Investigating the Performance of Real-time Object Detection Frameworks YOLOv8, YOLOv9 and YOLOv10 for Object Detection in adverse weather conditions

"How do the real-time object detection frameworks; YOLOv8, YOLOv9, and YOLOv10, comparatively perform in terms of model efficiency in detecting various objects in adverse weather conditions? "

A Computer Science Extended Essay

Word Count: 3,984
Session: November 2024

Table Of Contents

1 Introduction

1.1 Context and Scope

There is an extensive use of Computer Vision (CV) in today's day of age, and the proliferation of its use has advanced. CV is often used to substitute manual labour in organisations and enterprises. One such interesting domain is Rescue Robots. Integrating CV into Search and Rescue missions (SAR) facilitates their deployment in disaster relief and search operations *(Flynn et al).*

CV is made up of Neural Networks (NNs). NNs are capable of processing raw data to output valuable information *("Computer Vision")*. Datasets, which consist of labelled data, are fed into these networks to facilitate the creation of scenarios. Numerous other factors influence the prediction's accuracy such as frameworks which the model is built upon. Each framework has unique displayed behaviour when applied to varying datasets *(Li and Luo)*. Consequently, a framework that performs exceptionally well with a dataset might perform poorly with dataset. The variability necessitates the development of multiple frameworks and versions *("Machine Learning Frameworks")*.

For instance, during the aftermath of the meltdown of nuclear reactors in the Fukushima Daiichi Nuclear Powerplant *("Fukushima Daiichi Accident")*. There was an urgent need for rescue robots to *(Westcott)* assess the damage, search for survivors and conduct repairs. The site was also experiencing harsh weather conditions such as heavy rain and strong winds. This made it hard for the available robots to navigate or perform tasks as they were unable to navigate through dust, weather and debris *(Li et al.)*.

Due to the collection of environmental data through hardware such as cameras, there is little to no validation of the data received which could help differentiate and eliminate visuals which can't be interpreted by the model, thus raising a problem statement.

1.2 Research Question

In the recent advances in Artificial Intelligence; (AI) You Only Look Once (YOLO), which is a widely used framework and influential method for object detection (OD) has been brought to significant attention, and is often used for Real-time object detection tasks. Redmon et al introduced YOLO in 2015 since then, scholars have published updated iterations of the concept.

This investigation aims to examine algorithms YOLOv8, YOLOv9 and YOLOv10. Recent versions of YOLO frameworks such as YOLOv10 released in May 2024 and YOLOv9 released in February 2024 lack extensive documentation, particularly in comparison with YOLOv8. Notably, these three frameworks have yet to be compared against each other regarding the robustness of their performance in varying weather conditions.

This investigation will conduct an extensive evaluation of the performances of frameworks; YOLOv8, YOLOv9 and YOLOv10 evaluating these concerning the mean average precision, recall, and precision. Therefore, the research question proposed is: ***"How do the real-time object detection frameworks; YOLOv8, YOLOv9, and YOLOv10, comparatively perform in terms of model efficiency in detecting various objects in adverse weather conditions? "***Although this investigation does not rely on multimodal data inputs nor videos as datasets, it emphasises the detection of objects in images compromised by weather conditions– providing more insight into object detection in still images. With the recognition that video data or multimodal datasets could further enhance the quality of the research, the study aims to maintain simplicity while offering a foundation for future development and research.

1.3 Related work

Though there have been numerous studies on CV there seems to be a lack of documentation with regards to YOLOv10 and YOLOv9. A journal published in ISPRS examines the necessity of specialised enhancements in object detection models for adverse weather conditions but there has been a lack of specificity in the analysis of YOLO *(Toma et al.)*. Similarly, another paper released in 2019 shares a valuable understanding of how YOLO faces challenges in adverse weather conditions and further suggests the use of multimodal approaches– this paper however lacks in-depth research using the latest YOLO models *(Wang et al.)*.

2 Background

2.1 Neural Networks

2.1.1 Definitions

In addition to this, it is recommended to hover through Appendix A and Appendix B to gain a profound understanding of Neural networks and Convolutional neural networks.

Terminology	Definition
Backbone	Extracts crucial features from input dataset through the use of CNNs
Neck	Connects the backbone to the head. Helps aggregate features provided by backbone
Head	Uses features extracted through the neck and backbone to make predictions.
Convolutional Layer	Appliance of convolutional operations to input data to capture spatial hierarchies in images by extracting features. *(LeCun et al.)*
Convolution	The mathematical operation is applied in convolutional layers to filter data by sliding a kernel/filter across input aimed at producing a feature map. *(LeCun et al.)*
Feature maps	Represents various features example, edges and textures. *(LeCun, Bengio, and Hinton)*
Kernal size	Dimension of filter used in a convolutional layer expressed as height multiplied by width. *(LeCun et al.)*
Channel count	Number of channels in an image or features (RGB or feature representations). *(Jain)*
Post-Processing	Techniques applied after a model has done learning. *("References for Papers")*

Activation Function	Mathematical structures applied to a neural network's output to introduce non-linearity, helping the network to model complex patterns. *(Jain)*
Inference	Using a trained machine learning model to make predictions based on new data. *(Jain)*
Inference cost	Computational resources are required to generate predictions for inference. *(Chollet)*
Bottleneck	Series of layers where the dimensionality of feature representation is reduced to compress the information or improve computational efficiency. *(Chollet)*
Anchor free detection	Methods that don't rely on predefined anchor boxes, rather predicting object locations directly. *(Tian et al.)*
Lightweight model	A network designed to be resource-efficient. *(Redmon and Farhadi)*
Up-sample	Process of increasing the spatial resolution of feature maps. *(Odena et al.)*
Downsample	Process of reducing spatial resolution of feature maps. *(LeCun et al.)*
Decoupling	Separation of different aspects of processing to allow more efficient architectural designs.
Spatial reduction	Decreasing spatial dimensions of feature maps. *(LeCun et al.)*
Pointwise convolution	Convolutional operation with kernel *(Redmon and Farhadi)*

Cross-stage partial connections	A technique where feature maps are partially propagated through different stages of the network– enhances gradient flow. *(Wang et al.)*
Spatial Pyramid Pooling Faster	Pools feature maps at multiple scales to improve OD. *(Hirschfeld and Ziemann)*
Non-Maximum Suppression (NMS)	Post-processing technique which eliminates redundant bounding box predictions and selects the ones with high confidence.
Non-uniform matching	Approach to OD where matching between predicted bounding boxes and ground truth boxes are adjusted dynamically or irregularly.
NUMS post-processing	Post-processing technique which leverages non-uniform matching strategies during.
One-to-one head	Each anchor point is assigned a single object label, facilitating simpler matching. *(He et al.)*
One-to-one matching predictions	Predicted objects are matched with one ground truth object during training. *(Zhou et al.)*
Auxiliary framework	Additional components or branches are added to the main model to support the learning process by supplying gradient information and enhancing performance. *(Szegedy et al.)*
Gradient path planning	A technique used to optimise the flow of gradients when backpropagation occurs.

Transformer Model	Relies on self-attention mechanisms to process input in parallel. Efficiently handles sequential data.
Vision transformer	NN architecture that uses transformer models.
Python SDK	Software development kit with a compilation of kits, documentation and libraries.

Table 1: Important Definitions and Terminologies

2.2 You Only Look Once (YOLO)

You Only Look Once (YOLO) algorithm is a family of object detectors that have iterated throughout the years since its initial release by Joseph Redmon in 2015 *(Redmon et al.)*. Each iteration advanced its predecessors by including a novel method. YOLO is notoriously known for its ability to proceed with real-time object detection by dividing the input images into a grid matrix and predicting the bounding boxes along with its class probabilities in parallel *(Redmon et al.)*.

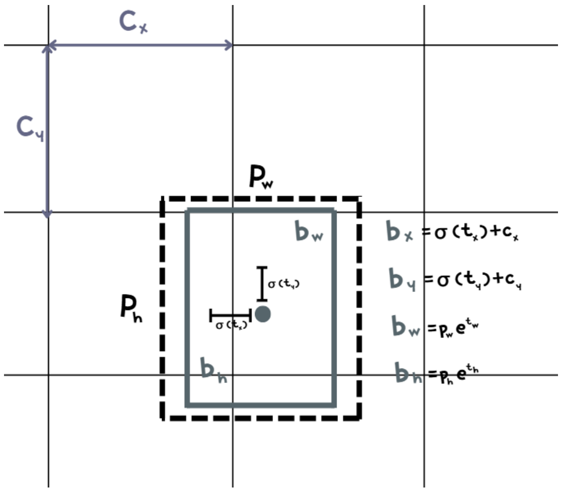


Figure 1: Process of bounding box prediction; centered coordinate prediction; use of sigmoid function; use of the location of the grid cell.

2.2.1 YOLOv8

Compared to its predecessors YOLOv8 includes a newer backbone that consists of a CSPDarknet architecture *(Touvron et al.)* which holds 53 convolutional layers and as well as equips itself with cross-stage partial connections, along with this, YOLOv8 uses the SiLU activation function which mitigates the vanishing gradient problem *(Chen et al.)*. This intensifies information transmission in deep neural networks. The C2f module combines features on a high level with context to enhance detection accuracy. Spatial pyramid pooling faster (SPPF) *(Simonyan and Zisserman)* is another module and the other convolutional layers process the features in variable scales.

In YOLOv8 the head is detachable hence it handles classification, object scores and regression work independently– due to this the overall accuracy is increased. Up sample (U) layers help increase the resolution of provided feature maps. Convolutional layers are included in the head to analyse these feature maps. Overall, the head is designed to optimise its speed and accuracy and hence consideration is given to kernel sizes and channel count of each layer.

YOLOv8 uses anchor-free detection to speed up post-processing (Non-Maximum Suppression) additionally YOLOv8 also includes a newer convolutional layer aimed to detect features using learnable filters. The input layers are detected in variable resolutions and sizes to allow the network to be versatile *(Fischer et al.)*.

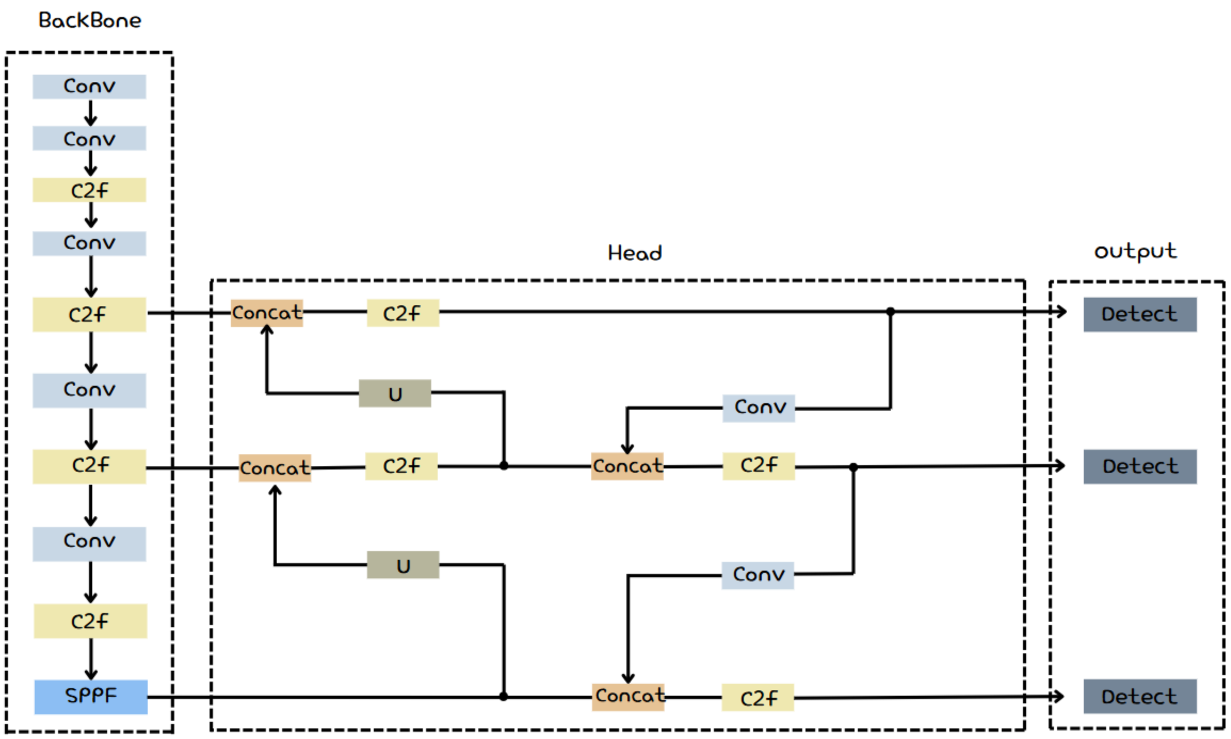


Figure 2: YOLOv8 Architecture

Predecessors of YOLOv8 had equipped a C3 convolutional layer which YOLOv8 replaces using a C2f layer. The C2f layers help utilise all the bottlenecks. Splitting YOLOv8 achieves parallel processing. Additionally, the kernel size has been increased in YOLOv8.

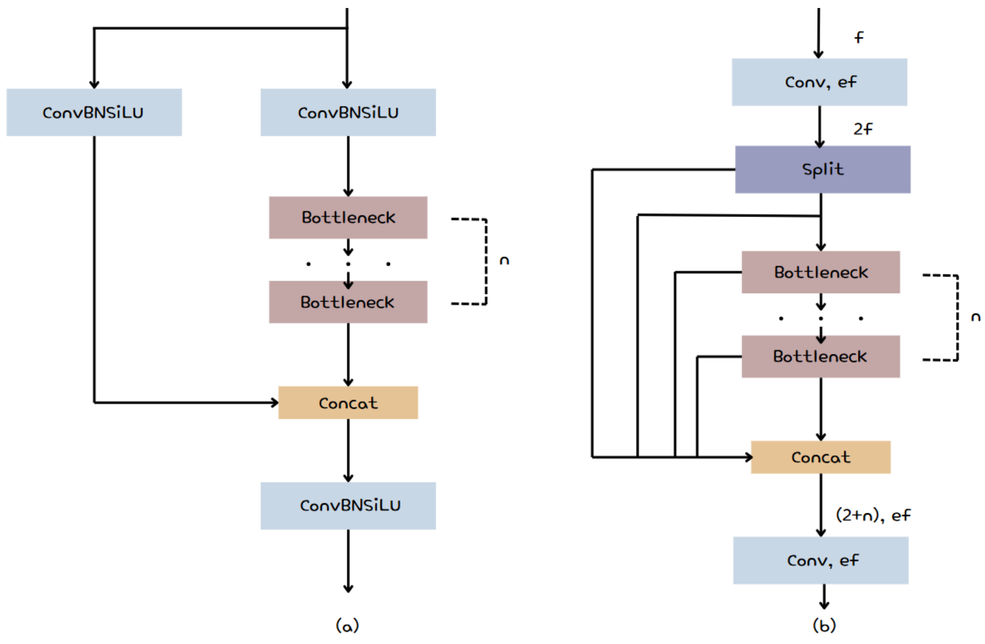


Figure 3: (a) defines bottlenecks in YOLOv5 defined as n (b) defines bottleneck layers in YOLOv8 defined as n

In addition to these architectural features, YOLOv8 employs user accessibility by integrating Python SDK and Command Line Interface (CLI) which further supports programmers to be able to use this model *(Ahmed et al.)*.

2.2.2 YOLOv9

During bottleneck *(Kingma and Ba)* conditions, it can be especially hard for ML models to be able to interpret data. Traditional methods of Masked modelling *(Wang et al.)* and reversible architectures *(Nijkamp et al.)* had to be iterated due to their drawbacks. To address these issues especially the bottleneck situation YOLOv9 has proposed Programmable gradient information (PGI). PGI generates reliable gradient information for model network weight updates. Generalised ELAN *(Jiang et al.)* (GELAN) simultaneously takes multiple parameters such as accuracy, computational efficiency, and speed into account which allows this design to allow programmers to make decisions upon choosing appropriate computational blocks for different devices. The combination of PGI and GELAN gave rise to YOLOv9. YOLOv9 proposes that increasing the model size accumulating more parameters and adding enhanced data transformers can help information retainment– however, this does not address the issue completely.

The introduced auxiliary framework PGI is divided into three components. As shown in the figure 4. PGI is dependent on the main branch and hence it does not require inference costs, the remainder is utilised to solve and precisely investigate important issues in learning methods. Auxiliary reversible branch tries to deal with information bottleneck which occurs when the network is deepened which will cause the loss function to be incapable of providing reliable gradients.

Finally, multi-level auxiliary information handles error accumulation problems which are generally caused due to deep supervision.

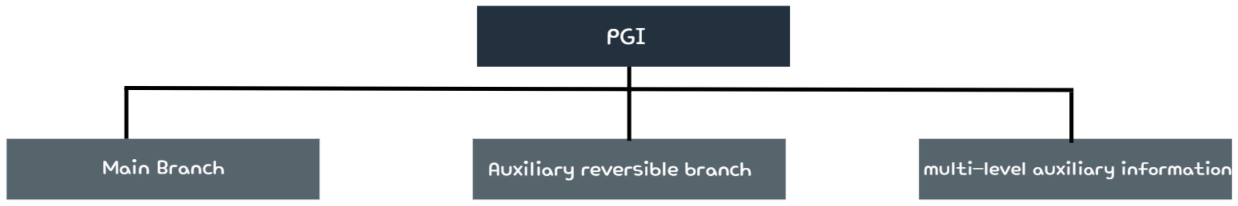

PGI
Main Branch
Auxiliary reversible branch
multi-level auxiliary information

Figure 4: Three divisions of PGI

GELAN is a new network architecture made from the combination of CSPNet *(Ramesh et al.)* and ELAN *(Jiang et al.)* which are innately made with gradient path planning. YOLOv9's GELAN takes into consideration the weight of the model, inference speed, and accuracy. GELAN validates PGI, especially in lightweight models.

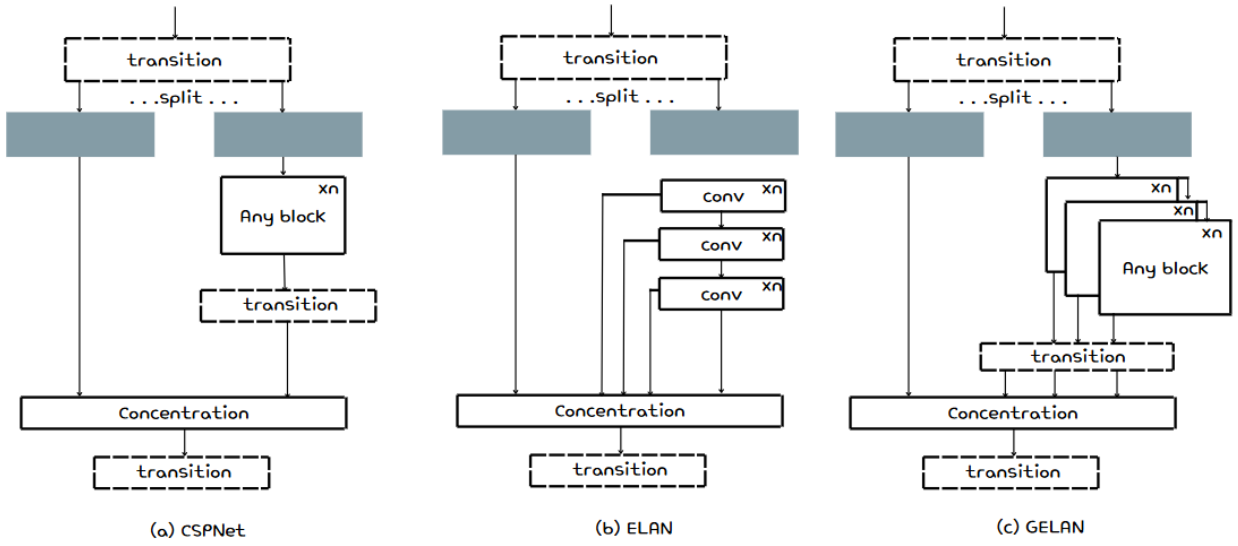


Figure 5: (a) CSPNet (b) ELAN and (c) GELAN architectures

2.2.3 YOLOv10

End-to-end object detection is a shift of methodology from the past where traditional pipelines *(Srivastava et al.)* were used. Baidu's RT-DETR *(Raffel et al.)* is used which is a vision transformer architecture. Hungarian loss is also occupied by it to reach one-to-one matching predictions *(Viana)* and hence it eliminates post-processing.

YOLOs rely on NUMS post-processing to allocate positive samples for each instance to leverage TAL *(Vaswani et al.)* during training– this is detrimental to the model's inference efficiency. YOLOv10 provides an NMS-free training strategy through the use of dual labels and consistent matching metrics *("YOLOv10 Documentation")*.

As shown in Figure 6 the incorporation of one-to-one head is seen. The optimisation objective remains the same as the original branch of one-to-many but this leverages to obtain label assignments *(Xu et al.)*.

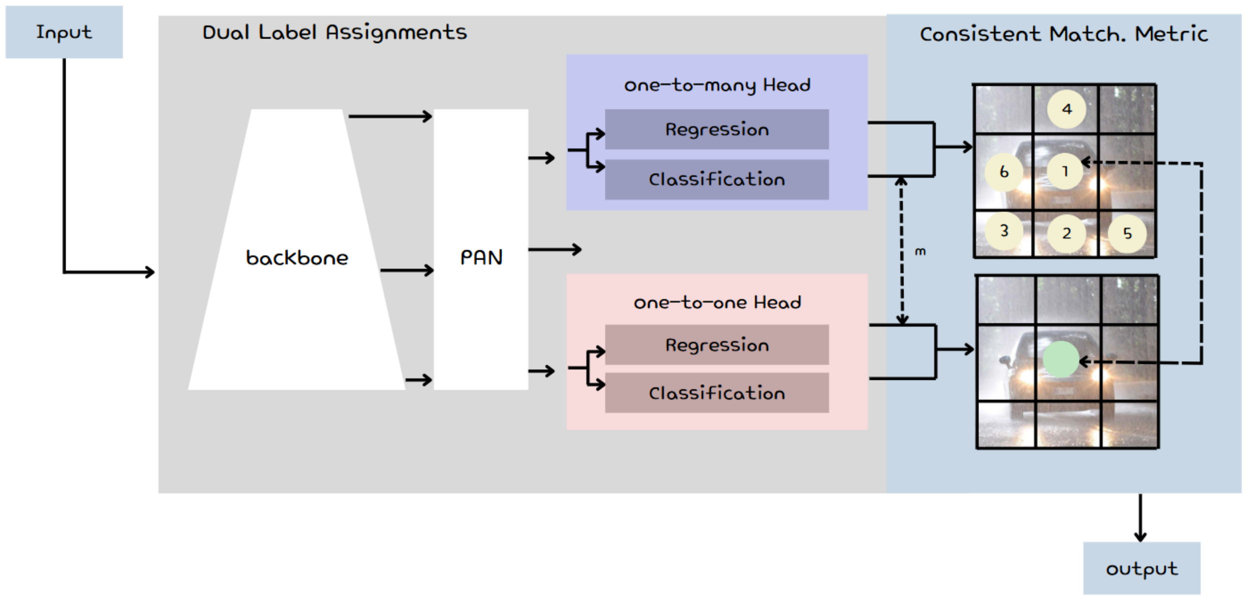


Figure 6: Architecture of YOLOv10

The conjoining of two heads allows the backbone to have optimal supervision *("YOLOv10: An Introduction")*. While inference is employed to avoid inference cost, a one-to-many head is discarded and a one-to-one head is used, which beats Hungarian matching by utilising lesser training time *(Xu et al.)*.

The final regression head takes the significance performance of YOLO compared to the classification head. We can reduce the overhead of the classification head without hurting the performance; therefore, a lightweight architecture is employed *("YOLOv10: Everything You Need to Know")*.

YOLOs use standard convolution stride to achieve spatial downsampling and channel transformation simultaneously. This raises computational costs and parameter counts. YOLOv10 decouples the spatial reduction and channel increase operations. Pointwise convolution is leveraged to modulate the dimension of the channel and further utilize depthwise convolution to achieve spatial downsampling. This maximises the information retained during downsampling as well as computational cost *(Xu et al.)*.

Rank-guided block design scheme decreases the complexity of redundant stages; essentially making a compact model. Compact inverted block (CIB) adopts depthwise convolution for spatial mixing and pointwise convolutions. Rank-guided block allocation helps achieve maximum efficiency while also maintaining good capacity *(Xu et al.)*.

3 Methodology

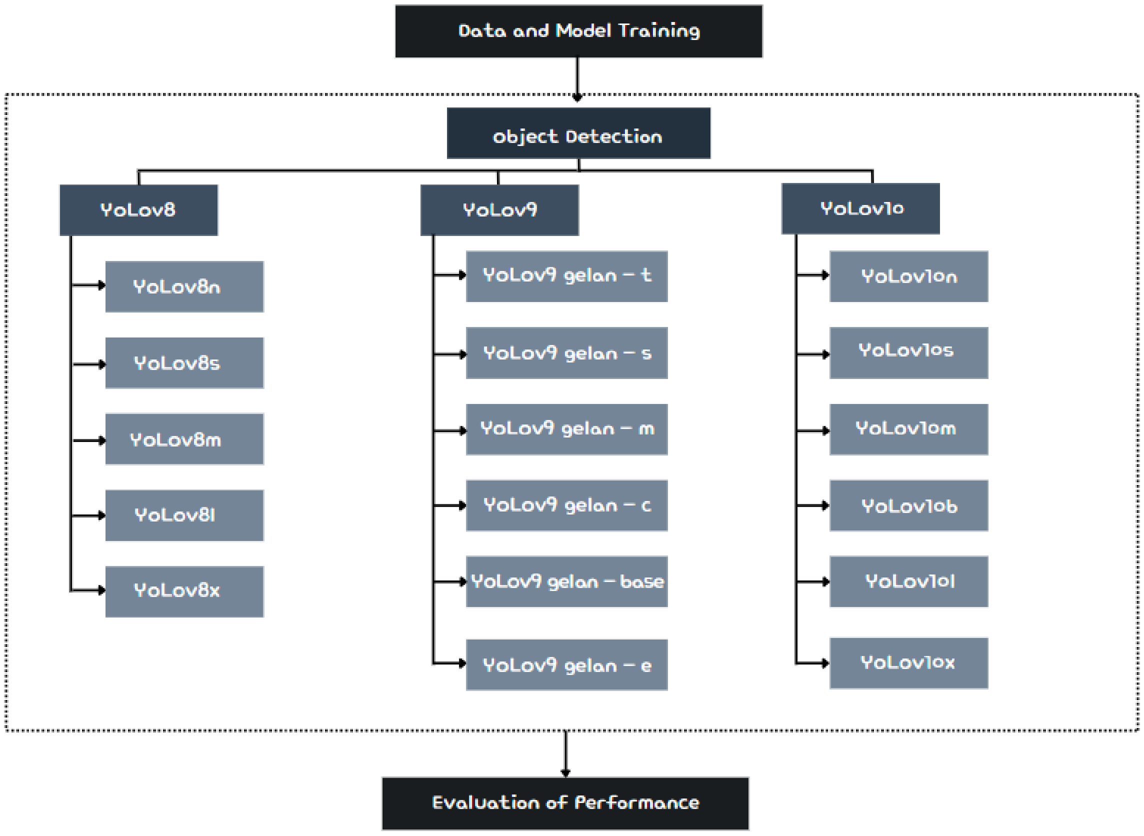


Figure 7: Flow chart of investigation to be proceeded

3.1 Dataset

Roboflow is an online directory of CV model resources, including datasets. It allows the download of these data sets through multiple version formats and these datasets can be easily integrated into Google Colaboratory through the use of APIs. BDD100K is another larger dataset comprising videos and still images of objects in adverse weather conditions. However, due to limitations in hardware along with Roboflow's ease of integrating datasets into Google Colaboratory and its service of cloud-based data storage, Roboflow was utilised for this investigation.

The selected public dataset “O.D IN BAD WEATHER “*("Object Detection in Bad Weather Dataset")* contains images of labelled subjects including Bikes, Buses, Cars, Motors, Persons and Riders. The dataset has innately been pre-processed using auto-orientation and resized to 640x640 pixels by its creator *("Foreign Object Aerodromes")*. Furthermore, it is split into training (815 images) validation (218 images), and test (117 images) subsets.

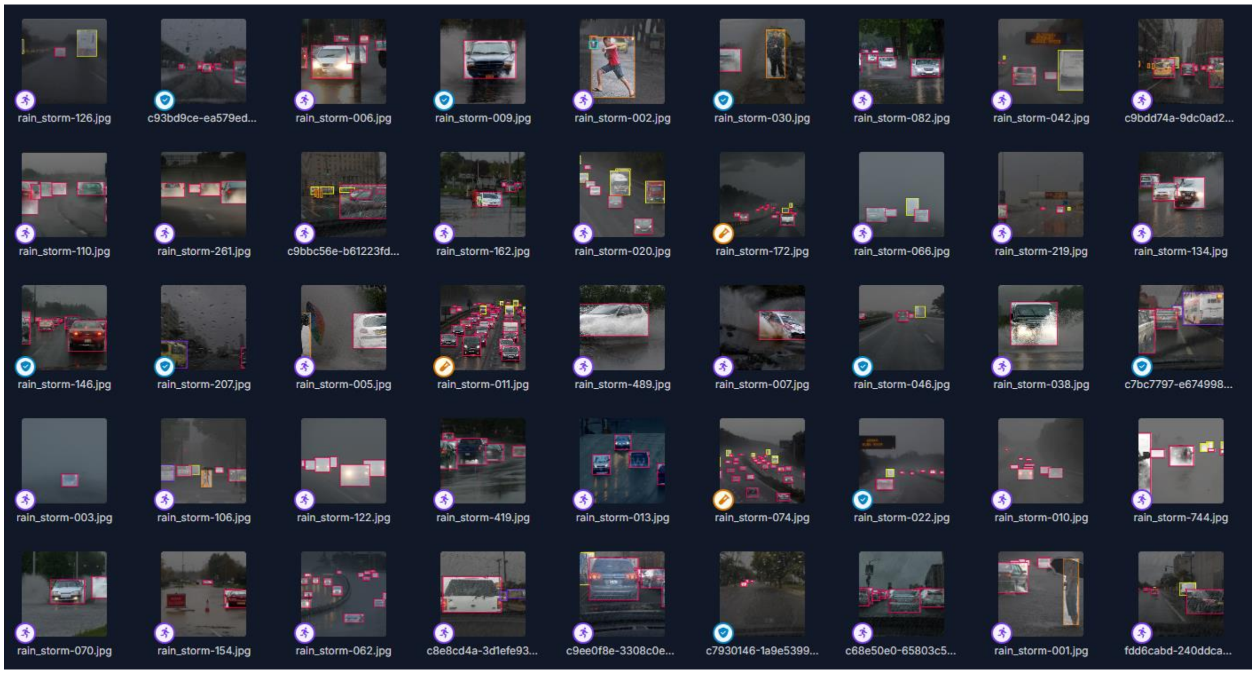


Figure 8: Visual view of the images compiled in the public dataset ("Object Detection in Bad Weather Dataset")

The dataset is integrated from Roboflow by the following snippet:

```
!pip install roboflow

from roboflow import Roboflow
rf = Roboflow(api_key="XXXXX")
project = rf.workspace("XXXXX").project("XXXXX")
version = project.version(x)
dataset = version.download("XX")
```

Figure 9: Integration of Roboflow API

3.2 Variables

During training, the following parameters were kept constant across each framework:

Parameter	Value
Epoch	25
Batch	16
img/imgz	640
Plots	true

Table 2: Showcases the parameters and chosen value

The dataset used remained constant throughout the investigation, with only the dataset version differing, which does not affect the final result; these variables act independently in this investigation.

The dependent variables are the obtained results from each framework post-training. These results are retrieved by visualising the trained models. The results are stored in the training results directory in the workspace and the visualization is accomplished using `IPython.display`. These raw results are then evaluated and compared against each other.

3.3 Evaluation Metrics

All models were evaluated using post-training results which are accessible in each model's results directory. All the results were measured using a constant evaluation metric. The following metrics were visualized:

1. Confusion Matrix
2. F1-Confidence Curve
3. Precision-Recall Curve
4. Precision-Confidence Curve
5. Recall-Confidence Curve

Additionally, overall model results were also visualized, including mean average precision (mAP), losses in validation, test and training datasets, precision and recall.

$$Precision = \frac{True\ Positive\ (TP)}{True\ Positive\ (TP) + False\ Positive\ (FP)} \qquad [\,1\,]$$

$$Recall = \frac{True\ Positive\ (TP)}{True\ Positive\ (TP) + False\ Negative\ (FN)} \qquad [\,2\,]$$

$$F1\ score = 2 \times \frac{Precision \times Recall}{Precision + Recall} \qquad [\,3\,]$$

$$mAP = \frac{\Sigma_{j+1}^{k} APi}{k} \qquad [4]$$

Precision helps evaluate the robustness of the object detached whereas recall indicates the model's ability to be able to detect instances of concern in the data input. The **F1 score** helps us measure both precision and recall in a balanced manner. Finally, **mAP** helps compare the growth truth bounding box and the bounding box detected by the model. The **Confusion matrix** summarizes the performance of an ML model on a set data set and includes specifics such as classes also known as the error matrix.

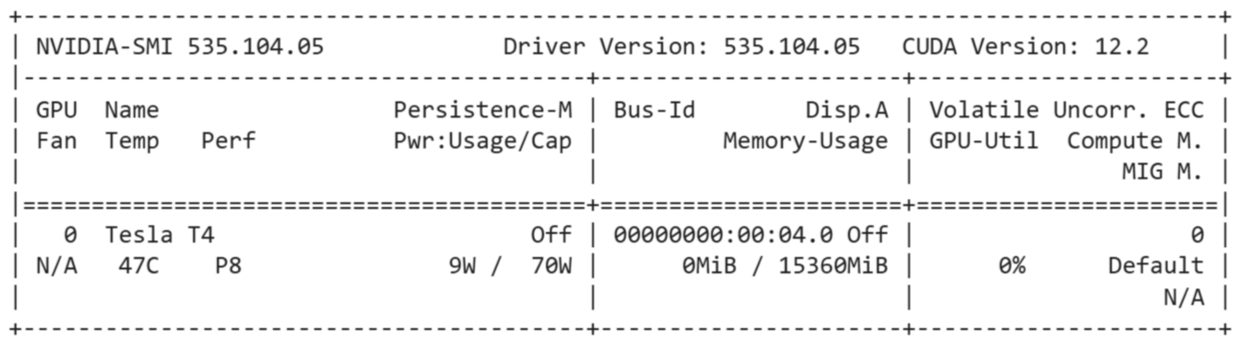
A **confusion matrix** is usually structured through the use of:

1. True Positive (TP): Number of instances where the model had correctly predicted the positive classes
2. False Positive (FP): Number of instances where the model had incorrectly predicted positive classes. (Predicted positive, but the real class was negative.)
3. True Negative (TN): The number of instances where the model had correctly predicted the negative classes.
4. False Negative (FN): The number of instances where the model had incorrectly predicted negative classes. (Predicted negative, but the real class value was positive.)

Higher TP and TN values showcase the model's performance as well subsequently higher FP values suggest the model is making multiple mistakes.

3.4 Model Training

To proceed with training the frameworks; Cloud GPU provided by Google Colaboratory was utilized due to GPU limitations. The following details specify the GPU used:



```
+---------------------------------------------------------------------------------------+
| NVIDIA-SMI 535.104.05             Driver Version: 535.104.05   CUDA Version: 12.2     |
|-----------------------------------------+----------------------+----------------------+
| GPU  Name                 Persistence-M | Bus-Id        Disp.A | Volatile Uncorr. ECC |
| Fan  Temp   Perf          Pwr:Usage/Cap |         Memory-Usage | GPU-Util  Compute M. |
|                                         |                      |               MIG M. |
|=========================================+======================+======================|
|   0  Tesla T4                       Off | 00000000:00:04.0 Off |                    0 |
| N/A   47C    P8               9W /  70W |      0MiB / 15360MiB |      0%      Default |
|                                         |                      |                  N/A |
+-----------------------------------------+----------------------+----------------------+
```

Figure 10 : details of GPU used for investigation

Each framework was run through constant parameters and visualised post-training.

3.4.1 YOLOv8

YOLOv8 was installed directly in the workspace using a pip function

```
!pip install ultralytics==8.0.196

from IPython import display
display.clear_output()

import ultralytics
ultralytics.checks()
```

Figure 11: Installation of YOLOv8

Subsequently, the dataset was imported to the workspace using RoboFlow. The dataset version used for this model was yolov8 and the training was conducted using the set parameters, as shown in the code snippet below.

```
!yolo task=detect mode=train model=yolov8s.pt
data=/content/datasets/O.D-IN-BAD-WEATHER-1/data.yaml epochs=25 imgsz=640
batch=16 plots=True
```

Figure 12: Data training on YOLOv8

Post the training, the model was visualised by outputting the training results.

3.4.2 YOLOv9

Despite the similarities with YOLOv8 while initializing the training process, due to the recent release of YOLOv9 it yet cannot be imported through a pip function, therefore the framework had to be cloned from its official GitHub repository.

```
!git clone https://github.com/SkalskiP/yolov9.git
%cd yolov9
!pip install -r requirements.txt -q
```

Figure 13: Installation of YOLOv9

Additionally, the YOLOv9 model at the current stage cannot download the weights directly, so they were manually downloaded from GitHub and stored separately in the workspace.

```
# YOLOv9 can not automatically download the weights so they are manually
downloaded
!wget -P {HOME}/weights -q
https://github.com/WongKinYiu/yolov9/releases/download/v0.1/yolov9-c.pt
!wget -P {HOME}/weights -q
https://github.com/WongKinYiu/yolov9/releases/download/v0.1/yolov9-e.pt
!wget -P {HOME}/weights -q
https://github.com/WongKinYiu/yolov9/releases/download/v0.1/gelan-c.pt
!wget -P {HOME}/weights -q
https://github.com/WongKinYiu/yolov9/releases/download/v0.1/gelan-e.pt
```

Figure 14: Installation of YOLOv9 weights

The parameters were kept constant for YOLOv8, and the training results were extracted and visualized post-training.

```
%cd {HOME}/yolov9

!python train.py \
--batch 16 --epochs 25 --img 640 --device 0 \
--data {dataset.location}/data.yaml \
--weights {HOME}/weights/gelan-c.pt \
--cfg models/detect/gelan-c.yaml \
--hyp hyp.scratch-high.yaml
```

Figure 15: Data training on YOLOv9

3.4.3 YOLOv10

Similar to YOLOv9, the recent release of YOLOv10 means it cannot be downloaded using a pip function and hence it was to be imported by cloning its GitHub repository.

```
!pip install -q git+https://github.com/THU-MIG/yolov10.git
```

Figure 16: Installation of YOLOv10

Following this, the weights were manually downloaded and stored in the workspace separately.

```
!wget -P {HOME}/weights -q
https://github.com/THU-MIG/yolov10/releases/download/v1.1/yolov10n.pt
!wget -P {HOME}/weights -q
https://github.com/THU-MIG/yolov10/releases/download/v1.1/yolov10s.pt
!wget -P {HOME}/weights -q
https://github.com/THU-MIG/yolov10/releases/download/v1.1/yolov10m.pt
!wget -P {HOME}/weights -q
https://github.com/THU-MIG/yolov10/releases/download/v1.1/yolov10b.pt
!wget -P {HOME}/weights -q
https://github.com/THU-MIG/yolov10/releases/download/v1.1/yolov10x.pt
!wget -P {HOME}/weights -q
https://github.com/THU-MIG/yolov10/releases/download/v1.1/yolov10l.pt
```

Figure 17: Installation of YOLOv10 weights

The dataset was downloaded in the same manner similar to YOLOv9 and YOLOv8. However, the model version remained yolov9 due to the unavailability of a yolo10 data type version. This change did not affect the performance. YOLOv10 was trained using constant parameters and visualized post-training results.

```
!yolo task=detect mode=train epochs=25 batch=16 imgsz=640 plots=True \
model={HOME}/weights/yolov10n.pt \
data=/content/datasets/O.D-IN-BAD-WEATHER-1/data.yaml
```

Figure 18: Data training on YOLOv10

3.5 Hypothesis

This study hypothesises that YOLOv10 will outperform the other models in precision, especially for well-defined object classes like cars and motors. NMS-free strategy and dual-head architecture employed by YOLO will aid this. However, YOLOv8 may have a higher recall because of its CSPDarknet backbone and finally YOLOv9 will exhibit higher precision in detecting small or occluded objects due to the introduction of PGI and GELAN.

4 Results and Analysis

4.1 Tabular Representation

	Precision			Recall			mAP50		
Class	YOLOv8	YOLOv9	YOLOv10	YOLOv8	YOLOv9	YOLOv10	YOLOv8	YOLOv9	YOLOv10
All	0.677	0.475	0.658	0.263	0.377	0.192	0.286	0.378	0.203
Bike	0.603	0.334	1	0.04	0.12	0	0.7	0.115	0
Bus	0.523	0.424	0.418	0.425	0.525	0.275	0.412	0.469	0.288
Car	0.704	0.705	0.603	0.709	0.749	0.631	0.741	0.777	0.655
Motor	1	0.575	1	0	0.145	0	0.02	0.209	0.7
Person	0.498	0.454	0.301	0.261	0.355	0.198	0.285	0.348	0.171
Rider	1	0.372	1	0	0.211	0	0.081	0.286	0
Truck	0.414	0.461	0.288	0.403	0.532	0.242	0.369	0.445	0.203

Table 3: Comparison of YOLOv8, YOLOv9 and YOLOv10's performances regarding Precision, Recall and mAP50

Precision: YOLOv8 provides the highest Precision overall (0.677), compared to YOLOv9 (0.475) and YOLOv10 (0.658). YOLOv9 has the lowest overall precision, suggesting that it may produce more false positives, lastly, YOLOv10 has slightly lower precision than YOLOv8 but higher than YOLOv9, indicating YOLOv10 holding a balance.

Recall: YOLOv8 employs the lowest Recall (0.263) hence it misses more objects in the input; This could be a significant drawback in addressing adverse weather conditions. YOLOv9 has a higher Recall (0.377) than YOLOv8 but still is behind YOLOv10, indicating a better ability to detect in challenging environments. YOLOv10 has the highest recall (0.658), making it able to detect most objects which is crucial in adverse weather.

mAP50: YOLOv8 shows a middle-group performance in mAP50 (0.286) but is however outperformed by YOLOv9 (0.378), indicating YOLOv9 has better accuracy. Lastly, YOLOv10 has the lowest mAP (0.203), indicating that while it detects multiple objects, the accuracy of these predictions is lower compared to YOLOv9 and YOLOv8.

4.2 Graphical Representation

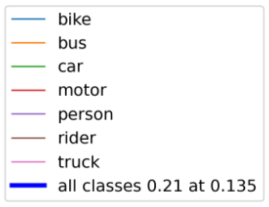
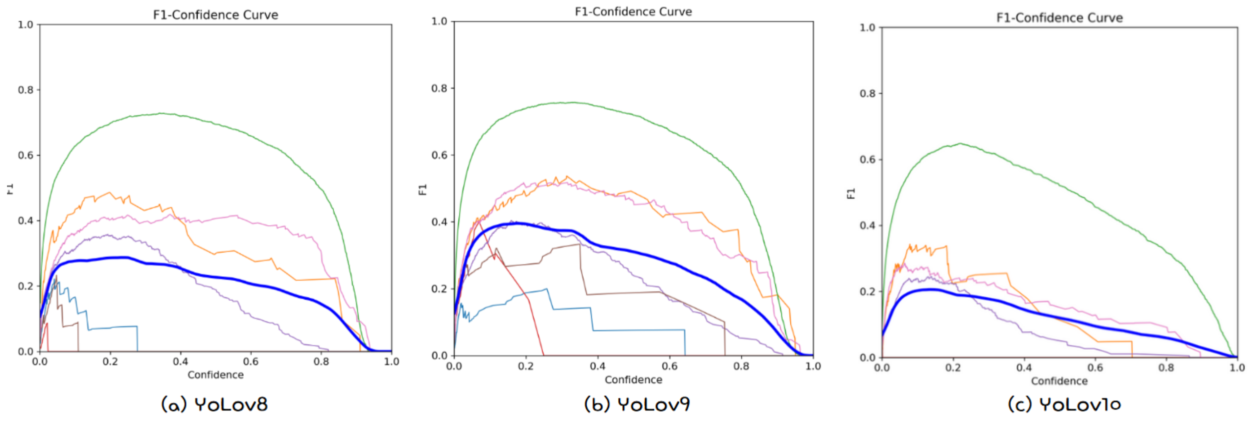


Figure 19: Colour coding for object classes

The image above showcases the colour assigned to each object class. The results graph each object class result on each evaluation metric.

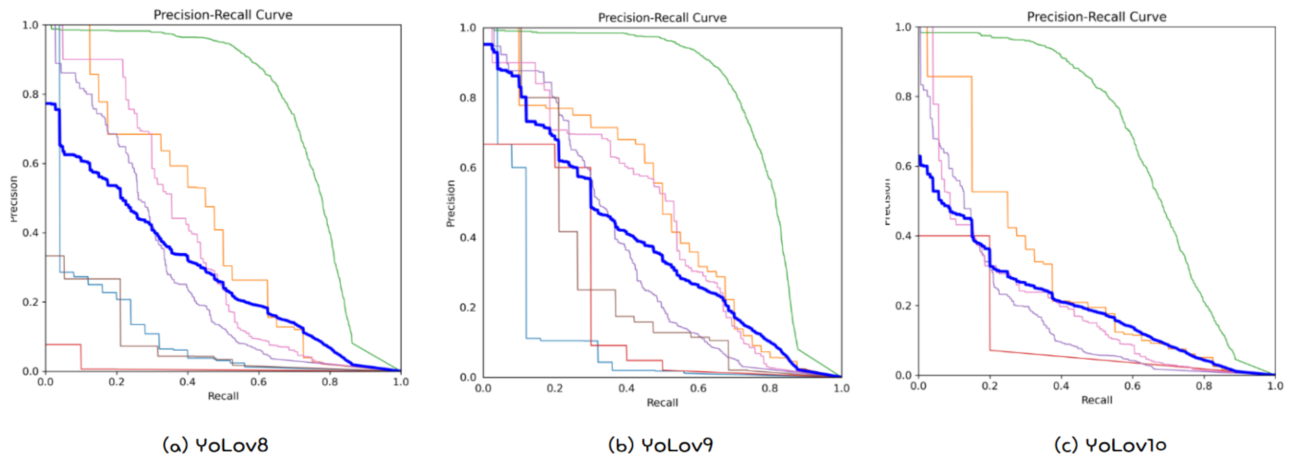
4.2.1 F1-Confidence



Graph 1: Graphical Result of YOLOv8, YOLOv9 and YOLOv10 on F1-Confidence Curve

YOLOv9 stands out to be the most balanced model across all the classes, with a higher and more stable F1 score over a range of confidence levels. YOLOv10's tabular results reflect its generally lower F1 scores. YOLOv8 lastly, provides a more moderate performance, with less consistent F1 scores which makes this framework more consistent than YOLOv10 and less favourable compared to YOLOv9.

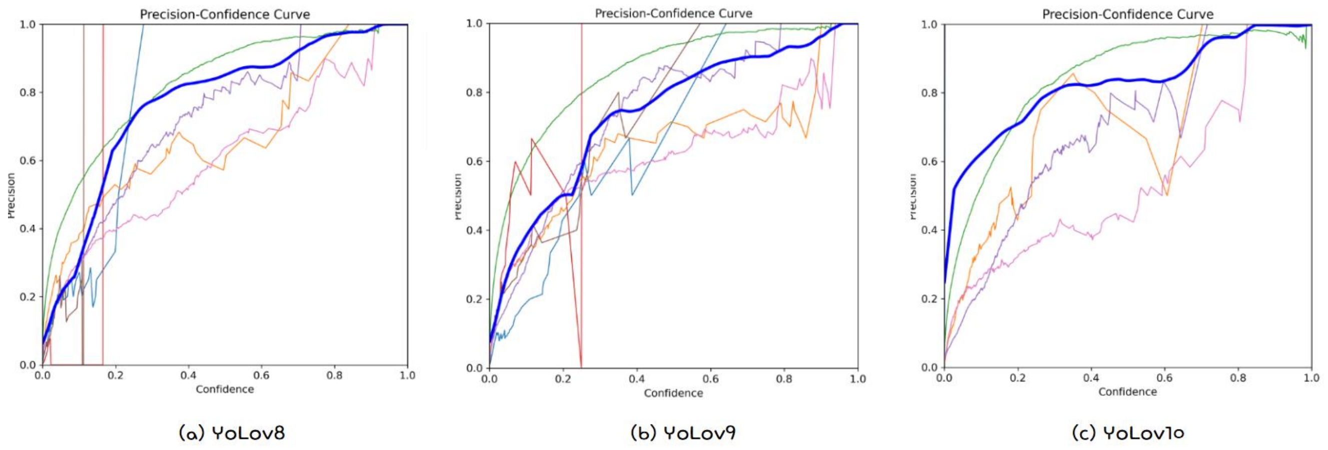
4.2.2 Precision on Recall



Graph 2: Graphical Result of YOLOv8, YOLOv9 and YOLOv10 on Precision/Recall Curve

YOLOv9 exhibits its balance by offering the best trade-off between precision and recall across most object classes consequently YOLOv10 shows strong precision for certain classes but tends to lose precision rapidly as recall increases, particularly for smaller or more complex objects. Lastly, YOLOv8 is less consistent than YOLOv9 with a more significant drop in precision as recall increases across several classes.

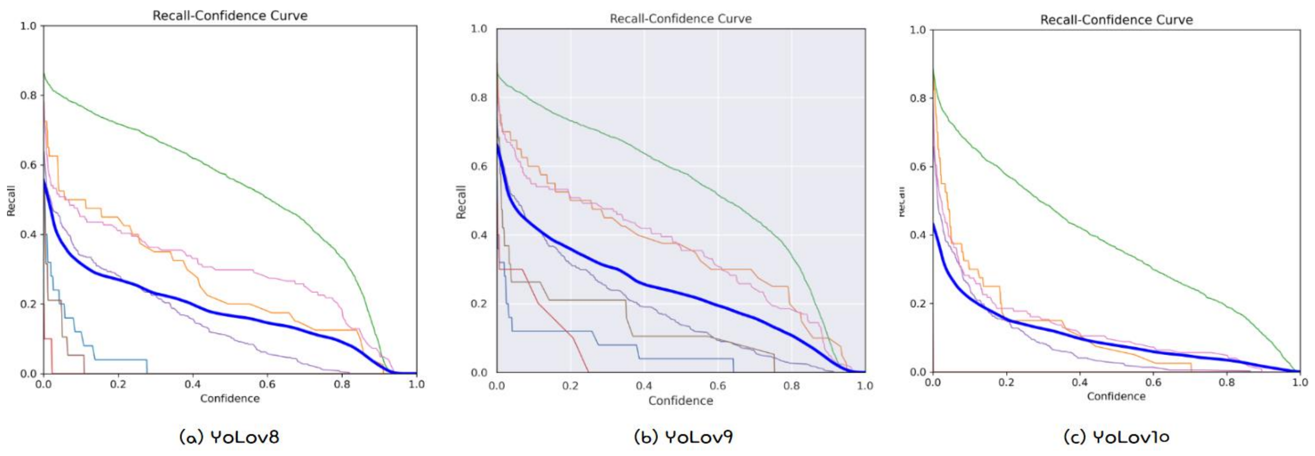
4.2.3 Precision on Confidence



Graph 3: Graphical Result of YOLOv8, YOLOv9 and YOLOv10 on Precision/Confidence Curve

YOLOv9 is presented as the most balanced model, by showcasing consistent and dependable accuracy thresholds for the majority of object classes. YOLOv10 achieves excellent precision but with greater variability between classes; overall, its performance may be less consistent and more class-dependent. Lastly, while YOLOv8 performs reasonably well, YOLOv9 often outperforms it, particularly when it comes to consistency across various object classes.

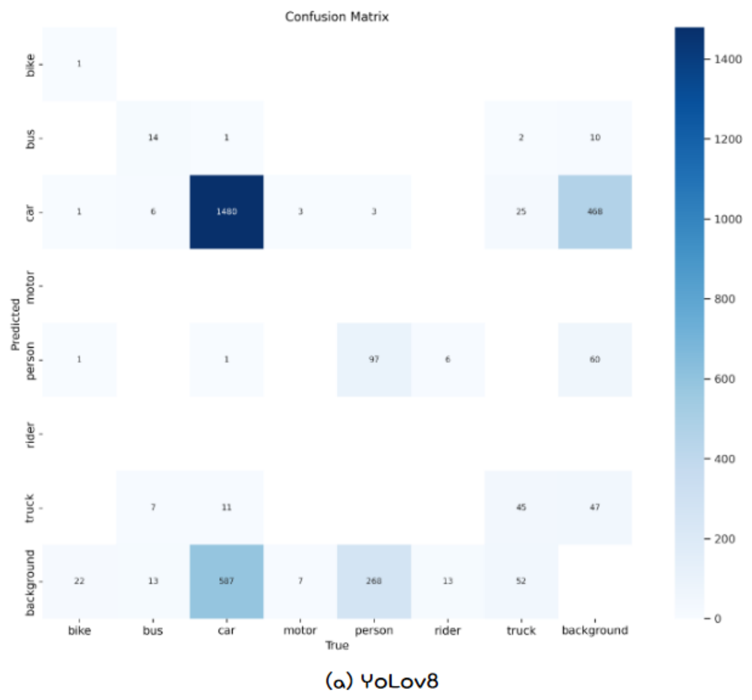
4.2.4 Recall on Confidence



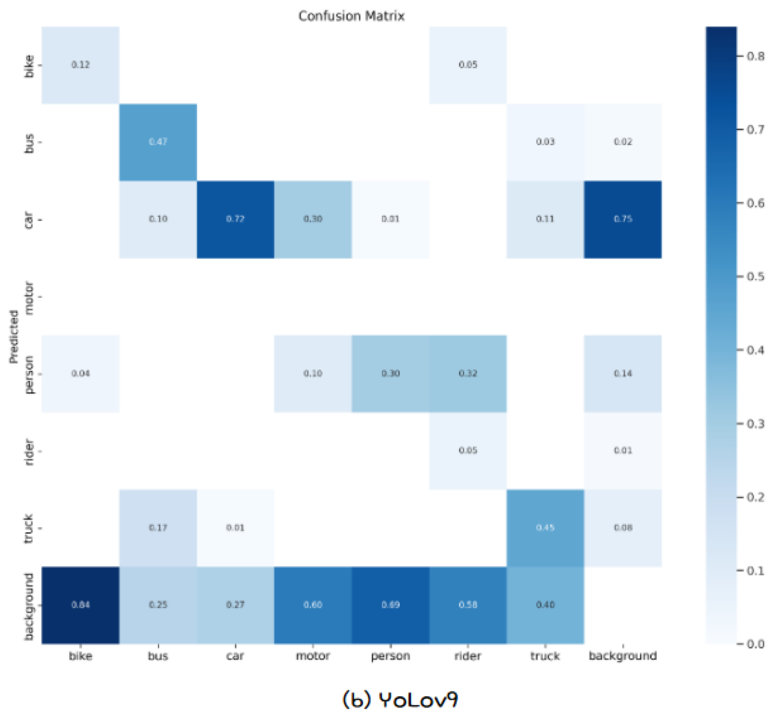
Graph 4: Graphical Result of YOLOv8, YOLOv9 and YOLOv10 on Recall/Confidence Curve

YOLOv9 once again is exhibited to be the most balanced, by maintaining higher recall across a range of confidence levels. YOLOv10 rapidly declines in recall as confidence increases, indicating a strong preference for high precision at the expense of recall. YOLOv8 generally struggles to maintain recall as the confidence is increased especially compared against YOLOv9

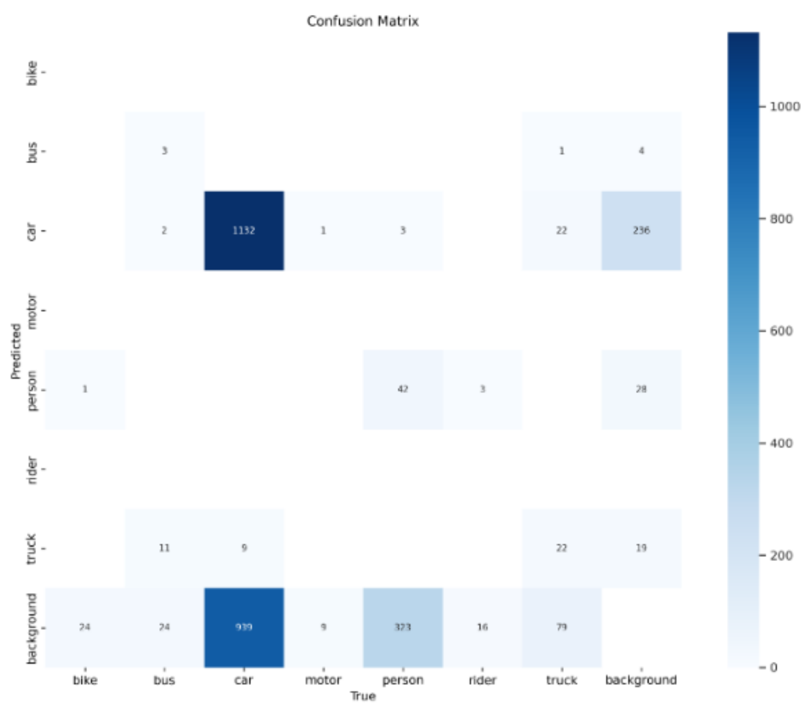
4.2.5 Confusion Matrix



Graph 5: Confusion Matrices of YOLOv8



Graph 6: Confusion Matrices of YOLOv8, YOLOv9 and YOLOv10



(c) YoLov1o

Graph 7: Confusion Matrices of YOLOv8, YOLOv9 and YOLOv10

YOLOv8 excels in predicting the class car but then showcases confusion with the background class and often misclassifies the background as car. YOLOv9 however improves this by offering a more balanced performance with normalized data regardless it does struggle with misclassification. Lastly, YOLOv10 outperforms the other models by maintaining overall accuracy and not consuming background with other classes, making it the most effective of the three of them.

4.3 Analysis

YOLOv8 has a high precision for classes such as Motor and Car and low precision for classes Bus and Truck this can be because of the CSPDarknet backbone as it is optimized for well-designed objects like cars and motors. Larger scale objects such as trucks could introduce challenges due to variability of size especially when distorted by adverse weather conditions.

Furthermore, reliance on convolutional layers, which are effective for high-contrast features may be the cause of low recall as high-contrast features are diminished by adverse conditions. Additionally, YOLOv8's anchor-free detection can make the model less reliable regardless of increasing the speed of detections.

The C2f model in YOLOv8 combines high-level features with context which can be effective in environments which are high in contrast and clear. However, given a dataset with distortions for this investigation, YOLOv8 is unable to maintain a balance in mAP50. It becomes low in classes where these distortions are more pronounced.

The use of anchor-free detection may have caused YOLOv8 to struggle with objects whose position and size vary unpredictably this directly contributes to the steep in the recall furthermore, the confusion matrix showcases misclassification between objects and the background, especially for Truck, this can be justified because of the use of CSPDarknet which is generally excellent for extracting complex features and hence there is a chance that noise and other elements could have confused the architecture to pick it up as well and in return why the model confuses objects such as trucks with the background.

YOLOv9 shows variability in precision-recall curves, and includes sharp declines in classes like Bike and Person– though YOLOv9 can maintain high precision it struggles with recall, especially in challenging detection scenarios. This is possible due to the introduction of PGI– which can affect gradient stability and make the model more sensitive to confidence thresholds.

YOLOv9 exhibits more fluctuation in the precision/confidence curve, especially at lower confidence levels– GELAN’s impact could have led to affect gesture aggregation, which may not always provide constant gesture quality across different classes, in return, this could lead to the precision and confidence variability. Similarly, the recall/confidence curve also showcases a noticeable sharp drop in YOLOv9 in recall as confidence increases– This suggests the model is cautious in its detections when operating under high confidence thresholds.

Achieving high true positive rates for Cars and trucks but also showing confusion in the background category, particularly in bike and motor can be justified due to the model’s sensitivity to background features whilst using GELAN– which aggregates features from different scales but doesn't effectively separate foreground objects from the background. Higher cross-entropy losses for these classes define the model’s struggle with class separability, particularly under the noise of adverse conditions.

YOLOv10 exhibits a conservative precision-recall trade-off. This is evident in the sharp decline of recall as precision increases– the NMS-free strategy employed can likely be the cause of this, which optimizes precision by eliminating overlapping detections but this costs the recall.

The weighting on precision may cause the model to prioritise certain high-confidence predictions which can lead to a low recall, especially for objects that are harder to detect–for example, objects smaller in size or covered by fog or adverse factors.

The dual head architecture could reason for the model's high precision at higher confidence and variability of precision at lower confidence– the model might over-prune detection from one-to-many heads during inference. Consequently, the recall/confidence curve showcases the decline of YOLOv10– this is because the model sacrifices recall to ensure higher precision. Decoupling of spatial and channel operations may also contribute to this making the model less sensitive to details which is necessary for detecting smaller objects.

The Confusion matrix suggests that YOLOv10 shows significant confusion in the background particularly with smaller classes such as Person and Motor. YOLOv10 reduces post-processing through NMS-free strategies this can contribute to filtering out lower confidence detection that would otherwise be true positives. The higher rates of false positives specifically in the background can be justified by the confidence-weighted prediction strategy where the model focuses on precision during inference and could lead to overconfidence in background misclassifications.

5 Conclusion

From this investigation, it was interpreted that in the evolution of frameworks from YOLOv8 to YOLOv10, there is a trend towards refining precision and recall. YOLOv9 serves substantial improvements compared to YOLOv8 making it robust for a wide range of OD tasks, especially in challenging environments. Technically, even though YOLOv10 is more advanced, it does not justify its performance. Given the evidence from the results, generally, YOLOv9 is suggested for OD in bad weather due to its balance and reliability compared to other models used in this investigation. However, in conclusion after the result analysis, the best model does not exist and can only be generalised. To gain maximum benefit, it is recommended to consider the performance of these frameworks to be case-dependent. YOLOv10 can be considered for specialized scenarios such as situations where the need for high precision is required, high confidence environments and specialised OD, similarly, YOLOv8 can be considered for lightweight applications and environments with low complexity.

6 Further Scope

According to this investigation; resulted in a sequence of best-performing to least-performing frameworks. There is scope to implement multiple framework models in a singular workspace aimed at one task and leverage the importance of each model's output to be case-dependent, similar to how weights work in a neural network– this approach could help the rescue robot not be dependent on a singular framework rather multiple working frameworks which are enhanced in a particular field such as precision.

7 Limitations and Challenges

The investigation would have been carried out to be more sophisticated perhaps if the dataset was not too small. It is hard to generalise and train a model using a small dataset. The BDDK100 dataset was not used to due poor computational power and hardware limitations. Furthermore, the usage of other modes of data types such as videos and other multimodal options would have further enhanced the integrity of this research but it was not proceeded with due to yet again limitations of resources, hence the investigation improvised to providing investigation upon a smaller data set with only still-images which gives rise for further studies in the future.

It took plentiful hours to find the dataset which matched this investigation's aim in addition to more time spent playing around with Google Colaboratory and the framework implementation and numerous inferences and model training to be done before this investigation.

The lack of proper documentation which YOLOv10 held made it immensely hard to be able to understand the architecture of the model. However, through the use of official documents, the GitHub repository and a few research journals this was made possible. Highlighting the extensive time taken to evaluate each research before referencing through their acceptance in terms of peer review and citations.

8 Bibliography

Flynn, Helen, et al. "Machine Learning Applied to Object Recognition in Robot Search and Rescue Systems." *ResearchGate*, 20, https://www.researchgate.net/profile/Helen-Flynn/publication/224773218_Machine_Learning_Applied_to_Object_Recognition_in_Robot_Search_and_Rescue_Systems/links/5440edd50cf2ebb036905a88/Machine-Learning-Applied-to-Object-Recognition-in-Robot-Search-and-Rescue-Systems.pdf. Accessed 1 April. 2024.

"Computer Vision." *IBM*, https://www.ibm.com/topics/computer-vision. Accessed 4 April. 2024.

Li, H., and L. S. Luo. "A Critical Review of Multiphase Flow Simulation Using the Lattice Boltzmann Method." *Archives of Computational Methods in Engineering*, 2019 https://link.springer.com/article/.07/s11831-018-09312-w. Accessed 1 June. 2024.

"Machine Learning Frameworks." *GeeksforGeeks*, 23 May 2023, https://www.geeksforgeeks.org/machine-learning-frameworks/. Accessed 3 May 2024.

"Fukushima Daiichi Accident." *World Nuclear Association*, June 2023, https://world-nuclear.org/information-library/safety-and-security/safety-of-plants/fukushima-daiichi-accident. Accessed 2 June. 2024.

Westcott, Lucy. "Robots Sent into Fukushima Have 'Died.'" *Newsweek*, Mar. 2016, https://www.newsweek.com/robots-sent-fukushima-have-died-435332. Accessed 5 June. 2024.

Li, Qiang, et al. "An Overview of the Cuckoo Search Algorithm and Its Applications." *Proceedings of the IEEE*, 2011, https://ieeexplore.ieee.org/document/5876227. Accessed 5 June. 2024.

Toma, David, et al. "Comprehensive Survey of Machine Learning-Based Prediction Models for the Solar Energy System and Evaluations of Solar Irradiance." *Neurocomputing*, 2022 https://www.sciencedirect.com/science/article/pii/S0924271622003367. Accessed 5 June. 2024.

Wang, Cheng, et al. "A Review of Machine Learning Applications in Wireless Networks: Algorithms, Datasets, and Results." *arXiv*, 19 Jul. 2019, https://arxiv.org/abs/1907.09408. Accessed 5 May 2024.

LeCun, Yann, et al. "Gradient-Based Learning Applied to Document Recognition." *Proceedings of the IEEE*, ResearchGate, https://www.researchgate.net/publication/2985446_Gradient-Based_Learning_Applied_to_Document_Recognition. Accessed 25 June 2024.

LeCun, Yann, Yoshua Bengio, and Geoffrey Hinton. "Deep Learning." *Nature*, 2015, https://www.cs.toronto.edu/~hinton/absps/NatureDeepReview.pdf. Accessed 2 April 2024.

Jain, Avijit. *MIT Deep Learning Book PDF*. GitHub, https://github.com/janishar/mit-deep-learning-book-pdf. Accessed 25 June 2024.

"References for Papers." *Scientific Research Publishing*, https://www.scirp.org/reference/ReferencesPapers?ReferenceID=1308330. Accessed 25 June 2024.

Chollet, François. "Xception: Deep Learning with Depthwise Separable Convolutions." *IEEE Xplore*, 2017, https://ieeexplore.ieee.org/document/7780459. Accessed 7 January 2024.

Tian, Zhi, et al. "FCOS: Fully Convolutional One-Stage Object Detection." *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, 2019, https://openaccess.thecvf.com/content_ICCV_2019/html/Tian_FCOS_Fully_Convolutional_One-Stage_Object_Detection_ICCV_2019_paper.html. Accessed 15 July 2024.

Redmon, Joseph, and Ali Farhadi. "YOLO9000: Better, Faster, Stronger." *arXiv*, 2017, https://ar5iv.labs.arxiv.org/html/1704.04861. Accessed 15 July 2024.

Odena, Augustus, et al. "Deconvolution and Checkerboard Artifacts." *Distill*, 2016, https://distill.pub/2016/deconv-checkerboard/. Accessed 15 July 2024.

Wang, Chien-Yao, et al. "CSPNet: A New Backbone that Can Enhance Learning Capability of CNN." *NYCU Scholar*, https://scholar.nycu.edu.tw/en/publications/cspnet-a-new-backbone-that-can-enhance-learning-capability-of-cnn. Accessed 15 July 2024

Hirschfeld, Gerrit, and Ulf Ziemann. "Neuroplasticity and Functional Recovery After Stroke: Evidence from TMS and fMRI." *PubMed*, 2015, https://pubmed.ncbi.nlm.nih.gov/26353135/. Accessed 15 July 2024.

He, Tong, et al. "Bag of Tricks for Image Classification with Convolutional Neural Networks." arXiv, 2017, https://arxiv.org/abs/1711.06897/. Accessed 15 July 2024.

Zhou, Zheng, et al. "Multi-Scale Feature Attention for Person Re-Identification." *Lecture Notes in Computer Science (LNCS)*, vol. 12335, Springer, 2020 https://link.springer.com/chapter/.07/978-3-030-58452-8_13. Accessed 15 July 2024.

Szegedy, Christian, et al. "Going Deeper with Convolutions." *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2015, https://www.cv-foundation.org/openaccess/content_cvpr_2015/html/Szegedy_Going_Deeper_With_2015_CVPR_paper.html. Accessed 15 July 2024.

Sanderson, Grant. "Neural Networks." *BlueBrown*, https://www.3blue1brown.com/lessons/neural-networks. Accessed 7 January 2024.

Redmon, Joseph, et al. "You Only Look Once: Unified, Real-Time Object Detection." *arXiv*, 8 Jun. 2015, https://arxiv.org/abs/1506.02640. Accessed 7 June 2024.

Chen, Zhenyu, et al. "An Extensive Survey on Vision Transformers: Transforming Vision for Better Future." *arXiv*, 5 Jul. 2024, https://arxiv.org/html/2407.02988v1#S3. Accessed 7 June 2024.

Kingma, Diederik P., and Jimmy Ba. "Adam: A Method for Stochastic Optimization." *arXiv*, 3 Mar. 2015, https://arxiv.org/abs/1503.02406. Accessed 7 June 2024.

Wang, Alex, et al. "GLUE: A Multi-Task Benchmark and Analysis Platform for Natural Language Understanding." *arXiv*, 22 Oct. 2020, https://arxiv.org/abs/20.11929. Accessed 8 June 2024.

Nijkamp, Erik, et al. "A Survey on Visual Prompt Learning." *arXiv*, 23 Dec. 2022, https://arxiv.org/abs/2212.11696. Accessed 8 June 2024.

Jiang, Zhengkai, et al. "Text2Human: Text-Driven Controllable Human Image Generation." *arXiv*, 9 Nov. 2022, https://arxiv.org/abs/2211.04800. Accessed 7 June 2024.

Zhou, Xiangning, et al. "OpenXGPT: An Open-Source Instruction-Following Language Model." *arXiv*, 27 Feb. 2024, https://arxiv.org/pdf/2402.13616. Accessed 7 June 2024.

Ramesh, Aditya, et al. "Zero-Shot Text-to-Image Generation." *arXiv*, May 2021, https://arxiv.org/abs/25.04206. Accessed 7 June 2024.

Jiang, Zhengkai, et al. "Text2Human: Text-Driven Controllable Human Image Generation." *arXiv*, 9 Nov. 2022, https://arxiv.org/abs/2211.04800. Accessed 7 June 2024.

Srivastava, Rupesh Kumar, et al. "Highway Networks." *arXiv*, 15 Jun. 2015, https://arxiv.org/abs/1506.04878. Accessed 8 June 2024.

Raffel, Colin, et al. "Exploring the Limits of Transfer Learning with a Unified Text-to-Text Transformer." *arXiv*, 26 May 2020, https://arxiv.org/abs/2005.12872. Accessed 8 June 2024.

Viana, André Lucas. "Hungarian-Loss: A Python Library for the Hungarian Algorithm." *PyPI*, https://pypi.org/project/hungarian-loss/. Accessed 9 June. 2024.

Vaswani, Ashish, et al. "Scaling Vision Transformers." *arXiv*, 17 Aug. 2021, https://arxiv.org/abs/28.07755. Accessed 9 June. 2024.

"YOLOv10 Documentation." *Ultralytics*, https://docs.ultralytics.com/models/yolov10/. Accessed 15 June 2024.

Xu, Ming, et al. "Emerging Trends in Large Language Models: A Survey." *arXiv*, 25 May 2024, https://arxiv.org/pdf/2405.14458. Accessed 15 June . 2024.

"YOLOv:10 An Introduction to the Latest Version of YOLO." *LearnOpenCV*, https://learnopencv.com/yolov10/. Accessed 15 June . 2024.

"YOLOv10: Everything You Need to Know." *Roboflow Blog*, 25 July 2023, https://blog.roboflow.com/what-is-yolov10/. Accessed 15 June 2024.

"Object Detection in Bad Weather Dataset." *Roboflow Universe*, https://universe.roboflow.com/foreignobjectaerodromes/o.d-in-bad-weather/dataset/1. Accessed 15 June 2024.

"Foreign Object Aerodromes." *Roboflow Universe*, https://universe.roboflow.com/foreignobjectaerodromes. Accessed 15 June. 2024.

Hinton, Geoffrey E., et al. "Improving Neural Networks by Preventing Co-adaptation of Feature Detectors." *arXiv*, 24 Jun. 2012, https://arxiv.org/abs/1206.5538. Accessed 19 June 2024.

Karn, Ujjwal. "Ujjwal Karn's Blog." *Ujjwalkarn.me*, https://ujjwalkarn.me/. Accessed 19 June 2024.

Liu, Zhuang, et al. "Swin Transformer: Hierarchical Vision Transformer Using Shifted Windows." *arXiv*, 8 Jul. 2021, https://arxiv.org/abs/27.04191. Accessed 19 June 2024.

Devlin, Jacob, et al. "BERT: Pre-Training of Deep Bidirectional Transformers for Language Understanding." *arXiv*, 8 Sep. 2018, https://arxiv.org/abs/1809.03193. Accessed 19 June 2024.

Simonyan, Karen, and Andrew Zisserman. "Very Deep Convolutional Networks for Large-Scale Image Recognition." *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, IEEE, 2015 https://ieeexplore.ieee.org/document/7005506. Accessed 19 June 2024.

Touvron, Hugo, et al. "Training Data-Efficient Image Transformers and Distillation Through Attention." *arXiv*, 23 Apr. 2020, https://arxiv.org/abs/2004.934. Accessed 19 June 2024.

Ahmed, Sameer, et al. "A Review on YOLOv8 and Its Advancements." *ResearchGate*, 2024, https://www.researchgate.net/publication/377216968_A_Review_on_YOLOv8_and_Its_Advancements. Accessed 25 June 2024.

9 Appendices

9.1 Appendix A: Terminology

Terminology	Definition
Multimodal	Integration of multiple data types.
Grid Matrix	Division of an image into separate grid cells.
Bounding boxes	Boxes which are used to localize and locate objects in an image are defined by coordinates.
Deep Neural Networks	Neural network with multiple hidden layers between input and output.
Computation	Process of performing calculations through technology
Information retainment	The ability of a NN to retain information to utilize later
Cloud-based data storage	Storing data on remote servers which are accessed via the internet
API keys	Identifiers which are used to authenticate access to a program

Table 1: Additional basic terminology

9.2 Appendix B: Background

9.2.1 Artificial Neural Networks (NNs)

A neuron contains data and each neuron contains different activation values in essence the activation value decides upon the significance of the neuron *(Sanderson)*. The more significant a neuron it is likely to increase significance of the neuron subsequently to the neuron forward, the less significant the neuron is cancelled out. Assigning weights to neuron connections between layers helps classify and manually tweak the significance of each neuron to be specific about results *(Sanderson)*.

$$w_1a_1 + w_2a_2 + w_3a_3 + \ldots + w_na_n \qquad [1]$$

The above equation describes the addition of weight *(Sanderson)*. The weighted sum would give the specific feature which is looked up for. Each neuron consists of biases which is a scalar value added to the weighted sum before passing the result through an activation function which is used to adjust the output of each neuron along with the weighted inputs.

9.2.2 Convolutional Neural Networks (CNNs)

Computer vision involves the computer interpreting stimuli from its environment-- to achieve this a CNN is used. CNN helps interpret visual data on a sophisticated basis. There are multiple ways to interpret data such as; Object recognition, detection or segmentation.

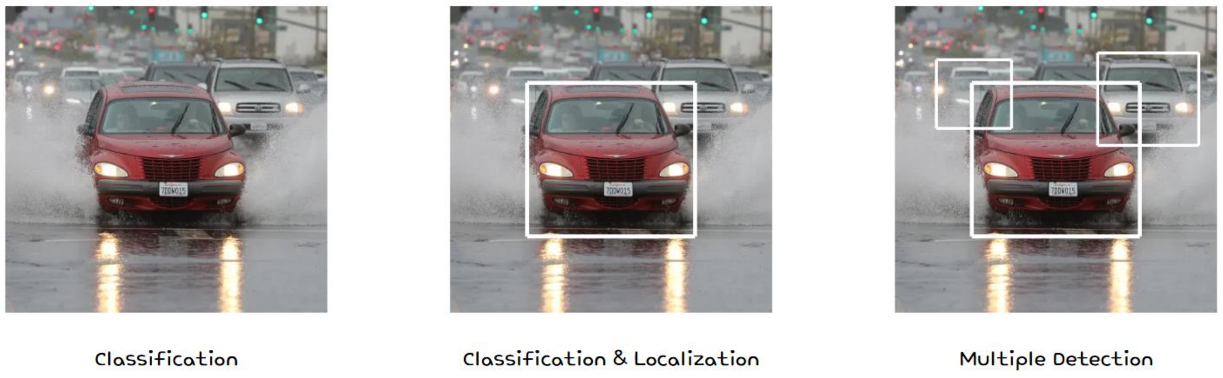


Figure 1: Classification, Localization and Segmentation Visualization

CNN is implemented through the use of " blocks " of convolution, it is essential to define Kernels/filters as wanted to be able to feature extract effectively *(Hinton et al.)*. Kernels and Filters describe small matrices used for convolutional operations. They help detect specific features in input data.

$$q_i^l = f\left(b_i^l + \sum_{j=0}^{d-1} w_{ij} \times x_j + j\right) \quad [1]$$

$$q_{ij}^l = f\left(b_{ij} + \sum_{k=0}^{d_1} \sum_{m=0}^{d_2} w_{(i+k)(j+m)}^{x_{(i+k)(j+m)}}\right) \quad [2]$$

Equation [1] The equation calculates the output q_{ij}^l of a neuron by applying intended activation f to the weighted sum of its inputs plus the bias, with an additional term j included. q_{ij}^l represents the output of a neuron i in layer l. $b_{i_j}^{\ l}$ represents the bias which has been added to the weighted sum of inputs. Equation [2] describes a neuron in a CNN where the output is calculated by applying the activation function to the sum of weighted inputs like [1] but over a local receptive field. The double summation reflects the process of convolving a filter across the input space to compute the neuron's output.

The output undergoes pooling to aggregate an emphasis on key features, achieved by reducing the spatial dimensions. Various pooling techniques are available such as average pooling, sum pooling, and max pooling *(Karn)*.

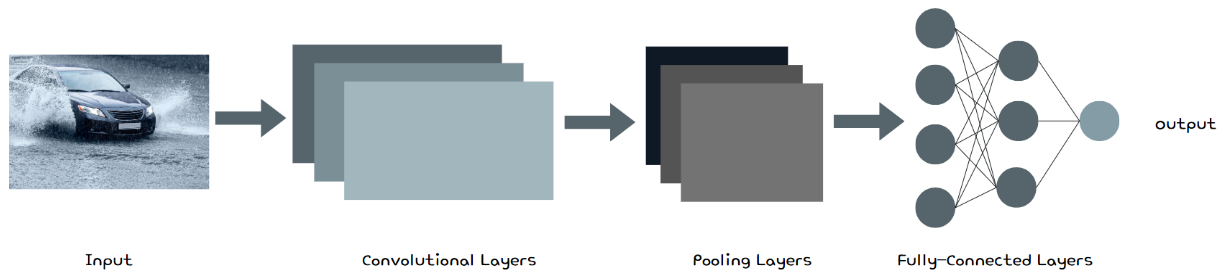


Figure 2: General structure of a CNN

(Liu et al.) Single-stage detectors identify objects in a single pass and hence remove the necessity for a separate region proposal step (like R-CNN). By employing multiple convolutional feature maps and varying scales for bounding box predictions, these detectors can effectively identify objects of different sizes and shapes in a single forward pass. Examples of single-shot detectors include the YOLO framework *(Devlin et al.)*

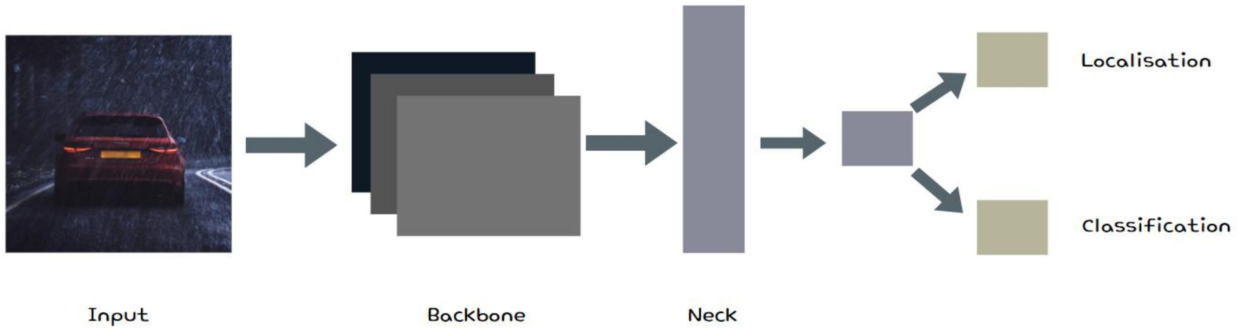
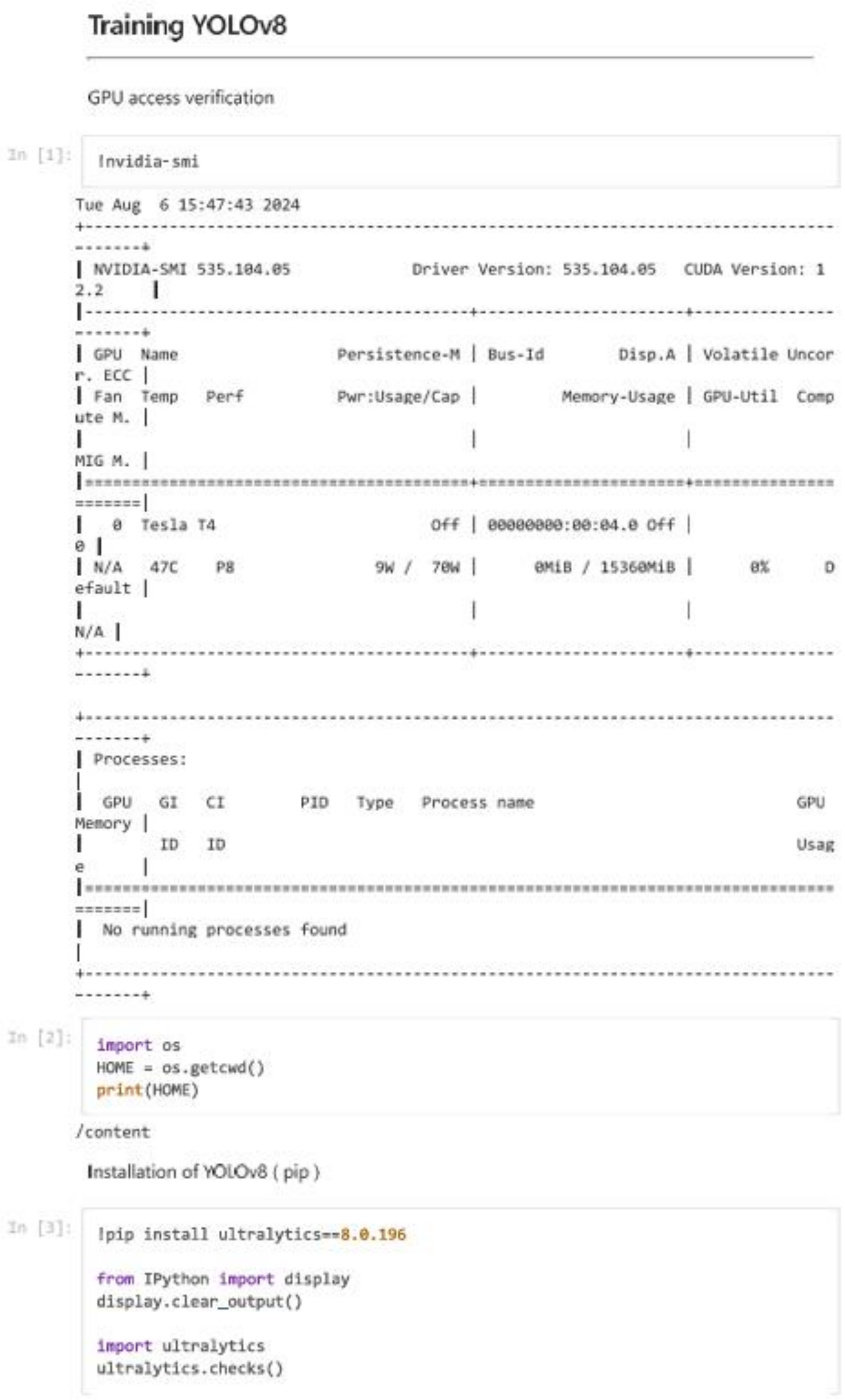


Figure 3: Abstract architecture of a single-stage object detector

9.3 Appendix D: Investigation Code



Training YOLOv8

GPU access verification

```
In [1]:  !nvidia-smi
```

```
Tue Aug  6 15:47:43 2024
+-----------------------------------------------------------------------------
-------+
| NVIDIA-SMI 535.104.05             Driver Version: 535.104.05   CUDA Version: 1
2.2     |
|-----------------------------------------+----------------------+--------------
-------+
| GPU  Name                 Persistence-M | Bus-Id        Disp.A | Volatile Uncor
r. ECC |
| Fan  Temp   Perf          Pwr:Usage/Cap |         Memory-Usage | GPU-Util  Comp
ute M. |
|                                         |                      |
MIG M. |
|=========================================+======================+==============
=======|
|   0  Tesla T4                       Off | 00000000:00:04.0 Off |
0 |
| N/A   47C    P8               9W /  70W |      0MiB / 15360MiB |      0%      D
efault |
|                                         |                      |
N/A |
+-----------------------------------------+----------------------+--------------
-------+

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

```
In [2]:  import os
         HOME = os.getcwd()
         print(HOME)
```

```
/content
```

Installation of YOLOv8 (pip)

```
In [3]:  !pip install ultralytics==8.0.196

         from IPython import display
         display.clear_output()

         import ultralytics
         ultralytics.checks()
```