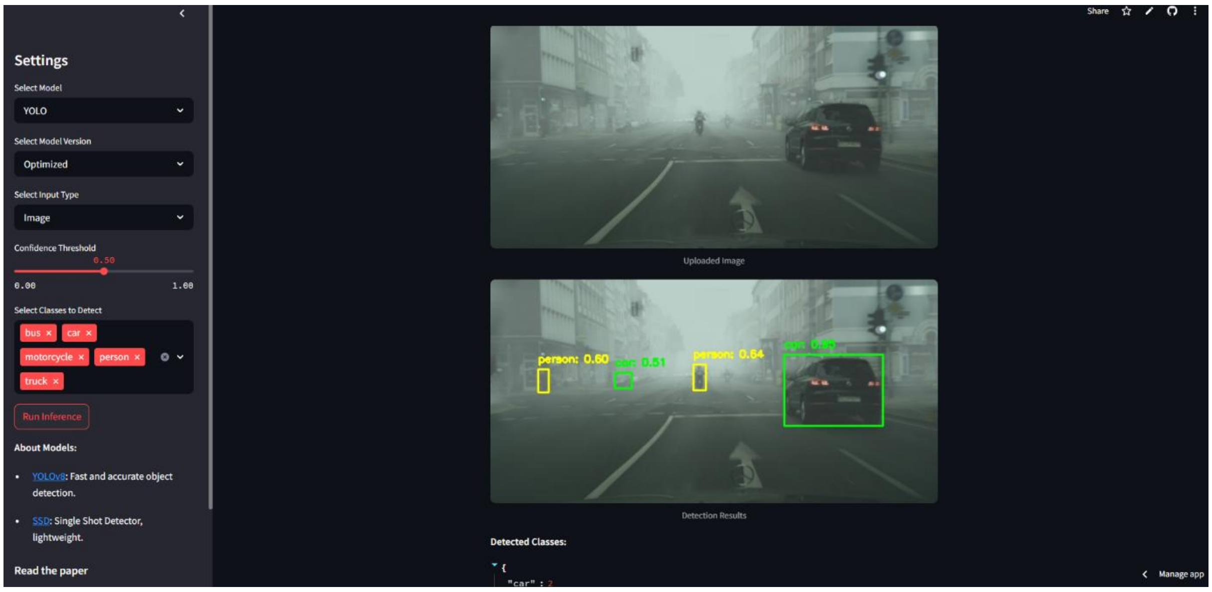
Table 17: Optimized model evaluation value

Train loss	**mAP**	**mAP50**
~ 1.712	~ 0.318	~ 0.471

The best trial significantly improved performance as observed in Figure 27 and corresponding values in Table 17, with the mAP reaching to 0.318 and mAP50 at 0.471, and training loss dropping to 1.712. These results indicate successful adaptation to the unified dataset, though performance plateaus here, as further hyperparameter tuning yielded the same metrics. Compared to COCO, the model now performs well, balancing efficiency and accuracy.

VII. RESULTS

To complement the theoretical and experimental evaluations of the base and optimized versions of the models presented in this thesis, a web application was developed using Streamlit (Streamlit, 2022), SecureDetect. This application provides an interactive platform for testing and comparing object detection models in practical, user-friendly environment. The feature of the app includes model selection, users can choose between the two models either the base or optimized version, it supports three input types: images, videos and Youtube (Google, 2025) links to make it versatile for testing various scenarios. Once a model is and input is selected, the app performs inference and displays annotated outputs. The app also provides quantitative feedback, including the average confidence, inference time (for images), and frames-per-second (FPS) processing rate for videos. It also counts the number of detections per class, offering insights into model behaviour.



Settings

Select Model: YOLO

Select Model Version: Optimized

Select Input Type: Image

Confidence Threshold: 0.50 (0.00 – 1.00)

Select Classes to Detect: bus, car, motorcycle, person, truck

About Models:

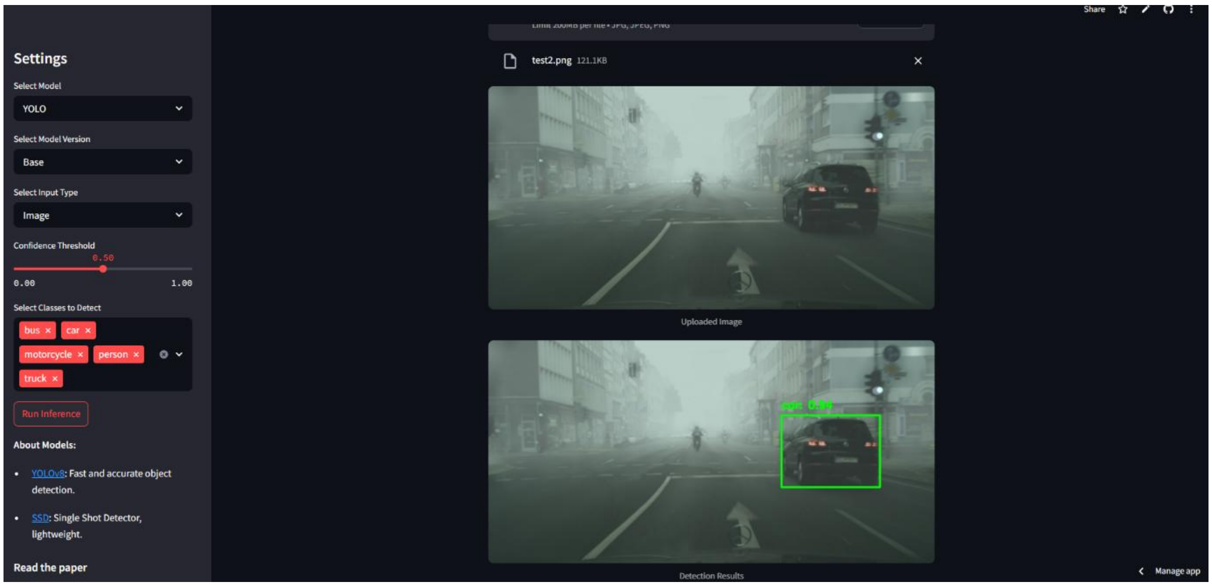
- YOLOv8: Fast and accurate object detection.
- SSD: Single Shot Detector, lightweight.

Read the paper

Detected Classes:

```
{
  "car" : 2
```

Figure 28: YOLOv8 optimized model's performance.


Settings

Select Model: YOLO

Select Model Version: Base

Select Input Type: Image

Confidence Threshold: 0.50 (0.00 – 1.00)

Select Classes to Detect: bus, car, motorcycle, person, truck

About Models:

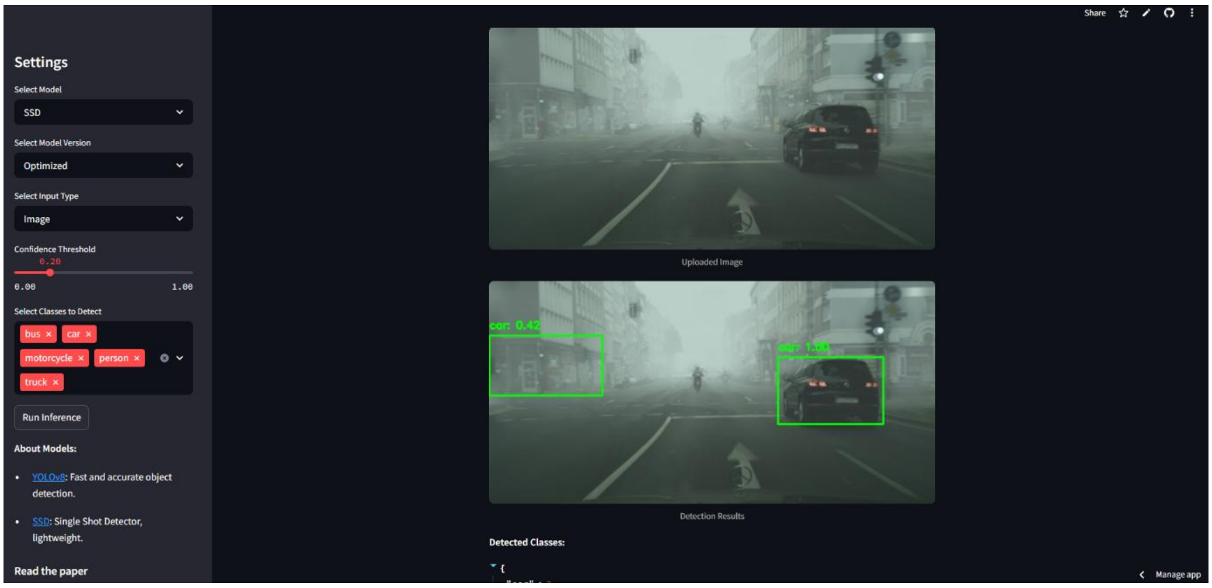
- YOLOv8: Fast and accurate object detection.
- SSD: Single Shot Detector, lightweight.

Read the paper

Limit 200MB per file • JPG, JPEG, PNG

test2.png 121.1KB

Figure 29: YOLOv8 base model's performance.


Settings

Select Model: SSD

Select Model Version: Optimized

Select Input Type: Image

Confidence Threshold: 0.20 (0.00 – 1.00)

Select Classes to Detect: bus, car, motorcycle, person, truck

About Models:

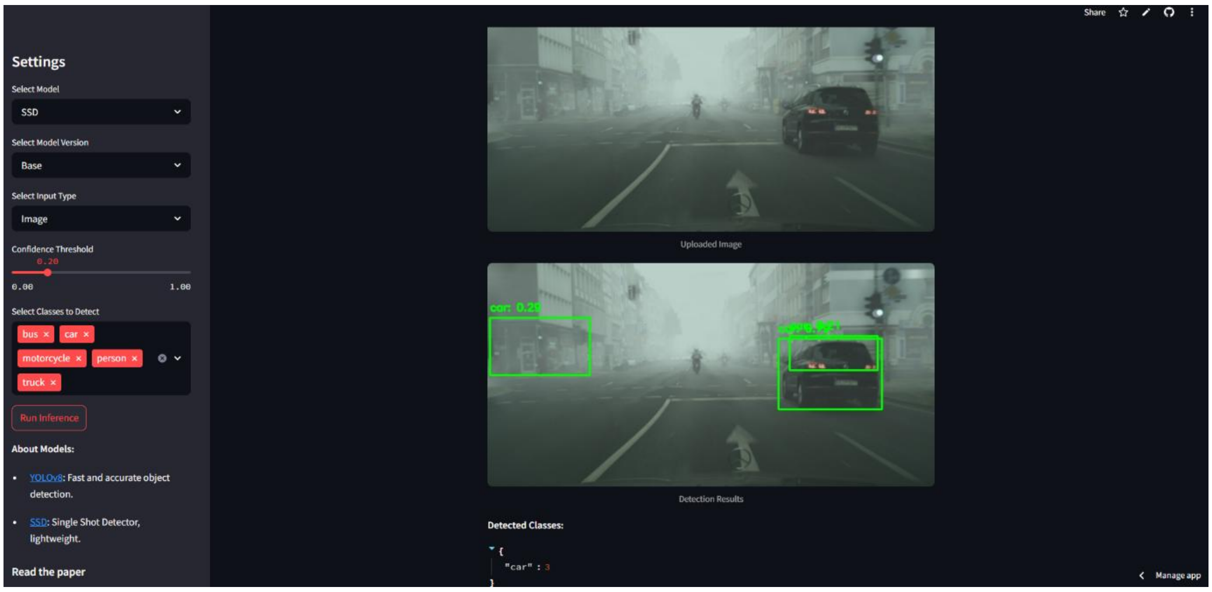
- YOLOv8: Fast and accurate object detection.
- SSD: Single Shot Detector, lightweight.

Read the paper

Detected Classes:

```
{
```

Figure 30: SSD optimized model's performance.


Settings

Select Model: SSD

Select Model Version: Base

Select Input Type: Image

Confidence Threshold: 0.20 (0.00 – 1.00)

Select Classes to Detect: bus, car, motorcycle, person, truck

About Models:

- YOLOv8: Fast and accurate object detection.
- SSD: Single Shot Detector, lightweight.

Read the paper

Detected Classes:

```
{
  "car" : 3
}
```

Figure 31: SSD base model's performance.

A critical aspect of the application's output is the confidence score assigned to each detection, it is the probability value (ranges from 0 to 1) that reflects the model's certainty that a detected object belongs to the predicted class and that the bounding box accurately encloses it. It is very vital for evaluating model performance because they directly influence the trade-off between precision and recall.

$$Confidence = Objectness\ Score * Class\ Probability$$

Comparative Analysis

The YOLOv8m optimized model significantly outperformed the SSDLite MobileNetV3 Large model, it was able to detect all objects in Figure 28 in terms of accuracy, with mAP50 of 0.914 compared to SSDLite's 0.471 which just detects one object correctly and misclassifies the building in Figure 31. However, SSDLite's lightweight architecture shown in Table 18 and faster inference time make it more suitable for edge devices where computational resources are limited. Although both optimized models outperformed their base model as seen in Figure 29 and Figure 31 respectively.

Table 18: comparative analysis of YOLOv8m (optimized) and SSD (optimized)

Model	**mAP50**	**mAP**	**Inference Time(ms)**	**Parameters (M)**
YOLOv8 (optimized)	0.914	0.792	12.9	25.9
SSDLite (optimized)	0.471	0.318	10	3.22

VIII. CONCLUSION

This thesis aimed to enhance object detection for security camera systems, addressing the critical need for reliable performance across diverse indoor and outdoor environmental conditions. The research objectives were met through a systematic five-phase approach for YOLOv8(Jocher Glenn et al., 2023) and a three-phase approach for SSDLite MobileNetV3 Large(Howard et al., 2019), which led to the development of models resilient to challenges such as varying lighting, adverse weather and occulusions. Leveraging pre-trained weights, fine-tuning, and hyperparameter optimization with Optuna, the optimized YOLOv8's model demonstrated superior performance, achieving mAP50 of 0.914 and mAP50-95 of 0.792 on a held-out test set, surpassing the fine-tuned model's 0.526 and 0.453, respectively. SSDLite MobileNetV3 Large, designed for efficiency, reached mAP50 of 0.471, demonstrating its suitability for real-time use on edge devices. The development of SecureDetect, a Streamlit-based web app validated these models' practical utility, offering an interactive platform for testing across various input types and validating the model's effectiveness in simulated security scenarios.

These findings are not just about performance metrics, it extends beyond that. The Optimized YOLOv8's high accuracy reflects the potential of tailored optimization to elevate detection reliability, a cornerstone for security applications where false alarms can have consequences. It is ability to maintain vigorous performance in challenging conditions as can be seen by the class wise gains, for example the motorcycle improving from mAP50-95 of 0.112 to 0.586 which underscores the value of unfreezing the backbone layers to capture domain specific features. On the other hand, SSDLite's while less precise than YOLOv8, it is lightweight architecture (3.2M parameters vs YOLOv8m's 25.9M) and faster inference make it an ideal candidate for resource constrained environments. This duality reveals a broader insight such that the choice of model depends not only on performance goals but also on deployment context as well, a particularly important thing to consider in real-world security system application.

In Conclusion, this thesis contributes more than just optimized models, it offers a robust and replicable framework for advancing real-time object detection in resource constrained and environmentally diverse settings. By providing a practical testing interface and achieving significant performance on these models enhances security surveillance and lays a foundation for future research. Beyond security, the optimization framework established; combining transfer learning, fine-tuning and hyperparameter tuning holds promise for other domains. It challenges the field to consider how well these models perform and adapt to complexities of the real world, making way for a smarter and more reliable system across industries.

IX. REFERENCES

Akiba, T. a. (2019). Optuna: A Next-generation Hyperparameter Optimization Framework. Proceedings of the 25th {ACM} {SIGKDD} International Conference on Knowledge Discovery and Data Mining.

Amazon. (2024, October 20). What is Deep Learning. Retrieved from AWS: https://aws.amazon.com/what-is/deep-learning/

Annan, J. (2024, june 15). Single Shot MultiBox Detector (SSD) Explained by A.J. Retrieved from Medium: https://medium.com/@jesse419419/single-shot-multibox-detector-ssd-explained-by-a-j-dda10ba42a29

Apache Software Foundation. (2024, July 8). Hyperparameter tuning with Optuna. Retrieved from databricks: https://docs.databricks.com/aws/en/machine-learning/automl-hyperparam-tuning/optuna

Bali, R. (2024, February 17). Exploring Object detection with R-CNN models. Retrieved from Medium: https://towardsdatascience.com/exploring-object-detection-with-r-cnn-models-a-comprehensive-beginners-guide-part-2-685bc89775e2

Bergstra, J. Y. (2013, January 9). Making a Science of Model Search: Hyperparameter Optimization in Hundreds of Dimensions for Vision Architectures. Proceedings of the 30th International Conference on Machine Learning (pp. 115-123). Atlanta, Georgia, USA: PMLR. Retrieved from Github: https://github.com/jaberg/hyperopt

Dalal Navneet, T. B. (2005). Histograms of Oriented Gradients for Human Detection. IEEE.

Data Science Wizards. (2023, July 7). Understanding the AdaBoost Algorithm | by Data Science Wizards. Retrieved from Medium: https://medium.com/@datasciencewizards/understanding-the-adaboost-algorithm-2e9344d83d9b

Dollár, T.-Y. L. (2015). Microsoft COCO: Common Objects in Context.

Gavrilescu, R. a. (2018). Faster R-CNN:an Approach to Real-Time Object Detection. 2018 International Conference and Exposition on Electrical And Power Engineering (EPE), 0165-0168.

GeeksForGeeks. (2024, May 26). Introduction to Deep Learning. Retrieved from GeeksForGeeks: https://www.geeksforgeeks.org/introduction-deep-learning/

GeeksforGeeks. (2025, March 11). Hyperparameter tuning . Retrieved from GeeksforGeeks: https://www.geeksforgeeks.org/hyperparameter-tuning/

Girshick, R. (2015). Fast R-CNN. 2015 IEEE International Conference on Computer Vision (ICCV), 1440-1448.

Girshick, R. a. (2014). Rich Feature Hierarchies for Accurate Object Detection and Semantic Segmentation. 2014 IEEE Conference on Computer Vision and Pattern Recognition (pp. 580-587). IEEE.

Google. (2024, October 19). What is Deep Learning? Retrieved from Google Cloud: https://cloud.google.com/discover/what-is-deep-learning

IBM. (2024, October 22). What are convolutional neural networks? Retrieved from IBM: https://www.ibm.com/topics/convolutional-neural-networks

IBM. (2024, October 21). What is a neural network? Retrieved from IBM: https://www.ibm.com/topics/neural-networks

Jacob Murel, E. K. (2024, September 20). What is Object Detection? Retrieved from IBM: https://www.ibm.com/topics/object-detection

Jia, Y. a. (2014). Caffe: Convolutional Architecture for Fast Feature Embedding. MM 2014 - Proceedings of the 2014 ACM Conference on Multimedia.

Jim Holdsworth, M. S. (2024, June 17). what is deep learning. Retrieved from IBM: https://www.ibm.com/topics/deep-learning

Jim Holdsworth, M. S. (2024, June 17). What is deep learning. Retrieved from ibm.com: https://www.ibm.com/topics/deep-learning

Joseph Redmon, A. F. (2018). YOLOv3: An Incremental Improvement. Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR). IEEE.

Kenk, M. A. (2020). DAWN: Vehicle Detection in Adverse Weather Nature Dataset. arXiv preprint arXiv:2008.05402. Retrieved from https://arxiv.org/abs/2008.05402

Krizhevsky, A. a. (2012). ImageNet Classification with Deep Convolutional Neural Networks. Neural Information Processing Systems.

Liu, W. a.-Y. (2016). SSD: Single Shot MultiBox Detector. IEEE.

Lowe, D. (1999). Object Recognition from Local Scale-Invariant Features. IEEE.

Mehra, A. a. (2021). ReViewNet: A Fast and Resource Optimized Network for Enabling Safe Autonomous Driving in Hazy Weather Conditions. IEEE Transactions on Intelligent Transportation Systems, 4256-4266.

Parti, A. (2024, March 18). Understanding Object Detection: A Comprehensive Guide. Retrieved from Pareto AI: https://pareto.ai/blog/object-detection

Redmon, J. a. (2016). 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR). IEEE.

Shaoqing Ren, K. H. (2015). Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks. IEEE Transactions on Pattern Analysis and Machine Intelligence, 1137-1149.

Uijlings, J. a. (2013). Selective search for object recognition. International journal of computer vision, 154-171.

Viola P, J. M. (2001). Rapid object detection using a boosted cascade of simple features. IEEE. Retrieved from https://ieeexplore.ieee.org/document/990517

Bergstra, J., Komer, B., Eliasmith, C., Yamins, D., & Cox, D. D. (2015). Hyperopt: A Python library for model selection and hyperparameter optimization. Computational Science and Discovery, 8(1). https://doi.org/10.1088/1749-4699/8/1/014008

Glorot, X., Bordes, A., & Bengio, Y. (2011). Deep sparse rectifier neural networks. Journal of Machine Learning Research, 15.

Howard, A., Sandler, M., Chen, B., Wang, W., Chen, L. C., Tan, M., Chu, G., Vasudevan, V., Zhu, Y., Pang, R., Le, Q., & Adam, H. (2019). Searching for mobileNetV3. Proceedings of the IEEE International Conference on Computer Vision, 2019-October. https://doi.org/10.1109/ICCV.2019.00140

Jocher Glenn, Chaurasia Ayush, & Qiu Jing. (2023). YOLOv8 by Ultralytics. Https://Github.Com/Ultralytics/Ultralytics.

Lin, T. Y., Maire, M., Belongie, S., Hays, J., Perona, P., Ramanan, D., Dollár, P., & Zitnick, C. L. (2014). Microsoft COCO: Common objects in context. Lecture Notes in Computer Science (Including Subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics), 8693 LNCS(PART 5). https://doi.org/10.1007/978-3-319-10602-1_48

Liu, W., Anguelov, D., Erhan, D., Szegedy, C., Reed, S., Fu, C. Y., & Berg, A. C. (2016). SSD: Single shot multibox detector. Lecture Notes in Computer Science (Including Subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics), 9905 LNCS. https://doi.org/10.1007/978-3-319-46448-0_2

Redmon, J., Divvala, S., Girshick, R., & Farhadi, A. (2016). You only look once: Unified, real-time object detection. Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition, 2016-December. https://doi.org/10.1109/CVPR.2016.91

Shao, S., Li, Z., Zhang, T., Peng, C., Yu, G., Zhang, X., Li, J., & Sun, J. (2019). Objects365: A large-scale, high-quality dataset for object detection. Proceedings of the IEEE International Conference on Computer Vision, 2019-October. https://doi.org/10.1109/ICCV.2019.00852

Streamlit. (2022). Streamlit Documentation. Streamlit Inc.

Zaharia, M., Chen, A., Davidson, A., Ghodsi, A., Hong, S. A., Konwinski, A., Murching, S., Nykodym, T., Ogilvie, P., Parkhe, M., Xie, F., & Zumar, C. (2018). Accelerating the Machine Learning Lifecycle with MLflow. Bulletin of the IEEE Computer Society Technical Committee on Data Engineering.

DECLARATION

of the originality of the writing

(According to Code of Studies and Examinations of the University of Pécs, Annex nr. 14/1.*)*

I, the undersigned, **Abimbola Mohammed Ogunsakin (DKM9BE)**, declare under penalty of perjury that every part of my writing, **Object Detection for Security Camera Systems in Indoor and Outdoor Environments** is the result of my own, autonomus work, I only used referred sources (special literature, tools, etc.) and I observed the pertaining rules of the University of Pécs while preparing my writing.

I am aware that the University of Pécs has the right to check the observation of copyright rules through a plagiarism tracing system.

Pécs, 05/03/2025

[illegible]

………………………………………………….

signature of the student