Detection and classification of protected species bycatch in Swedish small-scale fisheries

Object Detection and Classification using Applied Machine Learning in a Federated Framework

Master's thesis in Data Science and Artificial Intelligence

Feroz Basheer, Muhammad Abdullah

CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden 2022
ferozb@student.chalmers.se, muhabdu@student.chalmers.se

Master's thesis 2022

Detection and classification of protected species bycatch in Swedish small-scale fisheries

Object Detection and Classification using Applied Machine Learning in a Federated Framework

Feroz Basheer
Muhammad Abdullah

CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Computer Science and Engineering
Division of Data Science and Artificial Intelligence
Chalmers University of Technology
Gothenburg, Sweden 2022

Detection and classification of protected species bycatch in Swedish small-scale fisheries
Object Detection and Classification using Applied Machine Learning in a Federated Framework
Feroz Basheer
Muhammad Abdullah

Supervisor: Erik Svensson, AI Sweden
Supervisor: Sara Königson, SLU
Supervisor: Lachlan Fetterplace, SLU
Examiner: Marina Axelson-Fisk, Chalmers University, Department of Mathematical Sciences

Master's Thesis 2022
Department of Computer Science and Engineering
Division of Data Science and Artificial Intelligence
Chalmers University of Technology
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Cover: An example of a bycatch, a porpoise caught in a gillnet

Typeset in LaTeX
Printed by Chalmers Reproservice
Gothenburg, Sweden 2022

Detection and classification of protected species bycatch in Swedish small-scale fisheries
Object Detection and Classification using Applied Machine Learning in a Federated Framework
Feroz Basheer
Muhammad Abdullah
Department of Computer Science and Engineering
Division of Data Science and Artificial Intelligence
Chalmers University of Technology

Abstract

Bycatches are an adverse side effect of fishing. Though rare, their occurrences have a serious impact on PETS(protected, endangered, and threatened species). In this paper, the feasibility of machine learning in the video feed analysis process to aid bycatch research is conducted and it is found that this niche benefits from applied machine learning. Different object detection algorithms are implemented on bycatch datasets built from scratch. The object detection models are compared on metrics such as average precision, mean average precision and recall to pick a model that is best suited for the bycatch dataset. It is also discussed how the machine learning model could benefit from diversifying the dataset while addressing key concerns of sharing data between different stakeholders. This concern is addressed by the adaptation of federated learning. A hierarchical federated machine learning framework (FEDn from Scaleout) is implemented to train YOLOv5s with Swedish and Danish clients(local models). The results obtained show that although the clients learn from each other, the rate of convergence is far slower than the locally trained models therefore requires fine-tuning and needs rethinking of global weights aggregation that determines how the clients learn from each other. Finally, it is concluded that with a good quality dataset the object detection model can be used as an aid for researchers, potentially helping them identify bycatch even when a human fails to identify them.

Keywords: Federated Learning, Object Detection, Bycatch, YOLO, PETS

Acknowledgements

We would like to thank our supervisors Erik Svensson, Lachlan Fetterplace and Sara Königson for giving us an opportunity and great support throughout the project. We would like to thank AI Sweden for providing us with a collaborative work space and access to the computing hardware for the project. We would also like to thank SLU for inviting us to give a talk on applied machine learning, the researchers and staff at SLU who helped us annotate our dataset. Lastly we would like to thank Gildas Glemarec and Lotte Kindt-Larsen at DTU for letting us setup a machine at DTU and providing us with an opportunity to train a model on DTUs dataset.

Gothenburg, June 2022
Muhammad Abdullah
Feroz Basheer

List of Acronyms

Below is the list of acronyms that have been used throughout this thesis listed in alphabetical order:

AP	Average Precision
CSV	Comma-separated values
CNN	Convolutional Neural Networks
DTU	Technical University of Denmark
EM	Electronic Monitoring
FP	False Positives
Fps	Frames Per Second
FN	False Negatives
GPS	Global Positioning System
GB	Giga Bytes
GDPR	General Data Protection Regulation
GPU	Graphics processing unit
gRPC	gRPC remote procedure call
IOT	Internet of Things
IoU	Intersection Over Union
mAP	Mean Average Precision
MB	Mega Bytes
NMS	Non-Maximal suppression
OS	Operating System
REM	Remote Electronic Monitoring
RCNN	Region-based Convolutional Neural Network
REST	Representational State Transfer
SLU	Swedish University of Agricultural Sciences
TP	True Positives
TXT	Text
TB	Tera Bytes
VOC	Voice of Customer
VGG	Visual Geometry Group
XML	Extensible Markup Language
YOLO	You Only Look Once
PETS	Protected,endangered and threatened species
ML	Machine Learning

Contents

List of Figures

List of Tables

1
Introduction

Small-scale fishing boats make up the majority of the global fishing fleet and are an important source of food and employment in coastal communities [1]. Unintentional capture or "bycatch" of protected, endangered, and threatened species (PETS) in small-scale fisheries occurs sporadically but can negatively impact PETS populations [2]. Many of these species have relatively small population sizes and low fecundity, so even small numbers of individuals being bycaught can have detrimental effects at a population level [3]. Monitoring of PETS bycatch provides important information on mortality rates, periods or locations of high bycatch risk and helps inform management strategies for bycatch reduction[4]. To get reliable data on bycatches, monitoring fisheries do require human observers or equivalent to physically being onboard fishing boats and counting bycatch, or commercial fishers to self-report; both can have inherent drawbacks e.g. cost, safety concerns, limited coverage, and reliability.

Remote electronic monitoring (REM) of bycatch using camera-based systems is an alternative or complimentary monitoring method that is becoming more common [5]. REM is a system of video cameras and sensors (e.g GPS) used to record fishing activities on fishing vessels, providing among other things, such as information on catch composition, bycatch rates, and the location of where the catches occurred while also generating multiple video feeds. Benefits include relatively low costs compared to having observers onboard, reliability, ability to monitor continuously, and safety.

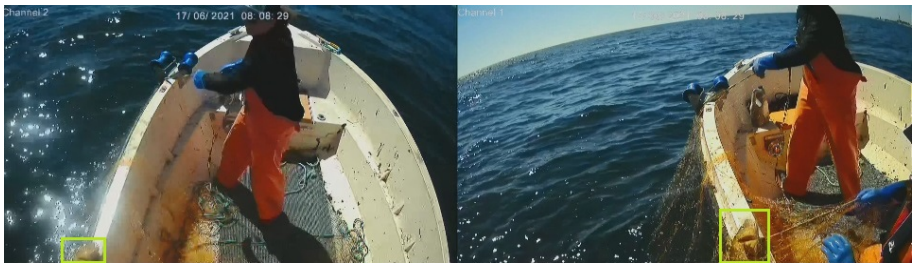


Figure 1.1: An example frame from the SLU dataset showing gillnets and a fish with bounding boxes from two different angles using REM

REM has been used in Sweden and Denmark on a limited number of small-scale boats to successfully monitor PETS. However, a drawback with REM both in Sweden and on a global scale is the time taken to analyse video footage. This is particularly problematic when monitoring PETS, as very large amounts of video data,

are needed to get reliable data. Currently, analysts must manually sort through all collected footage to count and identify catch and bycatch events. This entails considerable costs and increases the chance of human error in counts as the researchers have to sit too long and can lose concentration.

Machine learning techniques such as object detection potentially provide a means to speed up the analysis of REM video data. In this thesis, we develop and test a system to identify bycatch events, in REM dual camera footage, collected on Swedish and Danish small-scale fishing boats. The aim is to identify bycatch events using ML so that analysts only need to look at sections of video data where a catch occurs. This is extra challenging as the bycatch species are very rare and there is less training data, also the videos are private so there are some privacy concerns as well.

1.1 Related Work

Quantification of PETS bycatch is becoming a key aspect of protected marine species conservation and management [6]. A lot of work is being done in the field of monitoring protected species mostly with the help of REM. In the paper "Assessing Seabird Bycatch in Gillnet Fisheries Using Electronic Monitoring"[7] G. Glemarec, et al.(2020) state that Electronic monitoring with CCTV is a reliable solution for monitoring seabirds bycatch. As some vessels are too small to have an onboard observer, video monitoring the fishes helps in identifying bycatches at the species level as well as the sex of some species. An in-depth analysis of the data helped in the estimation of calculating the number of bird causalities at the fleet level.
In another paper "Observing Incidental Harbour Porpoise bycatch by Remote Electronic Monitoring"[6] Lotte Kindt-Larsen et al(2021) focus on the porpoise bycatch using REM. The paper states an important advantage of the REM system is that vessels that are too small to accommodate an observer also have data thanks to the REM.
Next to look at some related work where machine learning was involved, a competition N+1, N+2 challenge [8] in the US, in which participants build ML models to count, measure, and identify fish species on an on-board conveyor belt. Some of the ML techniques used were a U-Net [9] to find the position and orientation of the ruler to measure the size of the fish and an SSD network [10] to detect the fishes.
A research paper published in the Canadian Journal of Fisheries and Aquatic Sciences "Early lessons in deploying cameras and artificial intelligence technology for fisheries catch monitoring: where machine learning meets commercial fishing" [11] uses Electronic monitoring(EM) to monitor catch and bycatch events in wild capture fisheries using Computer Vision, machine learning, and an artificial intelligence-based system. Next the research paper "Machine learning to detect bycatch risk: Novel application to echo-sounder buoys data in tuna purse seine fisheries" [12], this paper uses big data and machine learning techniques to differentiate between high and low bycatch occurrence. The paper "Deep learning methods applied to electronic monitoring data: automated catch event detection for long-line fishing" [13]. In this paper, they use EM systems to get video data and then used deep learn-

ing techniques to extract video segments of catch events. Most of the work done is based on footage from long-line fishing and little effort has been done to adapt such systems to small-scale fishing which usually has a different field of view. It requires a solution that is cost-effective and efficient at the same time to be adopted by small-scale fisheries.

1.2 Purpose

The objective of this study is to develop a machine learning model which detects bycatch events in REM video data collected onboard small-scale fishing vessels. The model is intended to be capable of distinguishing between different species of fishes, birds, and marine mammals (Seals, Porpoise).
The initial step in the study is to try different object detection algorithms to pick one that works best for the given data and test its robustness to handle changes in the environment such as a change in camera angles, and different weather conditions. To draw insights from the results and document them.
The system also introduces a proof of concept of a federated Learning framework which helps in the training of models in different countries where the data information of the models is not shared instead the weights are shared and combined. This study will help in understanding the requirements to set up such a framework and analyze how an object detection model reacts and how it can be fine-tuned to produce promising results in the future.

2
Theory

This chapter goes through the basic theory underlying the work in this thesis. To start with a basic definition of object detection, convolutional neural networks (CNNs) are presented. The chapter further continues to explore the object detection algorithm YOLO and explains the evaluation metrics being used. Finally, this chapter also explores the fundamentals of federated learning in detail.

2.1 Object Detection

As the main objective is detecting bycatch visually, object detection became a natural choice. Object detection is to detect instances of a given class in an image. It can be performed using different machine learning algorithms and can be categorized as one-stage methods where the priority is given to the inference speed of the algorithm and as two-stage methods where the priority is given to the detection accuracy of the object of interest. The category of the network for various applications is decided by its use cases.

2.2 Convolutional Neural Networks

Convolutional Neural Networks (CNNs) is a combination of feature extractors (hidden layers) and classifiers (fully connected layer) that perform detection in an image frame. Some of the widely used objection detection algorithms are CNN based and are used in various fields since their introduction in 1989 by LeCun, Yann, et al. in their paper "Object recognition with gradient-based learning." [14] Where they reveal the CNN's ability to extract information from images and its ability to recognize multiple objects in the frame without any need for segmentation done explicitly when it learns the right features. From the 'AlexNet' [15] with a small 8-layer network introduced by Alex Krizhevsky et al in 2012 that won the 'ImageNet challenge'[16] to the 'ResNet'[17] by Kaiming He et. al introduced in 2015 with a much deep and complex network, CNN implementations has seen a lot of improvements. CNNs are the preferred algorithm of choice for several applications, especially in image classification.

2.2.1 Object Detection with CNNs

A CNN is usually used for feature extraction, consisting of Convolutional layers that downsample the input image and extract the important features, working together

with a classifier network consisting of fully connected layers. The figure below shows a simplified CNN model.

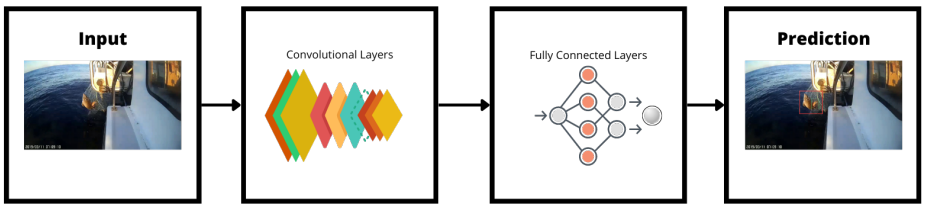


Figure 2.1: A Simplified CNN Model

The convolutional layers apply filters in the form of kernels to the pixel values of the image to extract features. When all the pixels in the images are processed through the convolutional layer it gives the output (features) to the fully connected layers which process these features to classify the input image.

2.3 You Only Look Once

YOLO[18] introduced in 2016 by Redmon, Joseph, et al is a one-stage object detection algorithm that aims to improve the inference speed of object detection. YOLO is a convolutional neural network based model that performs the prediction of bounding boxes and the class of the object in a single evaluation. Most object detection algorithms work as two-staged methods where the bounding box region is predicted initially before it is fed to a classifier network that identifies the class of the object. YOLO is said to be a suitable object detection algorithm for real-time inference because of its speed and was preferred over the Region-based Convolutional Neural Network (RCNN) [19] as they were slower compared to YOLO. RCNN's were two-staged and first finds the region of the bounding box and then try to identify the object using a classifier. Since its inception in 2016 YOLO[18] has seen a lot of improvements with every iteration.

2.3.1 Object Detection Using YOLO

The YOLO algorithm[20] divides the input image into a grid, each grid with a S x S region of equal size. The detection and localization of the object are contained in

each region. The computation power of YOLO is considerably reduced as a result of this. However, this results in a large number of duplicate predictions. The YOLO prediction contains bounding box coordinates, object labels, and the likelihood of the object class being present. YOLO overcomes the redundant predictions by employing Non-Maximal Suppression (NMS), which means that all bounding boxes with low probability scores are suppressed. By suppressing the bounding boxes with the largest intersection over union (IoU) with the current highest probability bounding box, this method is done multiple times until the highest probability bounding box is identified.

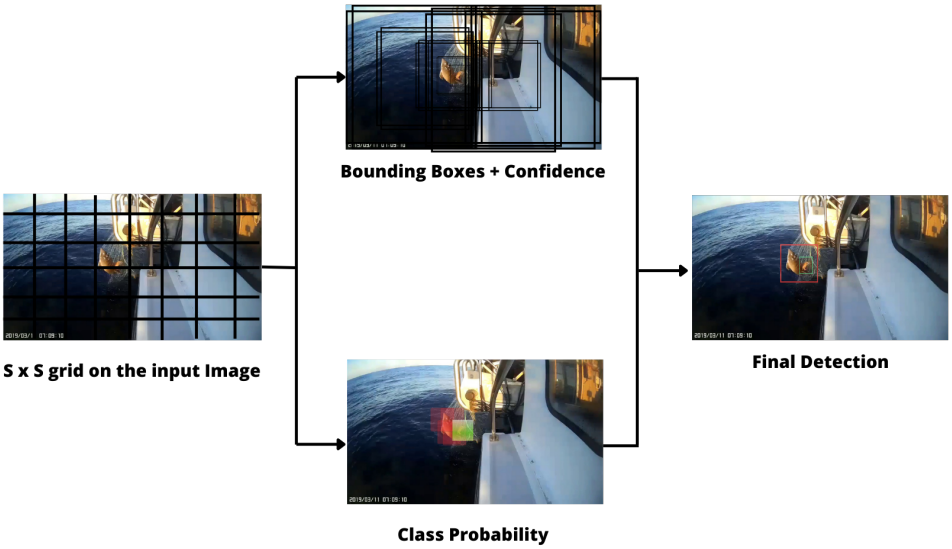


Figure 2.2: YOLO network process [21]

2.3.2 YOLO Architecture

To understand how the YOLO algorithm performs object detection let's look at its architecture, which consists of a Backbone and a Head. Surya Gutta et al [22] state that the backbone is used in the training of the model, and it can run on GPU or CPU platforms. The version YOLOv5 being used for this project uses CSPNet as the backbone similar to the previous version of YOLO i.e. YOLOv4. The head is either one stage or two stages based on the type of prediction being performed i.e. Dense prediction or Sparse prediction. Inspired by the GoogleNet architecture, YOLO's architecture consists of 24 convolutional layers [20].

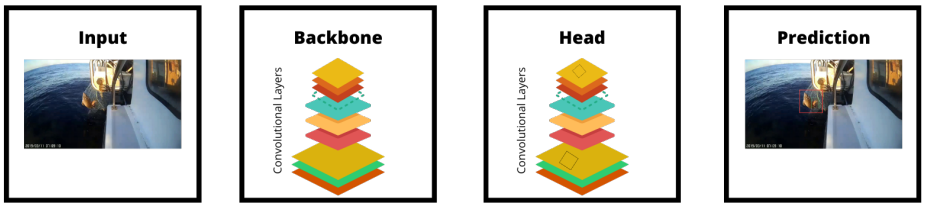


Figure 2.3: YOLO Architecture

2.3.3 Evaluation Metrics

The main tasks of object detection are usually divided into two parts: number one being Object Classification i.e. to classify the object into different classes present in the images. Number two: regression, localization of the object i.e. to see where the object lies in the image and to predict the coordinates of the bounding box.
One factor to consider when evaluating an object detection model is that some object classes may appear more frequently than others. Due to this difference, the accuracy(number of correct predictions) might not be the best way to represent the quality of the model as the dataset is imbalanced which can cause biased prediction. So, to evaluate the object detection models the best way is to use Mean Average Precision (mAP), and to understand mAP we need to know about the following performance metrics[10].

- Precision
- Recall
- Intersection Over Union (IoU)
- Average Precision
- Mean Average Precision

2.3.3.1 Precision and Recall

Precision and Recall[10] are a better choice of evaluation metrics than accuracy in the case of imbalanced datasets as they are not affected by the distribution of the classes in the dataset. Precision is the measure of how accurate the prediction of the model is i.e how accurate the percentage of the predictions are correct. The higher

the precision the more confident the model is in classifying the image as positive. Whereas Recall measures how good the model finds all the positives meaning how the model finds the positive cases in the top predictions, the higher the recall the more the model correctly classifies the image sample as positive. Mathematically precision and recall can be represented as

$$Precision = \frac{TP}{TP + FP}$$

$$Recall = \frac{TP}{TP + FN}$$

Where,

$$TP = TruePositive$$

$$FP = FalsePositive$$

$$FN = FalseNegative$$

In our case, True positive is when the model detects a class e.g. Porpoise as porpoise, False positive is when the model detects an object but there is no object present or detects the wrong class. False negative in our case is when the model does not detect any class object.

2.3.3.2 Intersection Over Union (IoU)

IoU [23] is the measure of the overlap between two bounding boxes, i.e., comparing the actual bounding box (the ground truth) with the predicted bounding box. The IoU score is between 1 and 0, where the higher the IoU value, the closer the predicted value is to the actual value.

$$IoU = \frac{AreaofOverlap}{AreaofUnion}$$

Where the area of overlap is computed between the predicted bounding box and the actual bounding box (The ground truth). The area of union is the area bound between both the predicted and the actual bounding box.

2.3.3.3 Average Precision (AP)

The average precision (AP) is a method of summarizing the precision-recall curve into a single number that represents the average of all precision. The precision-recall curve helps in the selection of the best threshold to maximize the precision and recall metrics by showing the trade-off between the two metrics.
The following equation is used to determine the AP.

$$AP = \sum_{k=0}^{k=n-1} [recall(k) - recalls(k+1)] * precision(k)$$

Where recall(n) = 0, precision(n) = 1 and n is the number of thresholds (The choice to turn the predicted probability into a class label[24]). The difference between

the current and next recall is determined using a loop that passes through all precision/recalls, and then multiplied by the current precision. To put it another way, the AP is the weighted sum of precision at each threshold, with the weight representing the increase in recall.

2.3.3.4 Mean Average Precision (mAP)

The Mean Average Precision [25] is the most frequent parameter for evaluating object detection models (mAP). mAP calculates a score by comparing the ground truth and predicted bounding boxes. The higher the score, the more accurate the model is. mAP is the mean of the Average Precision of all the classes.

$$mAP = \frac{1}{n} \sum AP_i$$

2.4 Federated Learning

Federated learning was introduced in 2016 by Google LLC in their paper 'Communication Efficient Learning of Deep Networks from Decentralized Data' [26] together with 'Federated optimization: Distributed machine learning for on-device intelligence'[27]. Federated learning is a decentralised learning technique where not one, but multiple local models are trained. In a federated framework, multiple devices are called clients, each of which participates in the training of a local model which is then combined to generate a global model that is sent back to all the clients to run inference on data and also to be trained further until the next sync happens with the global model generation.

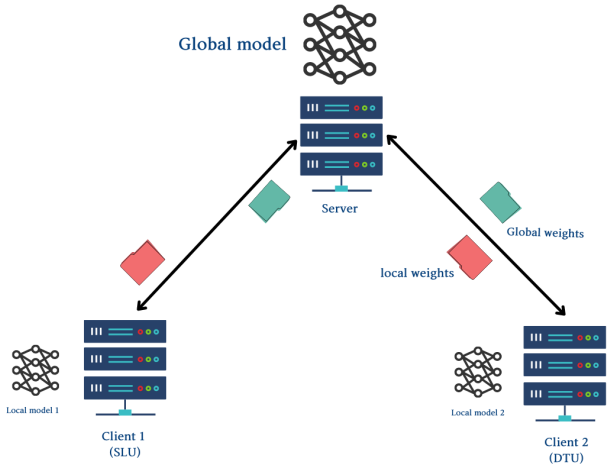


Figure 2.4: Federated Learning Framework

Google further investigates with top universities, the challenges faced in implementing federated learning in their paper 'Advances and Open Problems in Federated Learning'[28]. They discuss about how it comes down to be an interdisciplinary problem where only machine learning would not be sufficient to achieve a federated learning architecture and how it involves statistics, distributed optimisation, security, and privacy to name a few of them. Federated learning addresses the privacy concern over data sharing as well as the cost involved in handling large datasets as well. A federated approach ensures data protection as the data never leaves the client nodes and only the weights of the model are communicated to and from the local clients. It also allows the opportunity for clients to learn from each other.

2.4.1 FEDn framework from Scaleout

Scaleout introduced the FEDn framework in their paper [29] Scalable federated machine learning with FEDn. In their paper, they discuss how the existing federated learning setups have been more experimental and how a need for a framework that is highly scalable and able to handle both cross-devices and cross-silo setups. Cross-devices are when the client nodes are widely distributed with devices like mobile phones or IOT devices. Cross-devices train on the data they generate with the computing power available to them locally. Cross-silo is when large nodes with huge amounts of data and heavy computational power are available.
FEDn is based on the map-reduce programming model to be a robust learning framework that can be scalable in both scenarios where more edge-clients are added or if the model sizes increase largely. It has been tested and bench-marked in both cross-device and cross-silo machine learning models. It also allows for the deployment of a locally implemented system to a production level distributed framework with little to no change.

2.4.1.1 FEDn Architecture

There are three tiers to the architecture that the FEDn framework is based on. The clients make up the first tier. A local model is run on private client data on the client's behalf. The combiner and reducer make up the second tier. They play a significant role in the FEDn network's construction. The combiner is responsible for coordinating the updates from their subset of the clients. Each combiner is also responsible for the partial model update in the global rounds of the federated training, these partial models are the aggregation from the combiners associated clients. The third tier has two key components the controller and the discovery system, which are responsible for maintaining the global models by coordinating the computation in the global training round. The discovery system receives the client connection request and assigns the clients to the combiners on the network.

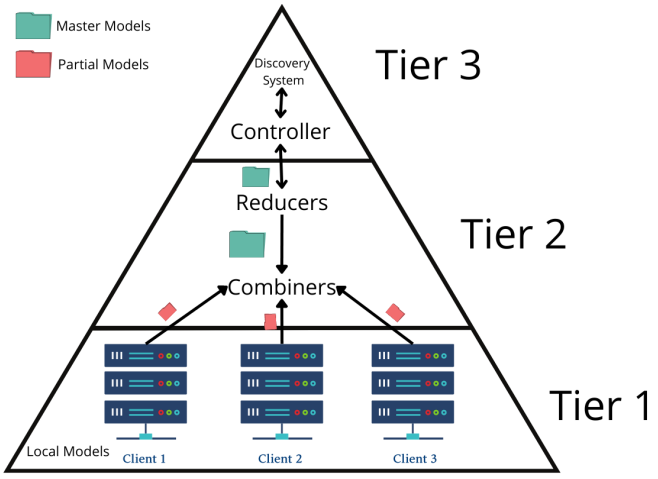


Figure 2.5: FEDn Architecture

2.4.1.2 Combiner

The Combiner acts as a stateless gRPC server with the main role to coordinate client updates and aggregate the model updates. During the model update round each combiner operates independently working on one and only one partial model update. The reason why the combiner is called a stateless gRPC server is so that the network is highly fault-tolerant i.e. if a combiner stops responding it can be easily replaced, only the updates related to that client which was associated with the combiner will be lost for that round. The combiner provides three types of services

- Connector Service: Connect the Client to the REST Service.
- Control Service: Connects the Combiner client to the controller.
- Model Service: Connects the client model to the model stored in the Discovery service.

2.4.1.3 Federated Averaging

The global weights in the FEDn are calculated using federated averaging. Federated Averaging(FedAvg) is a version of stochastic gradient descent used in a decentralized learning setup. M is the number of active client participants. The clients receive global weights and train the model for k epochs locally and perform a complete pass over of their data set D_k. The updated local weights W_k after k epochs are sent back to the server to be averaged.

$$w^{(i)} = \sum_{k=1}^{M} \frac{n_k}{n} w_k^{(i)}.$$

The above formula is used to calculate the global weights that are sent back to the M clients.

2.4.1.4 Reducer

The reducer's main function is to combine all the partial model updates into a single global model in each round computed by the combiners. A reducer is responsible for the preparation of the global model. The reducer can only communicate with the combiner and does not communicate with the clients.

2.4.1.5 Client

The client is a local system and is the only component in the entire architecture with direct access to the datasets and is responsible for running the local models that generate weights. A client's communication to the FEDn network is through its combiner assignments. The combiner assigns the train and validation tasks to the client, which in turn sends back the generated results for every request.

2.4.1.6 Initiating Federated Training

To initiate training a package containing the client folder is uploaded to the controller, the client folder contains the fedn. YAML file that contains the entry points namely 'train' and 'validate' to the entrypoint file. The entrypoint file contains the core python code required to train, validate the client model, and also to sync weights received from the combiner.

3
Tools

This chapter explores the tools which were used to collect, annotate and augment the data. It also lists the details of the computing hardware used.

3.1 Data Annotation and Augmentation Tools

Tools used for annotation as well as applying augmentation to the dataset.

3.1.1 Make-Sense

'Makesense.ai' [30] is an open source image annotation tool. It provides the user with a choice to use a pre-trained SSD model trained on the COCO dataset, which is handy for ML projects that deal with similar classes as the ones present in the COCO dataset. It provides a Posenet model for pose estimation of humans as well. It can also be used to annotate images manually on projects that can not leverage its ML models and provides the following annotation formats for object detection models.

- A CSV file
- VOC XML format
- YOLO Format

It is user-friendly and can be used locally when the dataset is private and is not computationally heavy as well.

3.1.2 VGG

VGG Image Annotator [31], is a simple and standalone manual annotation software for image annotations. It runs on the web browser and does not requires any installation and can be used as an offline application. The VGG Image Annotator lets you download annotations in the following format

- A CSV file
- JSON file
- COCO annotation format

3.1.3 Albumentation

Albumentation[32] is a data augmentation package that is user-friendly and has a wide range of image augmentations readily available. It was recently integrated to the YOLOv5 workflow and is used automatically during training if the package is available in the system. The required augmentations and their hyperparameters are mentioned in the train function of YOLOv5. The albumentation website allows for a demo site that lets the user see how different parameters affect an image when used and comes in handy to tune the augmentation applied to the dataset while training.

3.2 Data Collection Tools

The following tools were used by SLU and DTU to collect the video feeds.

3.2.1 Black Box Analyzer

Black Box Analyzer is a software product from Anchorlabs[33] that is widely used to perform video feed analysis in the fishing sector. They offer a different range of functionalities from mapping the GPS route of a given boat with its video feed. It is used to perform manual video analysis. It follows a more traditional software approach where the user is allowed to define an event and a flag is raised on every occurrence. To view sensor data as well among other things. It also lets the user generate a fishing report and trip reports as CSVs to be revived and for data warehousing purposes.

3.2.2 Camera System SLU X

SLU X is an in-house camera system custom built and used by SLU to perform remote electronic monitoring(REM) especially focused on small-scale boats.

Figure 3.1: SLU X

It is developed to be user-friendly to be easily installed with intuitive and straightforward instructions. It has features that allow on-board live feed monitoring and the option to configure video resolution based on the storage availability from a maximum of 60fps to a preconfigured 10fps as well. The setup also has a built-in GPS sensor that collects the geographical location of the boats as well.

3.3 Docker

Docker is a development platform used for deploying and running apps in a controlled environment. It offers OS-level virtualization and packages the software so that it runs on any hardware it is deployed on. It builds images that provide instructions on how to build a docker container with the requirements mentioned in a file called the 'dockerfile'. Many containers can be run simultaneously as they are isolated from each other. It offers a lot of functionality to build on an existing image by customization and mounting storage to the container to initiating another container. A setup like the federated framework would benefit a lot from using a docker environment as it provides isolated containers that can run on any number of devices if it's customized properly based on the needs of the client system.

3.4 Computing Hardware

The computing hardware was provided by AI Sweden's Edge Lab. It is a shared testbed offered by AI Sweden for its partnering organizations. The computing hardware includes a 'Quadro RTX 5000'[34] graphics card from NVIDIA and has total storage of 300 GB with an external drive mounted to it with 10 TB of space to house the data from SLU. The DTU computing hardware includes a Tesla T4 graphics card from NVIDIA and has a storage of 300 GB as well. In addition to the RTX 5000, there was a DGX-A100 also on standby in AI Sweden Edgel Lab for use.

4
Data

This chapter explains how the dataset was built with the available data. The chapter offers a detailed explanation of the datasets and how they were preprocessed. Finally, how the training, validation, and test divisions were made.

4.1 Data Collection

The Swedish University of Agricultural Sciences (SLU) aquatic resources department have developed custom Remote Electronic Monitoring System (REM) hardware for recording video and GPS data from small-scale fishing vessels. The system has two cameras that can be positioned to cover different areas of a fishing boat. On small-scale gillnet vessels monitored by SLU, one camera (channel 1) films over the side of the boat where the net leaves the water and the other camera (channel 2) films on deck footage of the catch being removed from the gillnets (see figure 4.3 and 4.1). A similar setup is used by the Technical University of Denmark (DTU) to monitor Danish gillnet fisheries.

4.2 SLU's Data

SLU has collected over 10 gigabytes of video feeds from small-scale fishing vessels over the course of two years. The average length of the videos varies on how long the vessel is on the trip. Each video consists of 10 frames per second. The video also shows the date and time on the left bottom corner for easy documentation. The data was obtained from 12 different boats that fish in various locations of the Baltic Sea. The data is protected by a special contract between the fishermen and SLU, for it to be used in the research. Some examples of what the SLU data looks like is shown in figures 4.1 and 4.3.

Figure 4.1: Channel 1 (Pointing outside the boat)

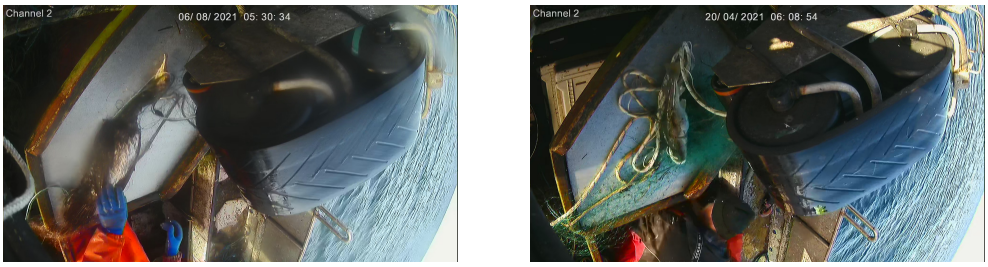


Figure 4.2: Channel 2 (Pointing inside the boat)

4.2.1 SLU DataSet

The video feeds from the REM are collected from time to time and stored at SLU. The current setup at SLU has an analyzer software (black-box analyzer created by Anchor labs [33]) that syncs the video feeds with the GPS information. It is used to identify and tag bycatch manually, going over hours of footage. The analyzer software generates reports of trips a boat takes along with a catch report. This setup was leveraged to generate a dataset that could be used to train ML models.

4.2.1.1 Class list

As a first step, it was necessary to determine the specie and non specie classes that were to be detected by the ML algorithm. In addition to porpoise, sharks and rays, different fish and bird species were also considered as a separate bycatch class. However, this led to an imbalance in the dataset with some of the bird species and fish species being observed only once or twice. It was decided to finalise a class list of species that are to be considered as bycatches based on the amount of data available to train, validate and test a class to a reasonable extent. Listed below are the 13 classes that were used for classification.

- Porpoise
- Seal
- Bird
- Ray
- Shark
- Dolphin

} Bycatch Classes

- Crustacean
- Banana_Pinger
- Unknown
- Bouy
- Fish
- Litter
- FO_Pinger

} Other Classes

Some examples of how the other classes look like

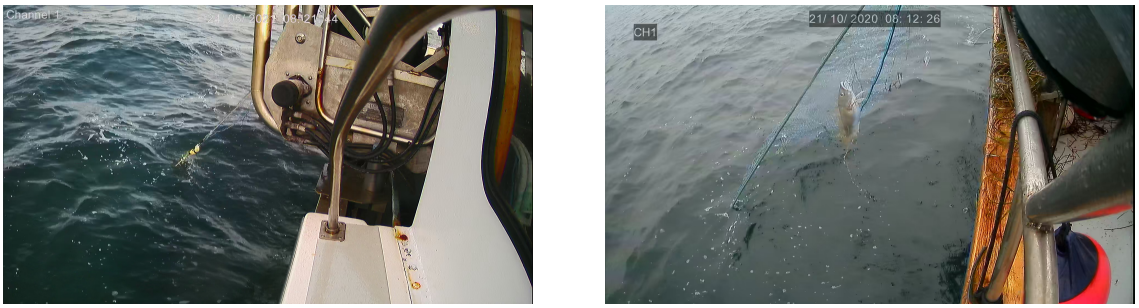


Figure 4.3: Banana Pinger(left), Fish (right)

Among the 13 classes there are 6 bycatch classes, namely Porpoise, Seal, Bird, Ray, Shark and Dolphin. As discussed earlier, bycatches are rare occurrences and the data available on these classes are limited. Listed below are the total count of image frames with objects of interest obtained from the SLU footage.

Bycatch class	Count
Porpoise	1074
Seal	431
Shark	164
Ray	44
Bird	300
Total	2013

Table 4.1: Bycatches classes in SLU data

In addition, we included a number of non-animal classes (Banana Pinger, FO Pinger, Unknown, Bouy and Litter). Pingers are devices attached to the gillnets that broadcast a frequency wave that keep porpoises at a distance from the net to avoid them being accidentally bycaught in the net. They were included to study their operation and if they actually make an impact and how they could be improved. The Bouy class was included to identify individual fishing hauls in the hopes of narrowing bycatch events to individual fishing hauls in the future.

4.2.1.2 Frame Generation

As an initial step towards preparing data for training, catch reports were collected from the black box analyzer. A manual analysis was performed to filter out the reports that contain bycatch entries. The selected reports were used to identify the video from fishing trips that contained bycatch data. As a way to extract video segments from the clips a python script was developed. The script goes over all the entries in the catch report and identifies the video files. It then calculates the video time that the bycatch was made and generates a detailed report which is finally used to generate short video segments of the bycatch. The video segments are finally sent to another python script that generates frames.The flow is shown in figure 4.4

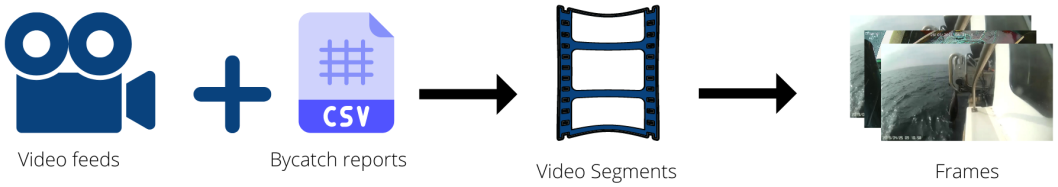


Figure 4.4: Flow chart of how frames were generated

A time interval of 10-5 seconds before and after the catch event occurs was included in each short video segment. The inclusion of non-object frames allows the model to learn how the background looks when there are no object of interest. Below is a snapshot of the final data frame that is used to generate video segments with the bycatch class information in them.

	Start-time	Catch-time	End-time	Species	Filename	Channel	Catchdate	Segment-start
0	07:46:40	07:50:56	07:55:06	PINGER	CH01-20210621-074640-075506-001000000000.mp4	CH01	20210621	00:04:16
1	07:46:40	07:53:40	07:55:06	PINGER	CH01-20210621-074640-075506-001000000000.mp4	CH01	20210621	00:07:00
2	07:55:06	07:57:45	08:03:33	PINGER	CH01-20210621-075506-080333-001000000000.mp4	CH01	20210621	00:02:39
3	07:55:06	07:59:56	08:03:33	PORPOISE	CH01-20210621-075506-080333-001000000000.mp4	CH01	20210621	00:04:50
4	07:55:06	08:03:07	08:03:33	PINGER	CH01-20210621-075506-080333-001000000000.mp4	CH01	20210621	00:08:01

Figure 4.5: Final Video Segmentation Dataframe

Segment-start column states the time when the catch event was registered in the black box analyzer report.

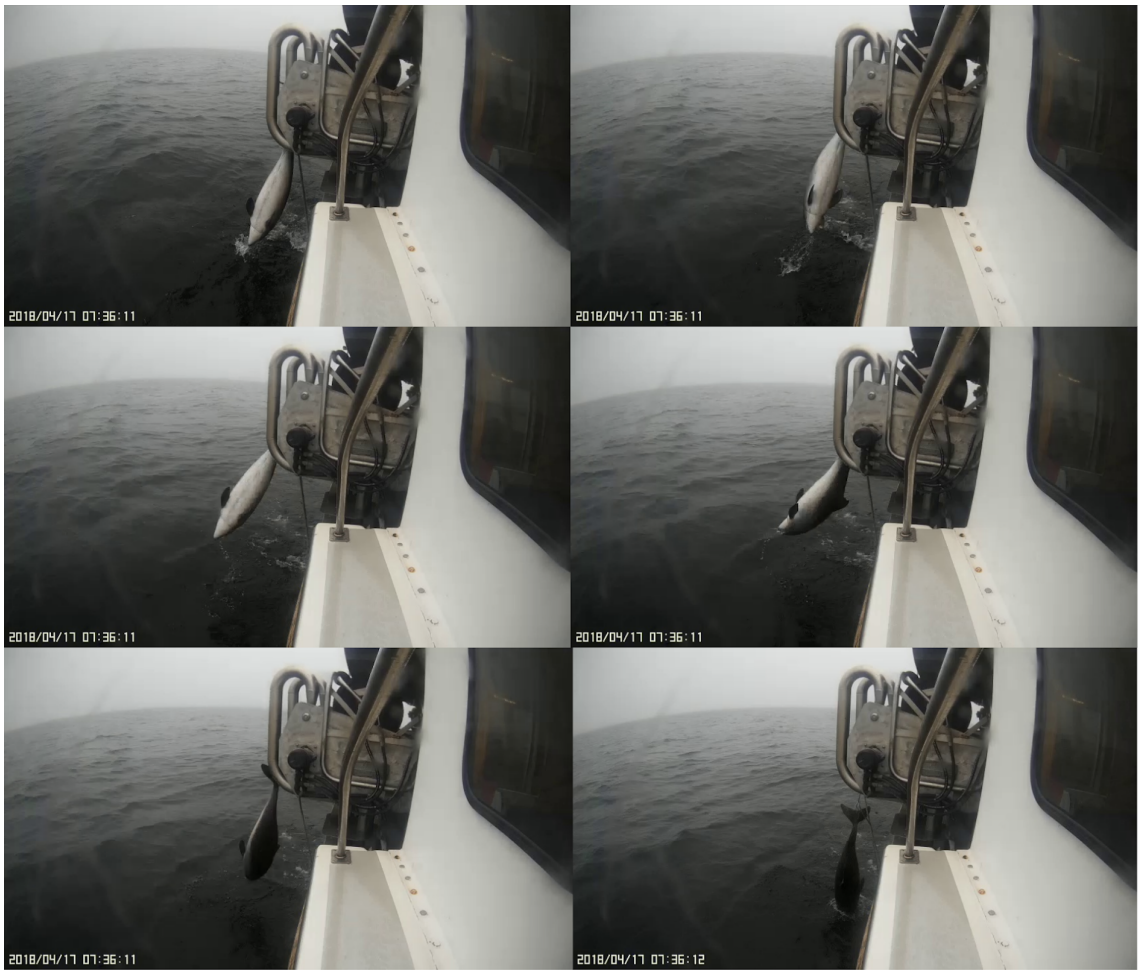


Figure 4.6: An example of a 6 frame sequence that was generated from a short bycatch video segment. The frames show a porpoise bycatch event captured by the channel 1 over the side camera

An example of a sequence of frames that was generated from the video segments is shown in Figure 4.6. This sequence is one of the many which were generated using the process explained in 4.4.

4.2.1.3 Data Annotation

The generated frames were annotated to be used to train an object detection algorithm. This was performed using intuitive tool called Make-sense[30]. The tool was hosted locally in the annotator's computer as the images were protected by GDPR and could only be viewed and shared by authorized individuals. The annotations are in the form of TXT files containing coordinates and size of the bounding box with a label indicating the class present inside the bounding box (see figure 4.7 for an example).This annotation format is specific for YOLO algorithm[35] and can be called YOLO format or Darknet Format. To generate the annotations as mentioned above Make-sense tool was used where a bounding box was created by drag and draw method around the object in the images and then labeled with the required class, The figure 4.7 shows how the annotation looks like in Make-sense tool.

Figure 4.7: An example annotated bycatch frame, from the SLU dataset, showing a Porpoise inside a bounding box. The bounding box is labelled by class in the accompanying text file that is generated

Figure 4.8 shows how the TXT file for one image looks like. From left to right the annotation contains the class label(shown in table 4.2),the coordinates of the bounding box i.e. where it lies on the image with respect to the image size and center of the image and height and width of the bounding box.

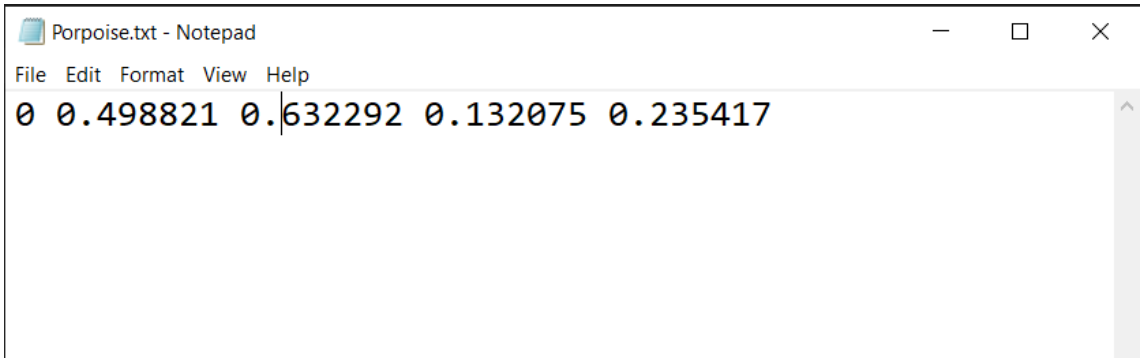


Figure 4.8: Example YOLO annotation (SLU dataset)

Below is a summary of the the number of annotations made for each class label in the SLU dataset.

Class label	Count
Fish	1229
Crustacean	421
Buoy	23
Banana Pinger	254
FO Pinger	15
Unknown	201
Litter	17
Porpoise	1074
Seal	431
Shark	164
Ray	44
Dolphin	0
Bird	300
Total	4173

Table 4.2: All the classes in SLU data

In addition to the 13 classes present, there are 1362 empty(no class present) frames present in the dataset as well. Each frame consists of a different background environment, which shows which boat the video is from. The addition of empty frames made splitting the dataset into training, validation, and testing a bit tricky. Due to the uniqueness of the background environment, it had to be taken into consideration that not all frames should be from the same boat. This would give identical information with minor changes and could lead to misleading results. Hence, the splitting was done manually with these points under consideration.

Bycatch Class	**Train**	**Test**	**Validation**
Shark	135	8	21
Ray	37	3	4
Seal	336	30	35
Porpoise	826	48	200
Bird	396	11	20

Table 4.3: Division of Bycatch classes into train, test, validation of SLU datset

There is a visible imbalance of most bycatch classes and it is hard to measure the model's performance in such classes and require more data to make a fair assessment. The images for validation and test are select on the basis that they were from the same boat on a different day and have visibly similar environment but not exactly identical to the train images. It would be fair to assume that the model's performance will increase with inclusion of more data and some agumentation.

4.3 DTU's Data

DTU has been gathering data from small scale fishing vessels for nearly ten years and currently has around 75 terabytes of footage. From these video feeds there are approximately 8781 images which consists of mostly bycatch classes. Where approximately 5500 are of rays, 400 of porpoises, 700 of fishes, 1000 of birds, and the rest classes (see table 4.5).
Some examples of what the DTU data looks like are shown in figures 4.9 and 4.10

Figure 4.9: Two example frames from Channel 1 camera (pointing outside the boat)

Figure 4.10: Channel 2 (Camera pointing inside the boat)

4.3.1 DTU Dataset

Similar to SLU, DTU videos are also gathered using REM on danish fishing vessels large and small. The current setup is using black-box analyzer to identify and tag the bycatch class manually.

4.3.2 Class list

The original labeling on the DTU dataset was different then the required one as the dataset was already annotated. The original labels were divided into four main parent classes and each parent class had a number of sub classes under them. Parent classes were generic and did not contain specie level information. Below is a list of all the parent classes present in the DTU dataset.

- Fish
- Mammals
- Birds
- Elasmobranch
- Other

The sub classes had the specie level information and can be viewed below.

1. Fish
   (a) Turbot
   (b) Monkfish
   (c) COD
2. Mammals
   (a) Porpoise
   (b) Grey Seal
   (c) Minke Whale
3. Birds
   (a) Common Eider
   (b) Velvet Scoter
   (c) Northern Fulmar
4. Elasmobranch
   (a) Sharks
   (b) Blonde Ray
   (c) Cat Shark
5. Other
   (a) Litter
   (b) Crabs
   (c) Human

The original classes division for DTU dataset is shown in figure 4.11

	bird	mammal	elasmobranch	fish	other
1	☐ Sm (common eider) ☐ Pc (great cormorant) ☐ Ua (common guillemot) ☐ At (razorbill) ☐ Alcidae (undetermined alcid) ☐ Anatidae (undetermined duck) ☐ Mf (velvet scoter) ☐ Mn (common scoter) ☐ Mel (undetermined scoter) ☐ Gaviidae (undetermined loon) ☐ Ga (black-throated loon) ☐ Gi (common loon) ☐ Larus (undetermined gull) ☐ Lm (greater black-backed seagull) ☐ Podicipedidae (undetermined grebe) ☐ Podic (great crested grebe) ☐ Pg (red-necked grebe) ☐ Fg (northern fulmar) ☐ Mb (northern gannet) ☐ Nl (undetermined bird)	☐ Pp (harbour porpoise) ☐ Se (undetermined seal) ☐ Hg (grey seal) ☐ Pv (harbour seal) ☐ Ba (Minke whale) ☐ La (white-beaked dolphin)	☐ shark (undetermined shark) ☐ Rb (blonde ray) ☐ Rm (spotted ray) ☐ Sa (spurdog) ☐ Sc (catshark) ☐ Gg (tope) ☐ Ray (undetermined ray) ☐ Mas (starry smooth hound)	☐ Rf (all species of roundfish) ☐ Ff (all species of flatfish) ☐ Gm (cod) ☐ Mm (ling) ☐ Cl (lumpsucker) ☐ Pvi (saithe) ☐ Lp (monkfish) ☐ Pf (European flounder) ☐ Ppl (European plaice) ☐ Sma (turbot) ☐ Scsc (Atlantic mackerel) ☐ Sso (common sole)	◯ HUMAN (Fisher) ◯ PLASTIC ◯ Hgammarus (European lobster) ◯ Crab (all spp) ◯ Starfish (all spp)

Figure 4.11: Original Class Division

4.3.3 Data Annotation

DTU dataset was annotated with the help of VGG image annotator and was provided in the csv format which was then converted into YOLO annotation format using a script. The conversion from the original annotations to the one required is shown below.

	filename	file_size	file_attributes	region_count	region_id	region_shape_attributes	region_attributes	class	x	y	width	height
0	29572_20220309-05443665_Sm_SEEN_male.jpg	387273	{}	2	0	{""name"":""rect"",""x"":546,""y"":113,""width...	{""bird"":{""Sm"":true},""mammal"":{},""elasmo...	bird	546	113	173	250
1	29572_20220309-05443665_Sm_SEEN_male.jpg	387273	{}	2	1	{""name"":""rect"",""x"":412,""y"":1,""width""...	{""bird"":{},""mammal"":{},""elasmobranch"":{}...	unknown	412	1	384	155
2	29573_20220309-054637223_Sm_SEEN_male.jpg	382902	{}	2	0	{""name"":""rect"",""x"":486,""y"":259,""width...	{""bird"":{""Sm"":true},""mammal"":{},""elasmo...	bird	486	259	124	216
3	29573_20220309-054637223_Sm_SEEN_male.jpg	382902	{}	2	1	{""name"":""rect"",""x"":345,""y"":4,""width""...	{""bird"":{},""mammal"":{},""elasmobranch"":{}...	unknown	345	4	468	252
4	29570_20220307-05295069_Sm_SEEN_male.jpg	438173	{}	1	0	{""name"":""rect"",""x"":413,""y"":351,""width...	{""bird"":{""Sm"":true},""mammal"":{},""elasmo...	bird	413	351	96	201
...	...	...	...	...	...	...	...	...	...	...	...	...

Figure 4.12: Original Annotations

	filename	class	x	y	width	height
0	29572_20220309-05443665_Sm_SEEN_male.jpg	Bird	546	113	173	250
1	29572_20220309-05443665_Sm_SEEN_male.jpg	Unknown	412	1	384	155
2	29573_20220309-054637223_Sm_SEEN_male.jpg	Bird	486	259	124	216
3	29573_20220309-054637223_Sm_SEEN_male.jpg	Unknown	345	4	468	252
4	29570_20220307-05295069_Sm_SEEN_male.jpg	Bird	413	351	96	201
...	...	...	...	...	...	...

Figure 4.13: Annotations Converted into Required Classes

	filename	class	x	y	width	height	ImageHeight	ImageWidth	yolo_x	yolo_y	yolo_w	yolo_h
0	29572_20220309-05443665_Sm_SEEN_male.jpg	Bird	546	113	173	250	768	1360	0.465074	0.309896	0.127206	0.325521
1	29572_20220309-05443665_Sm_SEEN_male.jpg	Unknown	412	1	384	155	768	1360	0.444118	0.102214	0.282353	0.201823
2	29573_20220309-054637223_Sm_SEEN_male.jpg	Bird	486	259	124	216	768	1360	0.402941	0.477865	0.091176	0.281250
3	29573_20220309-054637223_Sm_SEEN_male.jpg	Unknown	345	4	468	252	768	1360	0.425735	0.169271	0.344118	0.328125
4	29570_20220307-05295069_Sm_SEEN_male.jpg	Bird	413	351	96	201	768	1360	0.338971	0.587891	0.070588	0.261719
...	...	...	...	...	...	...	...	...	...	...	...	...

Figure 4.14: Final CSV Annotations with the YOLO Annotations

An example of the final annotation of one of the image from the DTU dataset with class Porpoise is shown in figure 4.15 .

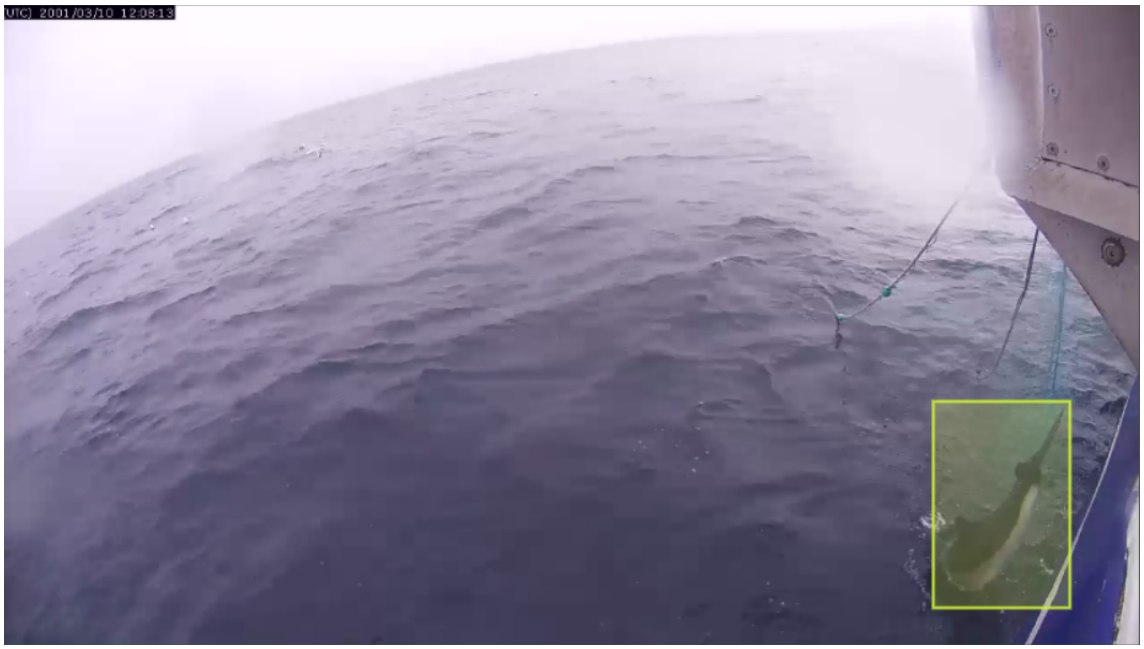


Figure 4.15: Example Image Annotated of DTU dataset (Porpoise)

The annotation of the above image looks something as follow.

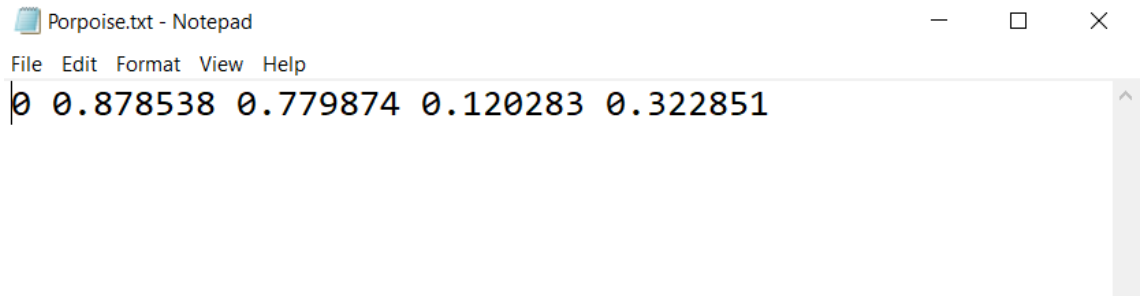


Figure 4.16: Example YOLO annotation (DTU dataset)

Once the annotation was sorted the image count decreased to 7800 as some of the images did not contain any annotations. The bycatch classes count looks something as follow for DTU dataset.

Class Label	Count
Shark	315
Ray	5110
Seal	211
Porpoise	377
Bird	820

Table 4.4: Bycatach classes in DTU data

Class Label	Count
Fish	1135
Shark	315
Ray	5110
Seal	211
Porpoise	377
Dolphin	0
Crustacean	113
Bouy	0
Bird	820
Banana_Pinger	0
FO_Pinger	0
Unknown	6903
Litter	7

Table 4.5: All classes in DTU data

From the above table, it is calculated to have a total number of 14991 annotations. For the DTU dataset, there are only images with instances of catch and bycatch in them, so there aren't any background images. The main purpose of the thesis is to focus on the bycatch events. The table below shows the bycatch classes and the count of their occurrences in the DTU data, with a total of 7233 instances. To train the model the dataset was divided into a division of 70/20/10, train,test and validation ,The dataset generation is based on random sampling.Table 4.6 shows the division count of the bycatch classes.

Bycatch Class	**Train**	**Test**	**Validation**
Shark	233	54	24
Ray	3671	1028	411
Seal	148	48	15
Porpoise	277	77	23
Bird	590	161	69

Table 4.6: Division of Bycatch classes into train, test, validation of DTU datset

4.4 Comparison of SLU and DTU datasets

Although both the datasets were sourced from small scale boats that operate in the baltic Sea both SLU and DTU generate datasets that are significantly different both in scale and the quality of the footage generated. DTU has been conducting the bycatch research for over a decade and use camera gear that is reasonably old and generate low quality frames, whereas SLU started collecting footage roughly over two years ago and have custom made in house camera system specially developed to observe bycatch. SLU cameras generates high definition frames. The ideal setup would be to build a dataset from both SLU and DTU, but such a task would be

time consuming and would require a lot of resource to consolidate them in a single location. It would also require a lot more computing power to train on such a large dataset. The footage being onboard boats also posed a privacy issue from it being shared between collaborators (also between countries). Before getting into details on a potential machine learning model that would work for both SLU and DTU, a closer inspection at the datasets(both SLU and DTU) is required.

Figure 4.17: A side by side comparison of how the data looks for the class porpoise in SLU and DTU

In comparison SLU has very few occurrences of bycatches but with high quality images in the dataset and also has sequence of frames. While on the other hand DTU has comparatively large number of images of low quality and has only one image per bycatch event. In the figure 4.17 it is seen that that SLU's video feed looks much warmer with clear details while the DTU's image looks a lot dimmer and is more pixelated. It is also observed that most images of the DTU dataset does not include the boat in the frame and are stable angles among all the boats.

Bycatch Class	**SLU**	**DTU**
Shark	164	315
Ray	44	5110
Seal	431	211
Porpoise	1074	377
Bird	300	820

Table 4.7: Comparison of Bycatch classes between SLU and DTU

The table 4.7 shows the count comparison of individual bycatch classes between SLU and DTU. To explore a few of them we notice that SLU has reasonably large

number of porpoises compared to DTU but it should be noted that DTU's count show 377 different occurrences of porpoises while SLU's 1074 is sourced from a total of 13 bycatch occurrences. Most porpoise images in SLU have similar features with slight changes in them. DTU has over 5110 images of rays while SLU has only 44 images that were sourced from 4 different bycatch occurrences that were annotated.

5
Methods

The chapter explores the machine learning for object detection and federated learning methods used in the project, how different types of YOLOv5 configurations were trained on SLU and DTU datasets, and finally explores the FEDn framework with both DTU and SLU as clients.

5.1 Object detection

Different configurations of YOLOv5 were considered to perform object detection and classification. The hyperparameters of these configurations were the same for both SLU and DTU model training. The optimiser used was the Stochastic gradient descent(SGD), and the images were resized to a uniform size of 640X640. The learning rate of the network was set to 0.01 and to avoid the model from overfitting to the training dataset the weight decay was set to 0.0005. The momentum of the learning rate change was set to 0.93. The first three epochs of the training used for warmup with the momentum was set to 0.8. Below is an overview of the network depth and the parameters they use.

ML Model	Layers	Parameters
YOLOv5-Nano	270	1781506
YOLOv5-Small	270	7054690
YOLOv5-Medium	369	20919810
YOLOv5-Large	468	46149064
YOLOv5x	567	86298562

Table 5.1: Object detection models and parameters

5.1.1 Object detection on SLU Dataset

All models were trained on NVIDIA Quadro RTX 5000[34], 16125MiB graphic card for a total of 60 epochs. The models were trained on the custom dataset split that is shown in the table 4.6. The dataset being small and unbalanced needed augmentation to generalise and the following hyperparameters of YOLOv5 were used to perform augmentation. HSV-Saturation augmentation with 0.7, HSV-Hue augmentation with 0.015, and HSV-Value augmentation with 0.4. Image mosaics were constructed when training the networks. Apart from the inbuilt hyperparameter tuning available in YOLOv5, the albumentation package was used to implement

augmentation in the training dataset. The albumentation tool[32] is integrated into the yolov5 workflow and is easy to use. The following augmentations were applied to the dataset. Blur with a limit between 3 and 10 with a probability of 0.1. Median blur with a limit between 3 and 9 and a probability of 0.1. ToGray with a probability of 0.3, to convert images to grayscale. GaussNoise with a probability of 0.2 and a limit between 10 and 115 to simulate rainy and foggy frames. HorizontalFlip with a probability of 0.1 it was used so the model learns to identify environments on both left and right side of the boats and a VerticalFlip with a probability of 0.1 was also applied to train the model to identify upside down frames as well. CLAHE (Contrast Limited Adaptive Histogram Equalization) with a limit between 1 and 4. Image compression is also applied with a quality between 55 and 100.

5.1.2 Object Detection on DTU Dataset

All models were trained on NVIDIA Tesla T4 15109MiB[36] graphics card for a total of 60 epochs. The augmentation used on the DTU dataset was very minimum as it contained very diverse images with varying background information and difference in quality. Most of the augmentation done on the DTU dataset was needed to simulate the human errors and environmental changes which were observed during data preprocessing. Such as flipping the images to simulate cameras being put upside down, GaussNoise to simulate the rain or water droplets on the camera lens, and toGray to simulate the nighttime. The train, validate and test split of the dataset was random as they did not contain a sequence of frames as in the SLU dataset.

5.2 Federated Learning using the FEDn Framework

The federated framework setup on the proposed system has the following components. The database, reducer, and combiner (which will be collectively referred to as the server) and client 1 (SLU will be referred to as Client 1) is hosted at the edge lab at AI Sweden in Gothenburg, Sweden. Client2 (DTU will be referred to as Client 2) is hosted at DTU in Copenhagen, Denmark. The clients are required to connect to the edge lab's network to communicate with the server and receive tasks. The clients participate in the federated training by initiating the docker container and by being connected to the same network. Both clients have high-performance GPUs and large storage. The FEDn training between SLU and DTU is a cross-silo connection.

5.2.1 Server

The server houses the database, combiner, and reducer in it. The database is used to store the training results and the weights. The combiner communicates with the clients to assign training tasks to clients and transfer the aggregated weights from the reducer. The reducer receives the weights from all the combiners and aggregates them and sends back global weights that are sent back to the clients. The server

is initiated in a docker container and the controller UI is exposed to the 8090 port of the server's IP address. The UI is used to view maps of the connection between the components in the network (clients, reducer, combiners) and to view training progress, validation and metrics.

5.2.2 SLU Client

SLU client is also hosted at the same place as the server, which contains a docker container when initialised establishes a connection with the server is easily established. Client 1 has an NVIDIA Quadro RTX 5000[24], 16125MiB graphic card, and a 200GB storage unit.

5.2.3 DTU Client

DTU's data is stored at a data center in Copenhagen; it requires a VPN connection to the edge lab's network. As the stored data is private remote access into their system was not possible, so to solve this Client 2 is run on a workstation on loan from AI Sweden. Client 2 has an NVIDIA Tesla T4 15109MiB[36] graphics card and a 300GB storage unit. The combiner and the reducer IP addresses are given to client 2 while setting up the docker image. Client 2 sets up a connection to the edge lab and initiates the docker container. Once the container is up and running the client is visible on the dashboard and the network map on the controller's UI.

5.2.4 FEDn MNIST Experiment

To test the connection and the communication between DTU and Edge lab, MNIST classification training was performed using two clients. After the server and all the clients (client 1 and client 2) are initialized, the reducer UI is checked to see if all the clients are visible and ready for training. The reducer UI is used to upload an initial model for training along with an initial seed file, which is sent to the client. The initial seed is a compiled version of a simple neural network with three fully connected layers. The model's training can be observed on the dashboard. The training is observed until the model converges. This test run is done to help identify hardware requirements and the feasibility of such an experiment between stakeholders that are present in two different countries. The only communication between the server and the clients would be the requests to initialize training and the syncing of aggregated weights from the reducer on a timely basis. The clients are required to be in sync during training to avoid lag in weight aggregation.

5.2.5 YOLOv5 FEDn Framework

Training an object detection model such as the YOLOv5 requires a lot of computing power than a simple MNIST. As stated earlier both SLU have good GPUs and storage units to handle such a computationally heavy load.

5.2.5.1 Setting up Yolov5 network into FEDn Framework

The FEDn framework is model agnostic and only requires a few modifications to train clients on YOLOv5. The following modifications are done to train the object detection and classification model. The entry point file in the client folder is edited to adapt the training and validation functions to YOLOv5. The JSON object used to collect the validation results is also updated to store the evaluation metrics used to measure the performance of object detection models.
The Git repository [29] consists of a train function that was used to initialize the training call. The train function accepts the image size that determines the size of the image the YOLOv5 model trains on. The number of epochs to be run on the local model, model configuration(nano, small, medium, large, and extra large), The data parameter that has the dataset YAML file that contains the location of the images used for training and validation along with their labels, The weights, which are used to initialize training (updated with global weights after every round), The project parameters (used to determine the output directory where the run results are stored) as parameters to start training and to store the best weights and send them to the server.
Similarly to the train function, there is also a validate function that is used to validate the overall model performance. It gets the following parameters to start the validation or test run of the local models: The image size that the model was trained on, The batch size (which determines the number of batches the validation is run on), and the data parameter have the dataset YAML file that contains the location of the images used for validation along with their labels, The weights that are to be used to run inference and verbose is enabled to get results in every class.

5.2.6 Running FEDn Training between SLU and DTU

Before beginning training, a package of the client folder containing the entry point file and the fedn.yaml file is generated; a seed file containing the required configuration (Small, Nano, Medium, or Large) of the YOLOv5 model is also generated locally. To train a YOLOv5 Federated model, the server is initialised in a docker container using the docker-compose command, it also initialises the docker container of client 1. Next, the workstation housing client 2 is connected to the edge lab's network, and client 2 is initialised in a docker container. The 8090 port of the server is used to load the client package followed by the seed model that was generated locally. The network tab is used to verify if all the components are visible and if the clients are active.
The control tab is used to initialise the training process, and input parameters such as the number of rounds and if the models are required to be validated are selected. A wait time is selected which determines how long a model has before a sync is initiated after every round. Once the required parameters are selected, the training is initiated and the logs are observed at the client nodes and the dashboard as well. The metrics such as MAP, precision, and recall are observed for the training process. A more detailed overview is explored in the results chapter.

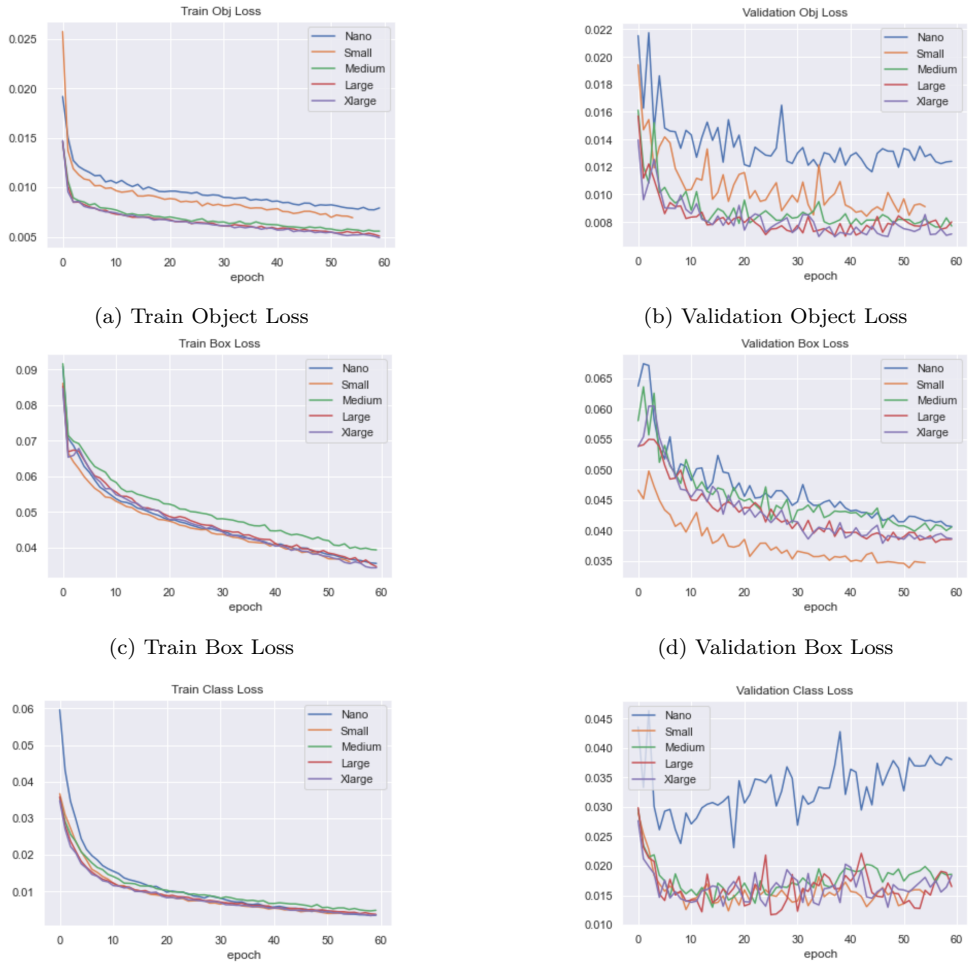
6
Results

This chapter presents the results of different YOLOv5 configurations trained on both datasets. It also shows how the results differ because of the different environments of data. It also shows how federated learning using the FEDn framework was performed on the mnist dataset. Finally how a FEDn YOLO model performs with DTU and SLU as its clients.

6.1 Object Detection and Classification

6.1.1 SLU Model

The comparison of training YOLOv5 configurations (nano, small, medium, large and extra large) over 60 epochs using the SLU dataset is reported and discussed below. Figure 6.1 shows the training and validation losses of all YOLOv5 configurations. it can be observed that the Nano configuration seems to be performing worse and is observed to start overfitting early on during training as observed on the 'Validation Class Loss'. The other model configurations seem to be performing similarly to each other, as the depth of the model increases from Small to XLarge we observe the model's improvement is very minimal on the given dataset. The validation box loss is the lowest in the Small model configuration. On closer observation, it is clear that the Small model configuration has less fluctuation in the validation class loss, even though it appears to be a little overfit. In 'validation obj loss' we observe that the performance improves as the model grows deeper from Nano to XLarge.



(a) Train Object Loss

(b) Validation Object Loss

(c) Train Box Loss

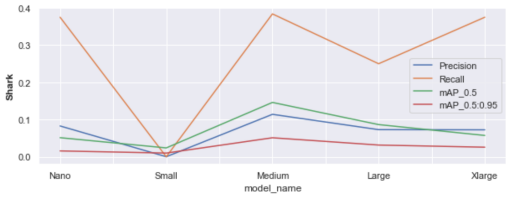
(d) Validation Box Loss

(e) Train Class Loss

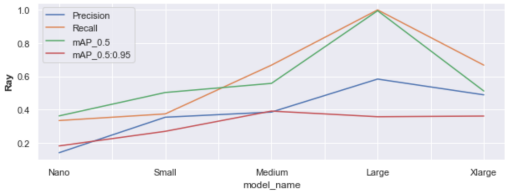
(f) Validation Class Loss

Figure 6.1: Validation and Train loss

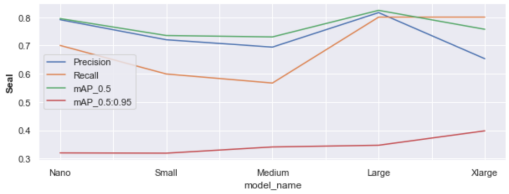
Figure 6.2 shows the performance result of all the model configurations, as the network layers increase in number we observe an increase in the performance of detection of most bycatch classes. These results should also be interpreted with the fact that some of the classes had very few images to train and test. Although it takes comparatively longer to train deeper versions of YOLOv5 we observe not every quality metric sees an increase. In terms of precision, it is observed that Large has the highest mean precision of over 0.80 and in recall, the Large again performs better than the other networks with a mean of 0.70.



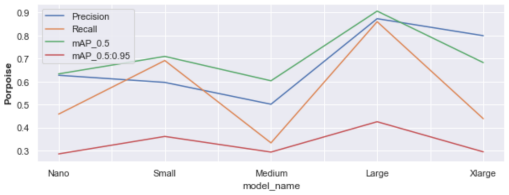
(a) Shark



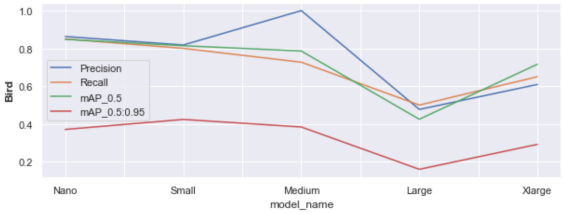
(b) Ray



(c) Seal



(d) Porpoise



(e) Bird

Figure 6.2: Bycatch performance of YOLO configurations over the test dataset

Next, the inference speed, total time taken and non-maximal suppression(nms) speed of each YOLOv5 model configuration is measured by running inference on the test dataset multiple times on an NVIDIA QUADRO RTX 5000[34] graphics card. NMS time is looked at to reduce the false positives produced by the models.

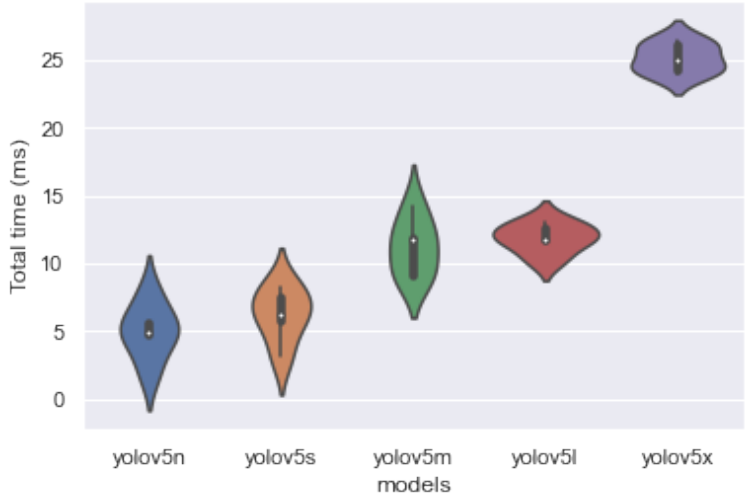


Figure 6.3: Total time in milliseconds

Figure 6.3 shows the total time the YOLO models take to make predictions on the test dataset of 128 images. The YOLOn and YOLOs models have fewer layers and both have a faster total runtime of around 5ms. As the model grows more layers are used to make predictions and as seen the total run time reaches 25ms for the YOLOv5x model.

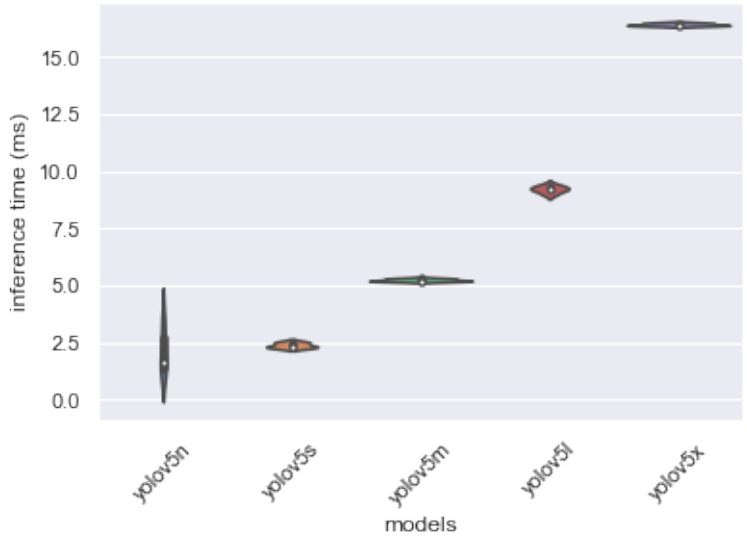


Figure 6.4: Time taken by models to detect all boxes (in milliseconds)

Figure 6.4 shows the inference time taken by the YOLO models to detect all the bounding boxes in a given image. The YOLOn and YOLOs models are performing faster than the deeper configurations.

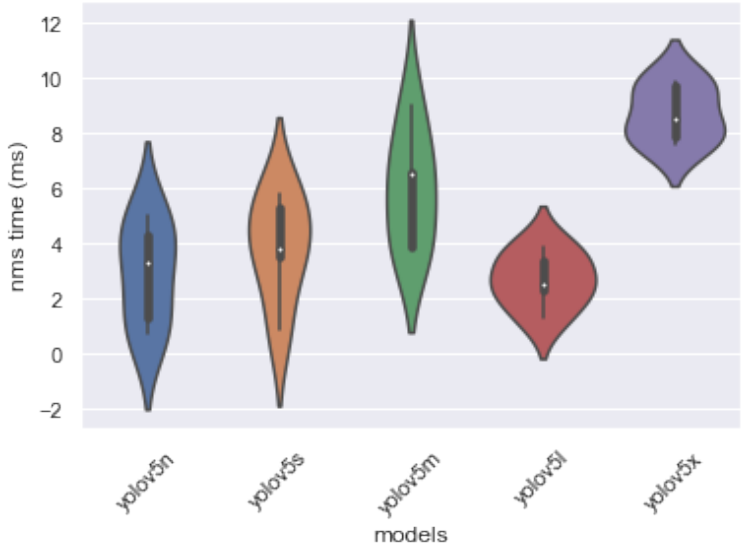
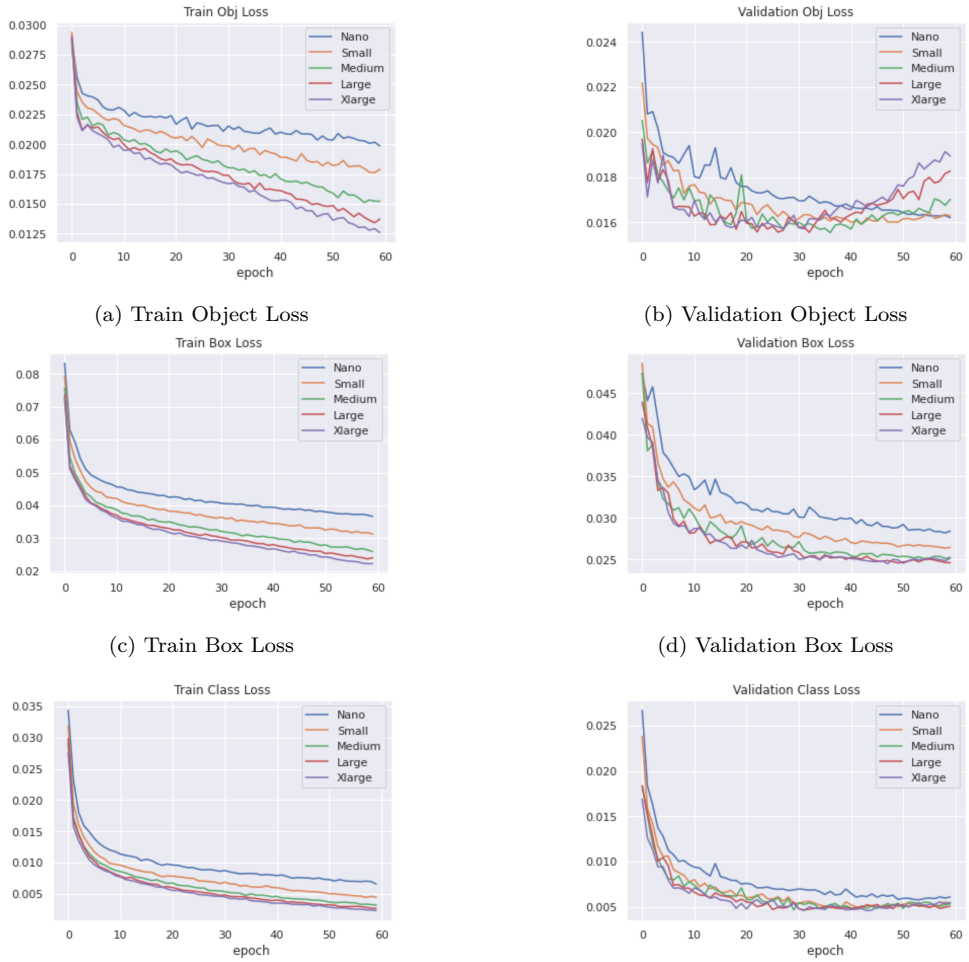


Figure 6.5: Time taken by models to remove unwanted boxes on IoU and Confidence threshold

Figure 6.5 shows the time taken by each YOLOv5 configuration to process the generated inference and remove the unwanted bounding boxes that do not fit into the IoU and confidence threshold set as hyperparameters. It is observed that the YoloV5n and Yolov5s have a mean processing time of 3 and 3.8ms, while the deeper network YOLOv5x has a mean processing time of 8ms.

6.1.2 DTU Model

For the DTU model, the YOLOv5 configuration described in table 5.1 was used. These models were trained on the train data from the split mentioned in table 4.6. Each colored projection represents a YOLOv5 configuration. The training was done for 60 epochs for each configuration, the train and validation losses can be shown in figure 6.6.



(a) Train Object Loss

(b) Validation Object Loss

(c) Train Box Loss

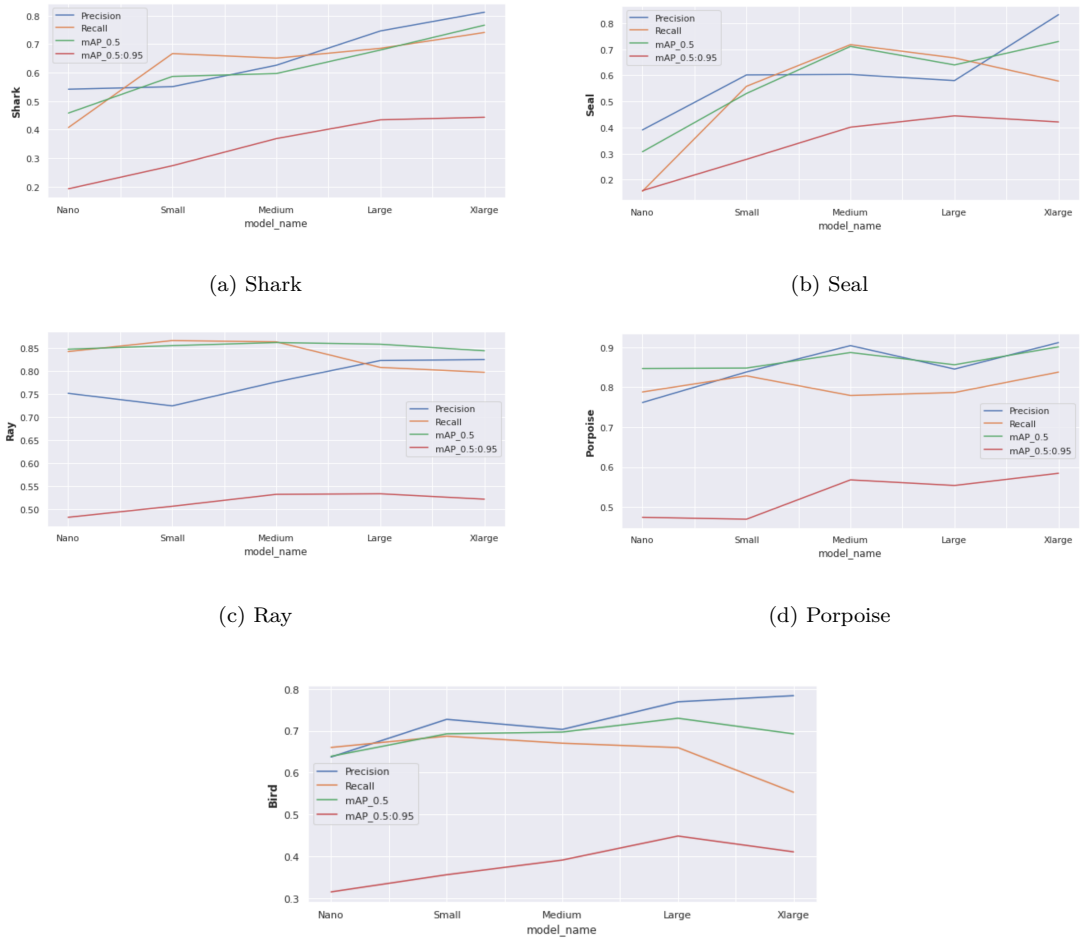
(d) Validation Box Loss

(e) Train Class Loss

(f) Validation Class Loss

Figure 6.6: Validation and Train loss

It can be observed in figure 6.6 that the performance of the model configuration is increasing with the increase in the depth of the model. That is why it can be seen that YOLO Nano is performing the worst. While looking at the validation class loss it can be observed that there is very little over-fitting which is due to having a large dataset. It can also be observed that the Validation Object loss is increasing for the larger model configurations that is YOLO Large and YOLO XLarge, this increase is because the model is predicting the annotations which were not declared in the ground truth.
Now to see how the models performed on the testing dataset, the graphs in figure 6.7 show the metrics on the bycatch classes mentioned in table 4.6. It can be seen that the mAP_0.5 of classes with a larger number of images such as Rays is well for all the model configurations. Comparing all the metrics it can be seen that as the network grows in depth the performance of the detection increases as well. YOLOv5x (XLarge) and YOLOv5l(Large) can be seen as almost identical in the case of performance.



(a) Shark

(b) Seal

(c) Ray

(d) Porpoise

(e) Bird

Figure 6.7: Metrices for DTU Bycatch classes

Figure 6.8 shows the total time different configuration take to run tests on the DTU dataset. YOLOv5n and YOLOv5s have fewer layers and both of them are faster in total run time. As the models grow deeper the total time also increases for making predictions.

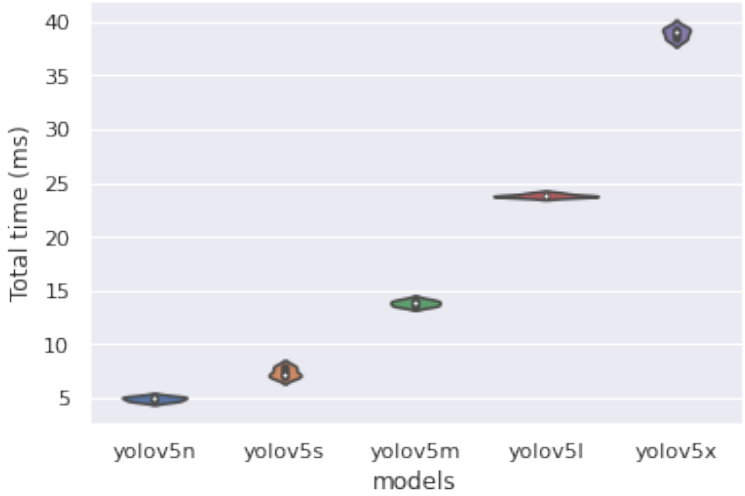


Figure 6.8: Total time in milliseconds

Figure 6.9 shows the inference time taken by the YOLO models on the DTU dataset i.e. detecting the bounding boxes on the test dataset. It can also be observed that YOLOv5n and YOLOv5s models are performing faster than the deeper configurations.

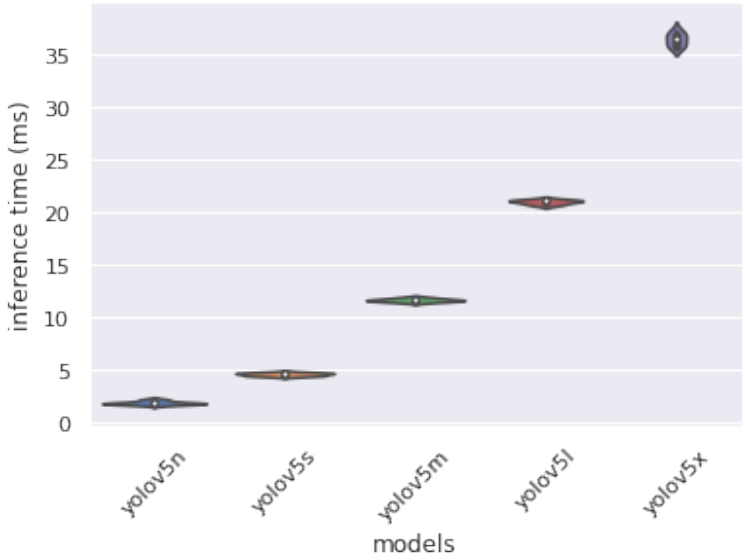


Figure 6.9: Time taken by models to detect all boxes (in milliseconds)

Figure 6.10 shows the time taken by each YOLOv5 configuration to remove the unwanted bounding boxes based on the IoU and confidence threshold.

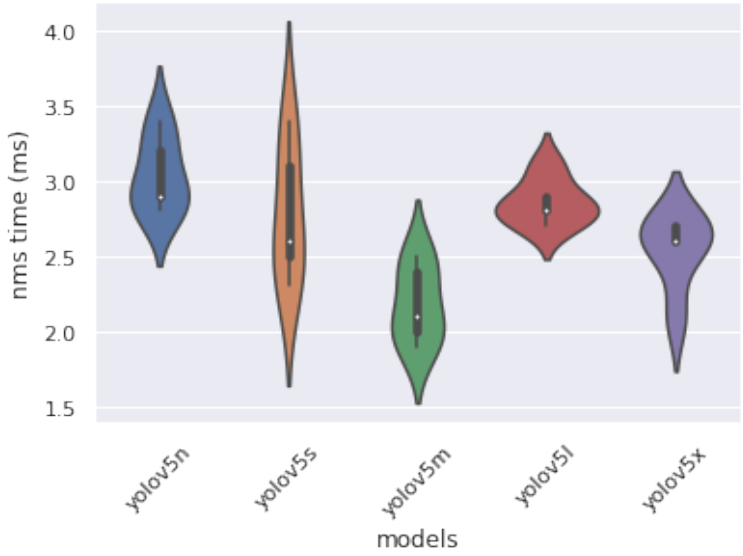


Figure 6.10: Time taken by models to remove unwanted boxes on IoU and Confidence threshold

6.1.3 FEDn MNIST Experiment between DTU and Edge Lab

The FEDn framework setup was tested using the MNIST classification tasks. The dataset was split into two and distributed among the clients. Below is an image showing the two clients connected to a combiner present at the edge lab.

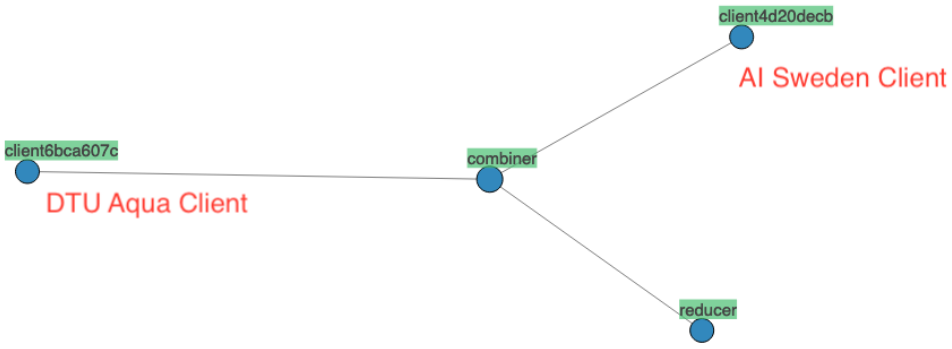


Figure 6.11: The Network map of FEDn MNIST experiment

Once the training is initiated the clients begin training on their share of the data. After every round, the weights are sent back to the combiner which calculates the partial weights and sends them to the reducer. The reducer calculates the FedAvg of the weights and sends them back to the combiner. Once the combiner sends them back to the clients they conclude one round of training.

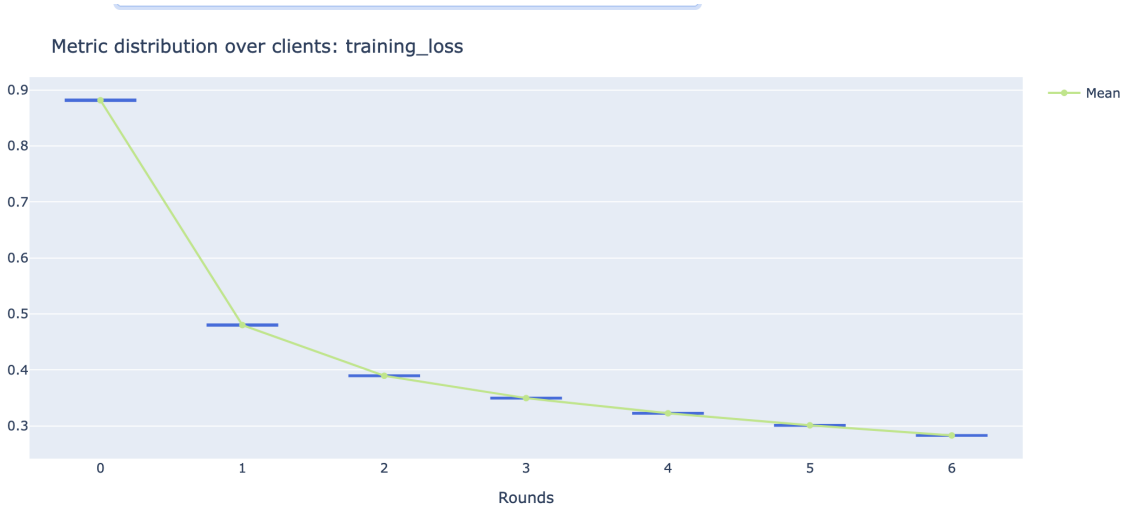


Figure 6.12: Training loss plot of FEDn Mnist

Figure 6.13: Training Accuracy plot of FEDn Mnist

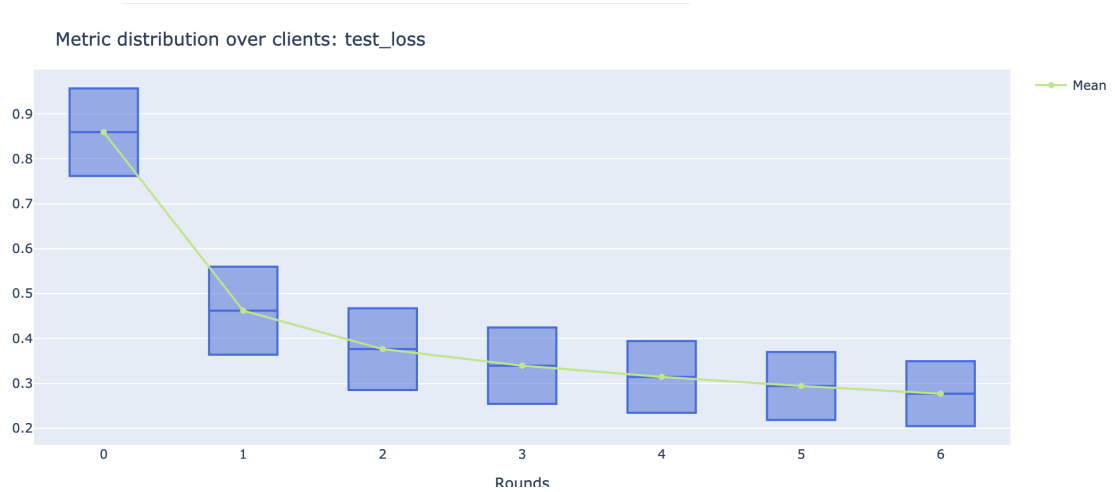


Figure 6.14: Test loss plot of FEDn Mnist

Figure 6.15: Test Accuracy plot of FEDn Mnist

The figures above show how the network improves over just 6 rounds of training in the FEDn framework. Receiving these results proves that a stable connection across two countries to perform federated learning is possible with FEDn framework. It is observed that the model converges and increases in accuracy above 90% in a short duration. This test helped in setting up the actual experiment of performing cross-silo federated learning between DTU and Edge lab that houses SLU dataset.

6.1.4 FEDn YOLOv5 Experiment

Building on the FEDn MNIST experiment, FEDn framework was setup to train a YOLOv5s network on a client (client 1) at Edge lab that has access to the SLU dataset and an external client (client 2) that connects remotely from DTU with their dataset. The training was done for 3 rounds, with each client running 20 epochs per round.

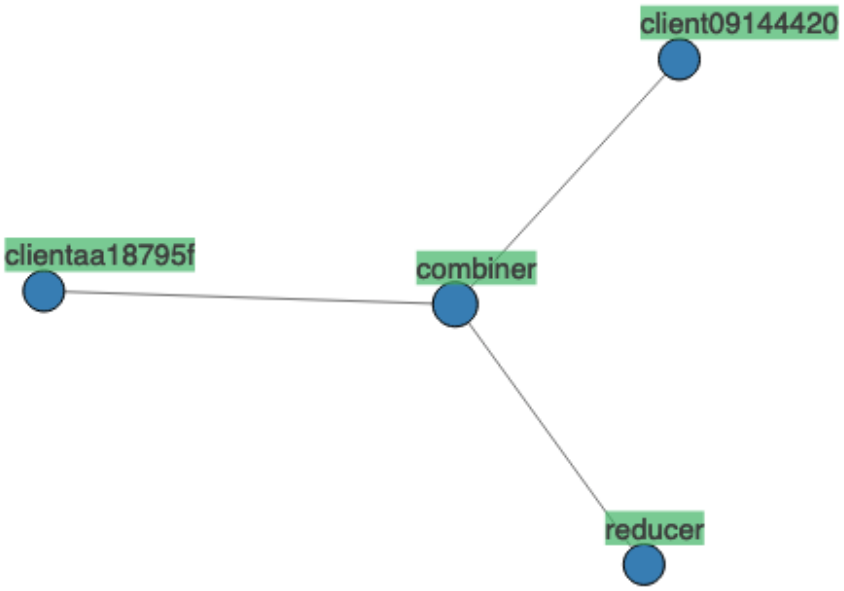
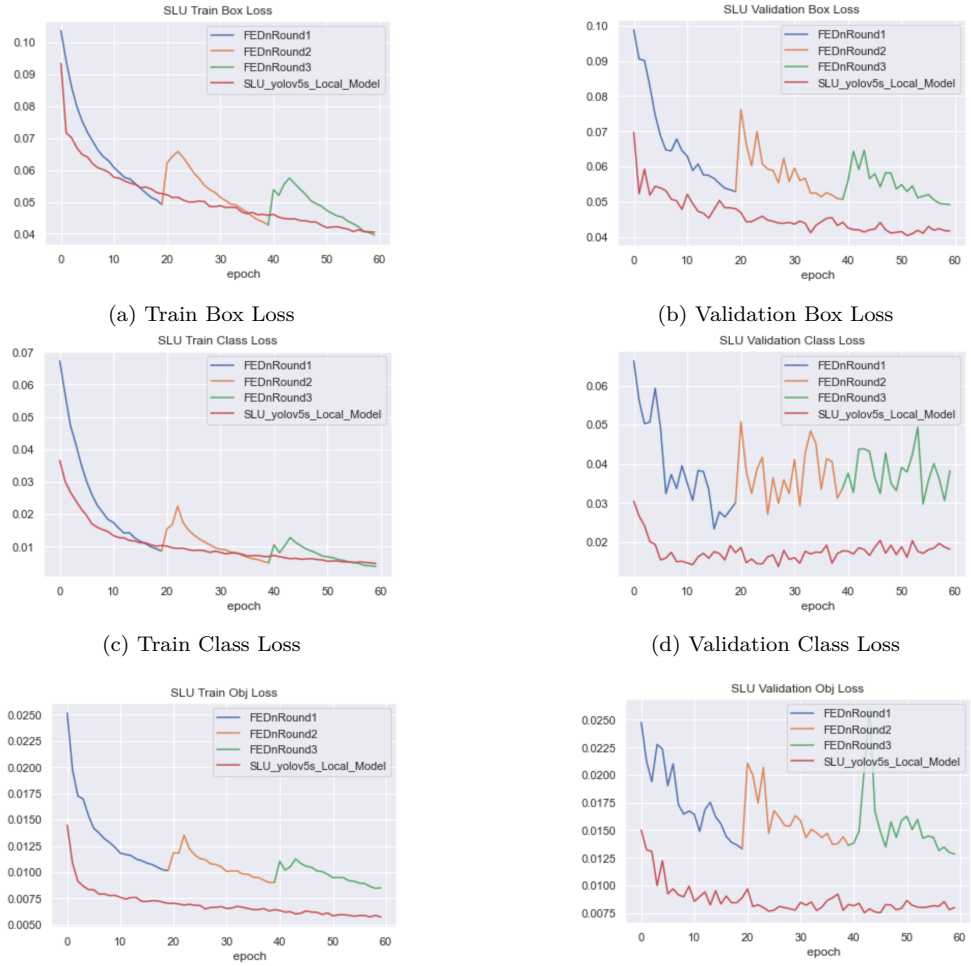


Figure 6.16: FEDn Network Map

Figure 6.16 shows a network map of the FEDn network showing the combination of the two clients to the reducer.

6.1.4.1 Comparison of SLU Local Model and SLU FEDn Client



(a) Train Box Loss

(b) Validation Box Loss

(c) Train Class Loss

(d) Validation Class Loss

(e) Train Object Loss

(f) Validation Object Loss

Figure 6.17: A comparison of Train and Validation losses of SLU Local model and SLU FEDn Client Model

The figure 6.17 shows the graphs of the training and validation loss comparison between the best version of the locally trained YOLOv5s at SLU and the Client YOLOv5s trained on the FEDn network. We observe that the FEDn client reaches the same Train class loss and box loss at the end of 60 epochs. The FEDn sees a sudden rise every 20 epochs as the local weights are sent to the server to be aggregated. The training starts again from the beginning after receiving the updated weights back from the server. The 'Train Obj loss' of the FEDn model never seems to meet with the local model at the end of training. The local model seems to be much better at all the validation loss metrics. It is also observed that the FEDn model seems to start overfitting from Epoch 20 (round 1 of weights update). The local model seems to see a stable decrease in the other two validation losses as well.

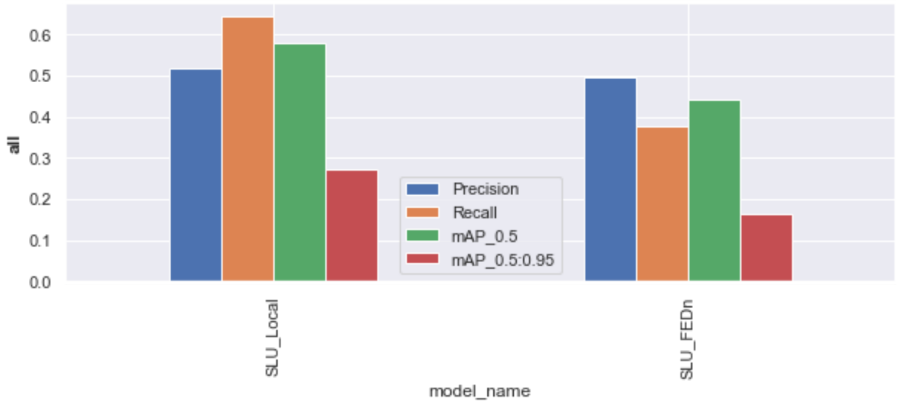
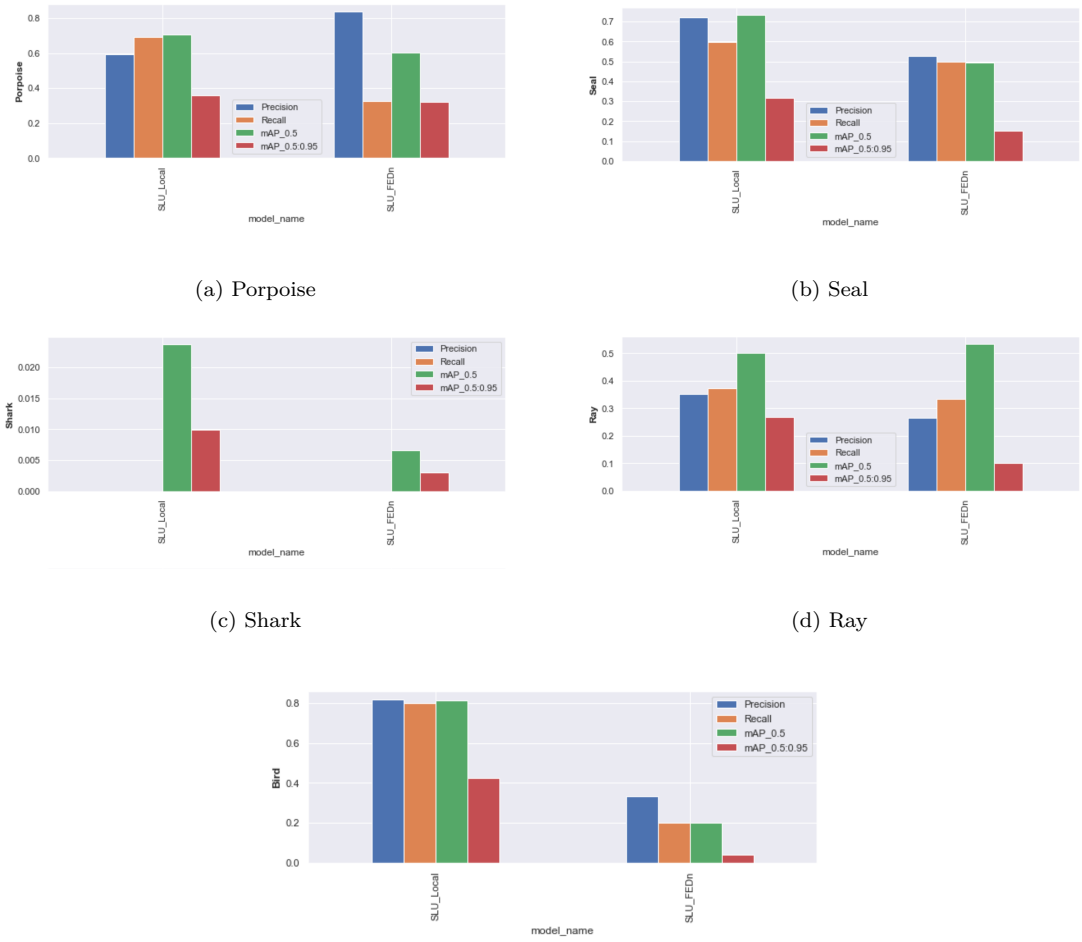


Figure 6.18: A comparison of overall quality metrics of SLU Local model and SLU FEDn Client Model on the test dataset

Figure 6.18 shows the overall quality metrics comparison of the local and FEDn client models over the test dataset. We observe that the local model scores are comparatively higher than the FEDn client model. The overall precision seems to be in the same range of 0.5 but the recall value seems to see a drastic drop from over 0.62 for the local model to below 0.4 for the FEDn client. The mean average precision values also see a decrease in the FEDn client. The overall performance of the locally trained YOLOv5s model seems to be better than the YOLOv5s model trained on the client of the FEDn framework.



(a) Porpoise

(b) Seal

(c) Shark

(d) Ray

(e) Bird

Figure 6.19: A comparison of Train and Validation losses of SLU Local model and SLU FEDn Client Model

Figure 6.19 shows the performance of the locally trained YOLOv5s model and the YOLOv5s model trained on the client of the FEDn framework. As observed in the overall metrics graphs, it is very clear that the local model has better metrics compared to the model on the FEDn client. Interestingly the precision of the Porpoise class sees an increase in the FEDn client with a high of 0.82 while the locally trained model reached a precision of 0.6.

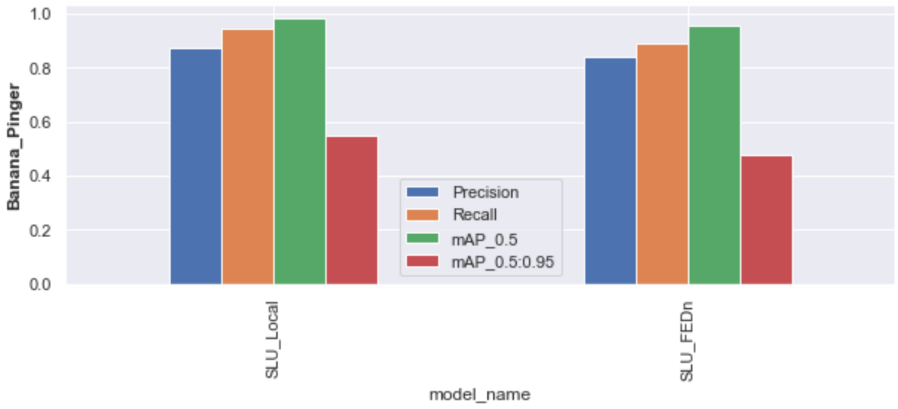
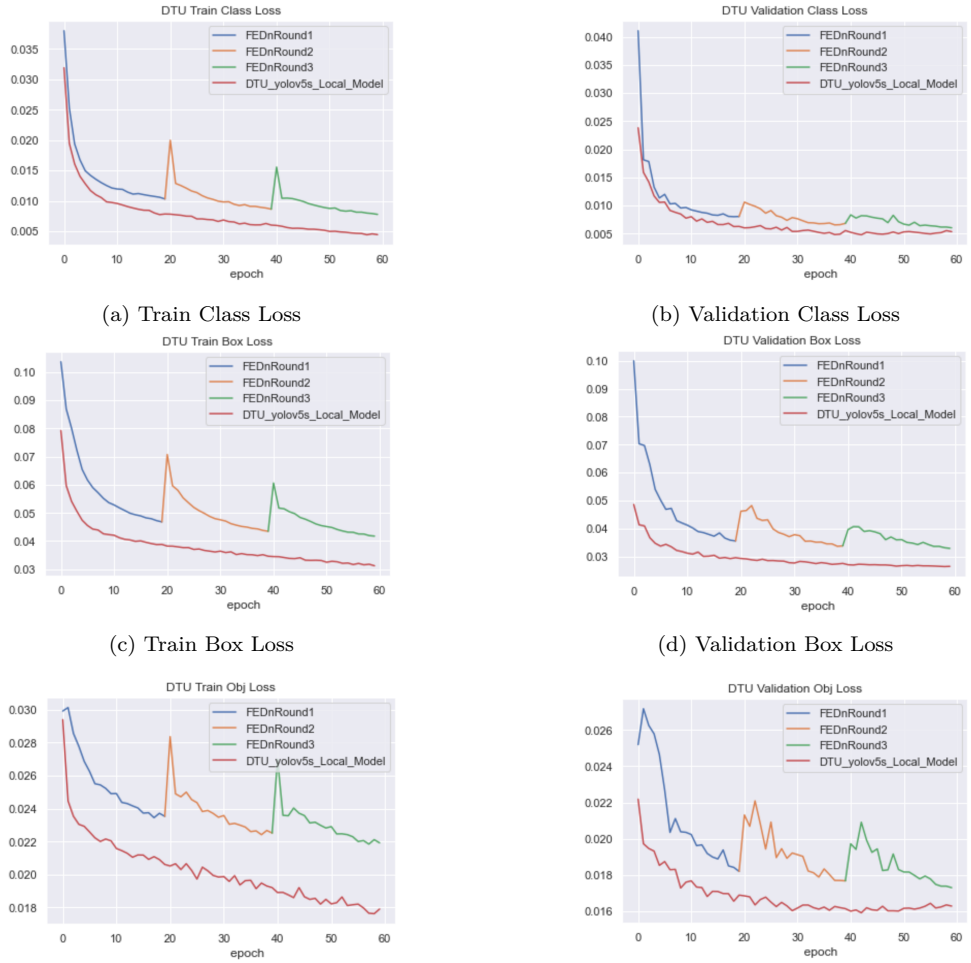


Figure 6.20: A comparison of Validation metrics of SLU Local model and SLU FEDn Client Model for the class banana pingers

As seen in the figure 6.20 it is interesting to observe that classes like 'Banana pinger' performance sees an equally impressive metric value on the FEDn network. The class is very unique in its visual features and the data is only present in the SLU dataset. Hence it does not seem to have any significant change in its quality metrics which are still above 0.8 in its precision, recall, and average precision0.5 metrics.

6.1.4.2 Comparison of DTU Local Model and DTU FEDn Client



(a) Train Class Loss

(b) Validation Class Loss

(c) Train Box Loss

(d) Validation Box Loss

(e) Train Object Loss

(f) Validation Object Loss

Figure 6.21: A comparison of Train and Validation losses of DTU Local model and DTU FEDn Client Model

Figure 6.21 shows the graphs of the training and validation loss comparison between the best version of the locally trained YOLOv5s on the DTU dataset and the FEDn client. When the FEDn client is observed it can be seen that the Train class loss and box loss do not reach the same loss as the local model at the end of 60 epochs. It can be seen that the FEDn sees a sudden rise every 20 epochs that is because the weights are sent to the server to be aggregated and sent back. It can be seen that the local model overall performs better than the FEDn model. Seeing the validation losses it can be observed that both the models almost perform the same on the validation dataset.

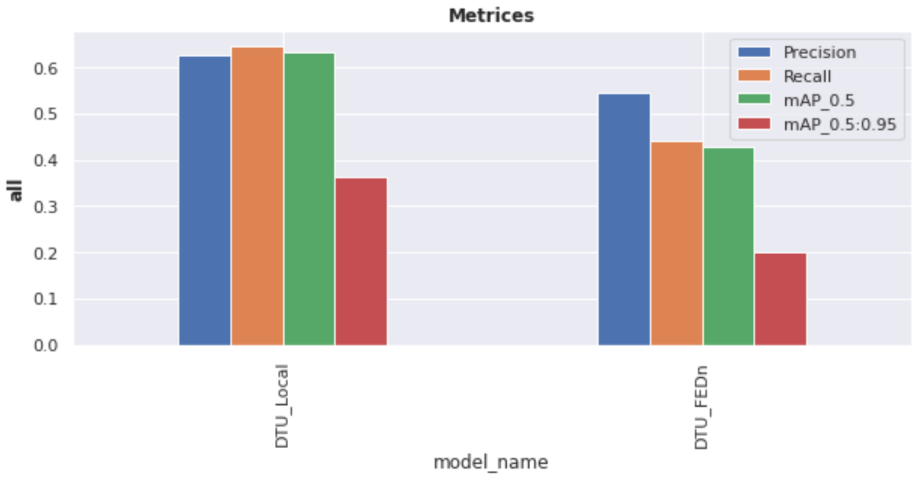
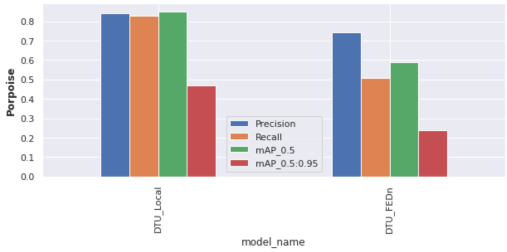
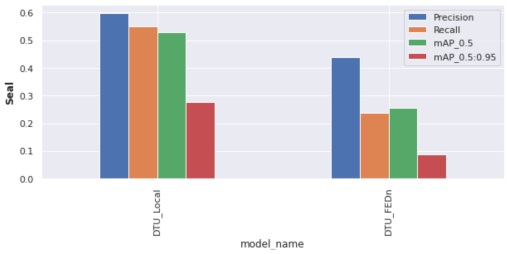


Figure 6.22: A comparison of overall quality metrics of DTU Local model and DTU FEDn Client Model on the test dataset

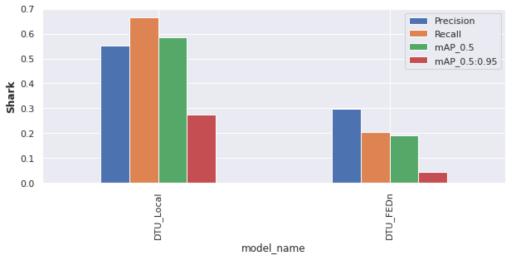
Figure 6.22 shows the overall quality metrics comparison of the local model and the FEDn model on the DTU test dataset. It can be observed that the Local model performs better than the FEDn model.



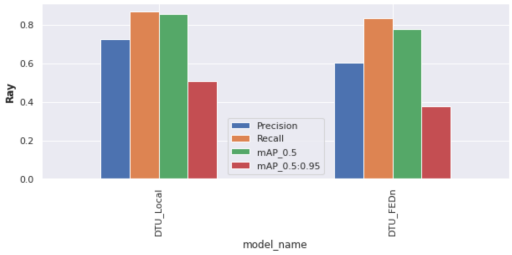
(a) Porpoise



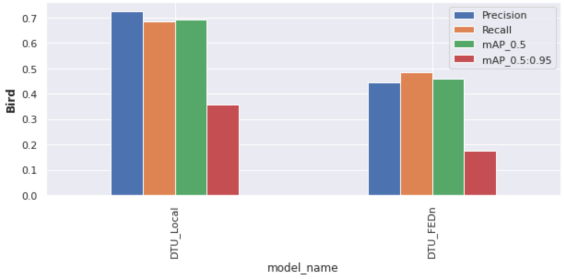
(b) Seal



(c) Shark



(d) Ray



(e) Bird

Figure 6.23: A comparison of validation metrics of DTU Local model and DTU FEDn Client Model for the bycatch classes

The figure 6.23 shows that there is a difference between the performances of the two models on the bycatch classes, FEDn models performance on the Ray class can be seen a bit close to the local model's performance that is because there is a large number of annotations for the ray class.

7

Discussion

This chapter contains discussions on the results drawn from the YOLO models on both the datasets and inferences from the federated learning experiment.

7.1 SLU Model

The predictions generated by the best-tuned model for the given dataset the YOLOv5s model is discussed in figure 7.1.

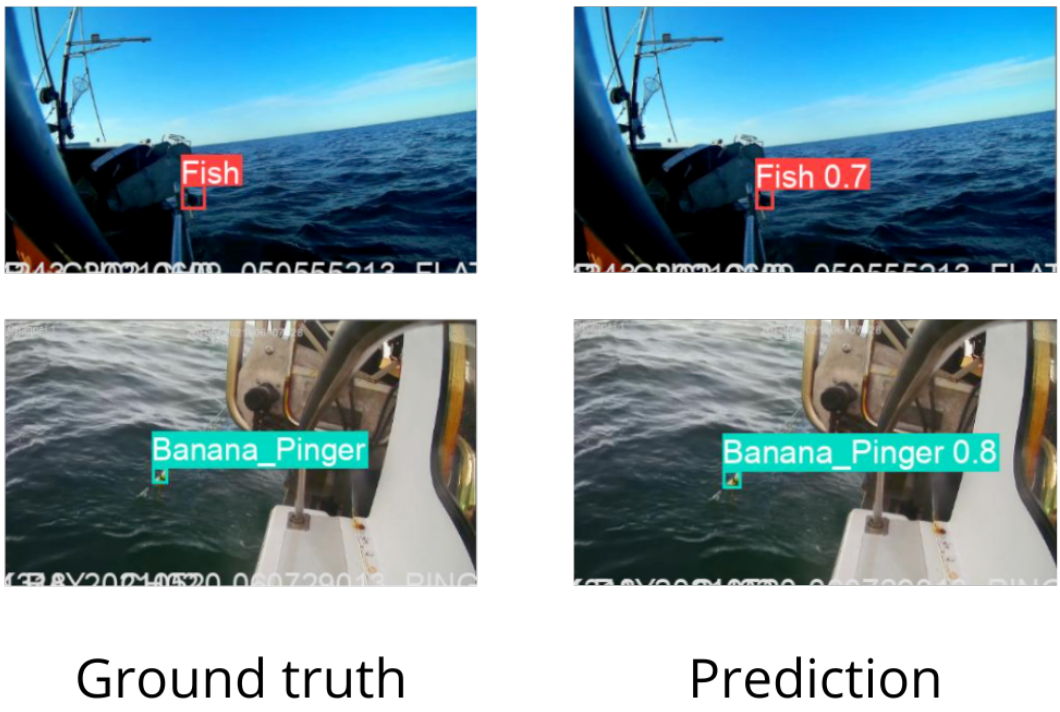


Figure 7.1: SLU Common Class Predictions

Figure 7.1 shows how the YOLOv5s model performs in predicting even in a harder test image, for example the class 'Fish' is being predicted with reasonably high confidence. This is a result of the inclusion of background images with no objects so the model learns how the environment looks and introducing augmentation to the dataset to simulate extreme weather conditions and different times of the day.

On the other hand classes like 'Banana' with distinctive features have always been detected with ease, the YOLOv5s model generated a confidence score of 1 in some cases. This shows that the more distinct a class looks the easier it is for the object detection model to learn what it looks like even under hard images like the one shown in figure 7.1.

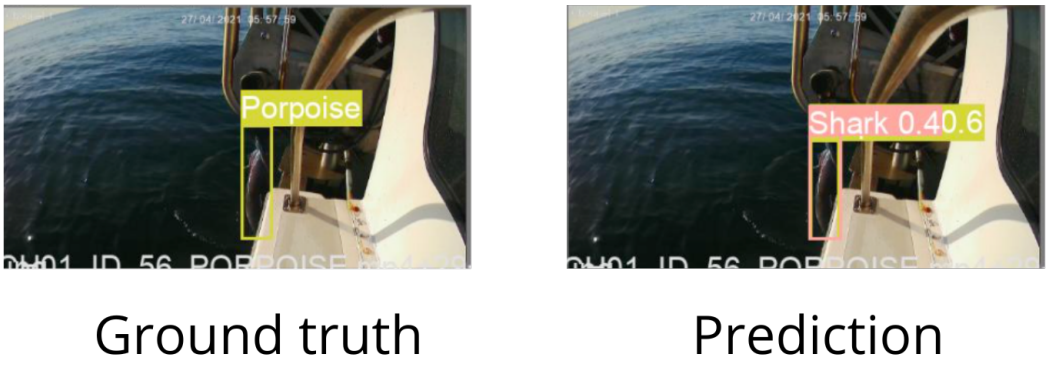


Figure 7.2: Miss Classification of SLU test set

Some interesting predictions were generated by the model. The visual features of some of the frames of sharks and porpoises are very similar to one another, for example, the color of their skin especially when they are partially visible as in figure 7.2 even the best performing model identifies such an object to be of both the classes i.e. Porpoise and a shark.

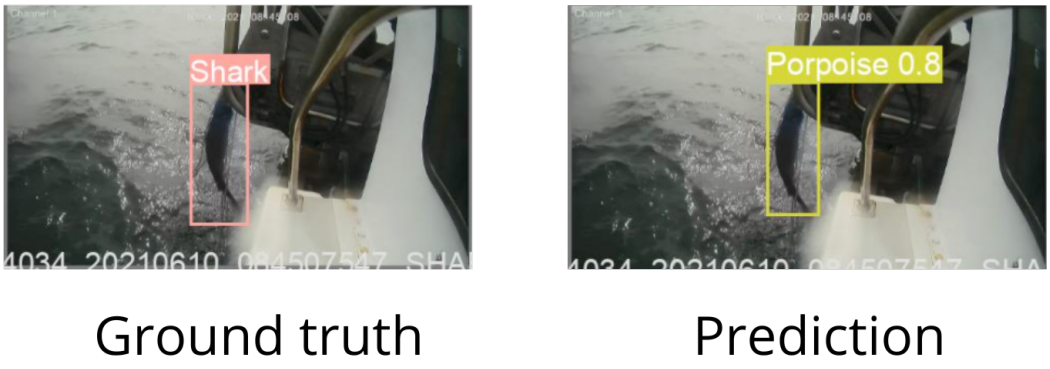


Figure 7.3: Miss Classification of SLU test set

Another scenario of these two classes (porpoises and sharks) is the amount of data that was used to train the models. The number of porpoise frames is large compared to the sharks and the network seems to be biased towards the Porpoises in general.

7.2 DTU Model

When the results were produced it was observed that some classes were being confused by the models. An example is shown in fig 7.4[1]. where it can be seen that the model confuses bird as a fish.

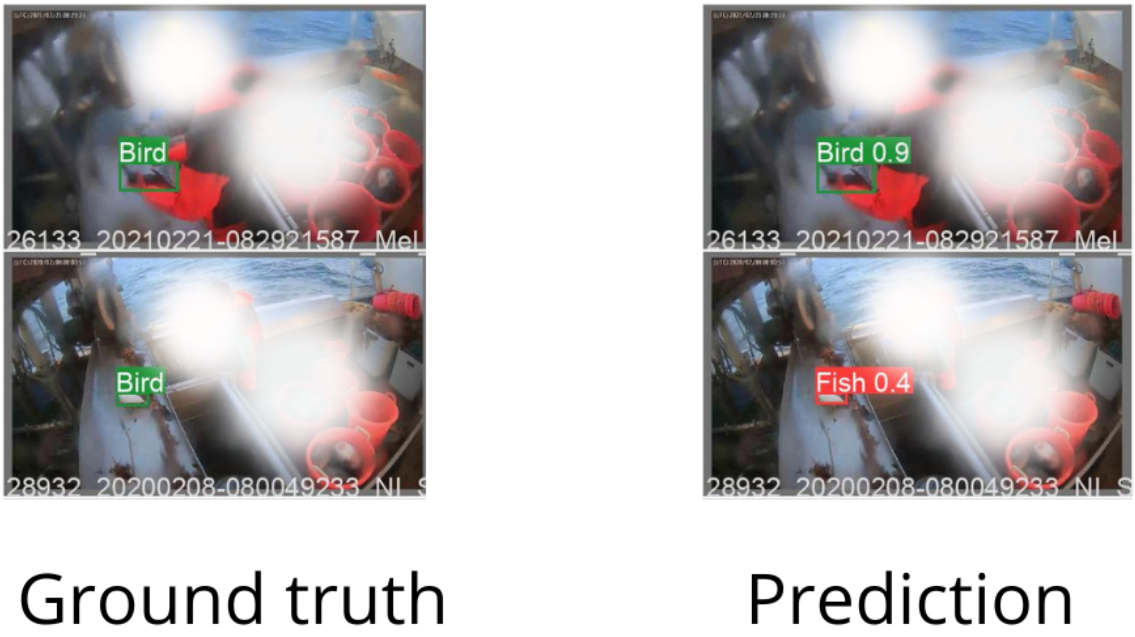


Figure 7.4: Miss Classification of DTU Test Set

In another example in figure 7.5 it can be seen that the model confuses a fish as a ray.

[1]The images in figure 7.4 do not have the white blur, they have been altered to hide the fishermen in the images.

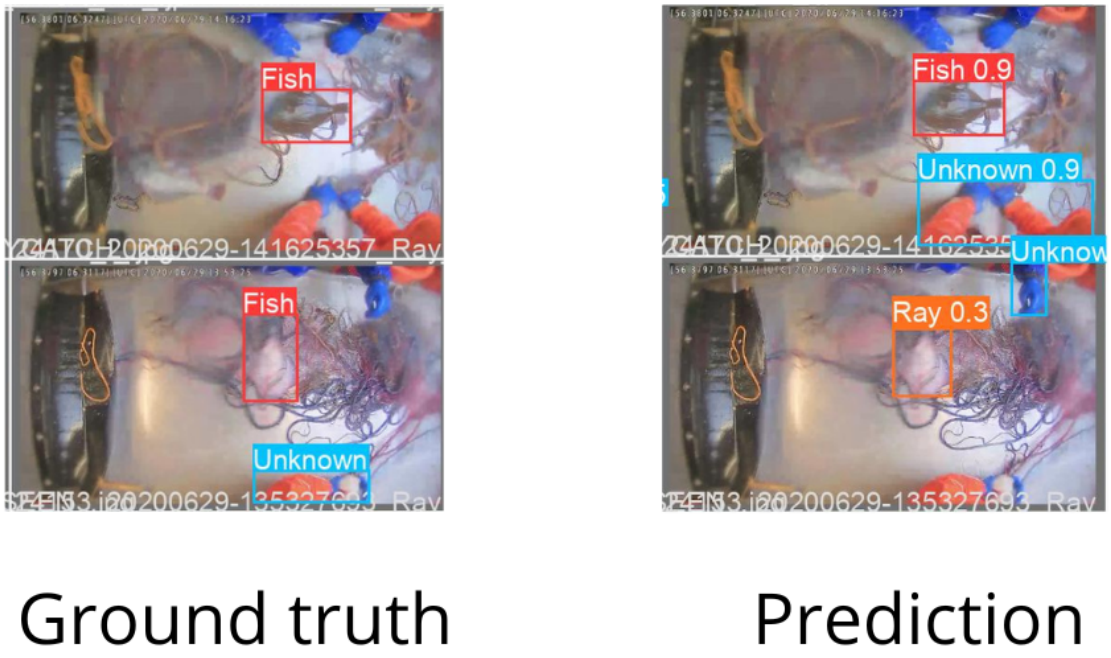


Figure 7.5: Miss Classification of DTU Test Set

These confusions were a product of imbalanced data i.e. rays having a count of almost 5110 annotations with respect to fish which only had 1135 annotations in the DTU dataset. The same is the case of birds being confused with fish because birds have fewer data i.e. 820 annotations. There is another reason why this miss classification is observed is because in the fish class there is a fish species called the flatfish which looks like rays. Which also confuses the model in detecting.

7.3 Discrepancies in data annotations

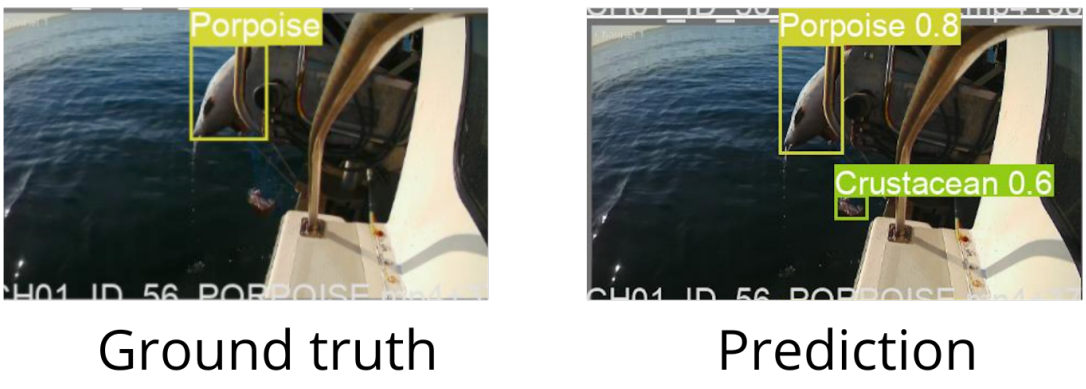


Figure 7.6: Missed Annotations on SLU dataset

Figure 7.6 shows the ground truth with a missed annotation of the class Crustacean in the SLU dataset and the model predicting it with reasonable confidence. This affects the precision and recall metrics of the model even when it makes the right prediction. The class-label consistency is a key factor and requires all instances of a class to be labeled. It can also be seen in the DTU dataset that in the ground truth some classes are not labeled as seen in the figure 7.7, but when the model is running the validation it predicts the classes.

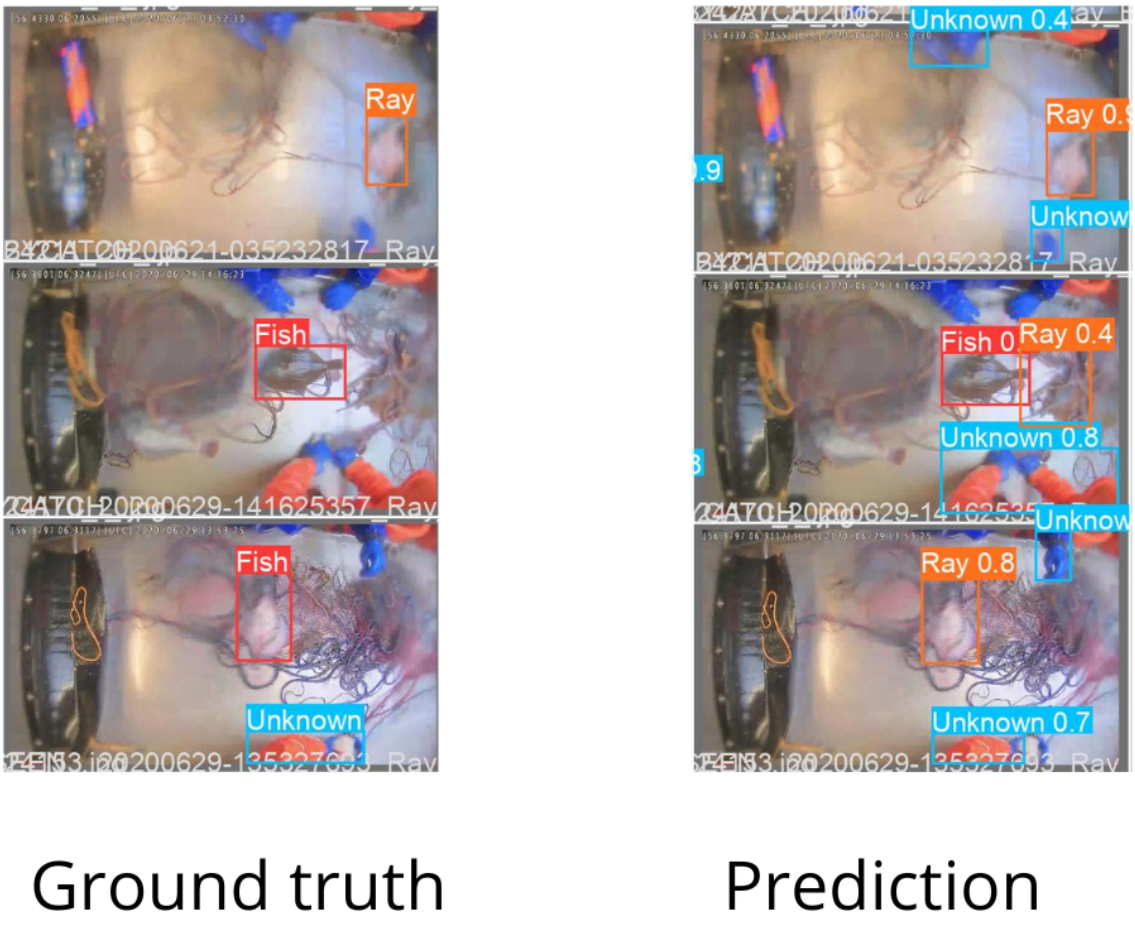


Figure 7.7: Miss match of prediction to ground truth

Figure 7.8: An ambiguous-looking object caught in the gillnet and was labeled as a fish. Also shows the labeling accuracy of the annotations.

The datasets(SLU and DTU) were annotated with the help of experts with domain knowledge, but there is still room for human error, and annotating images comes down to the annotator's perspective if a given object in the frame is relevant or if it's even the right class. These discrepancies affect the model's performance to a great extent when the missing annotation of objects or miss classifying them increases in number. Figure 7.8 shows an image where the object is hard to be guessed. When two different annotators were asked to annotate the image, one picked the class 'Fish' and the other picked the class 'Bird'. It is hard to make out when class-specific features like 'beak' are not visible in the given image even though it is a bird, it can also be seen that the bounding box is not tight enough (label accuracy) to the object as well. All these factors into the model's performance in predicting the right class and also the size of the bounding boxes.

7.4 General Reflection on SLU and DTU Models

Object detection models like YOLOv5 work best when the dataset has at least 1500 images and over 10000 annotations per class. The model should also be exposed to deployment environments (different times of the day and angles of the camera). It is fair to say that to assess such an object detection model with 13 classes, a balanced dataset is required. It is hard to collect data on classes that occur very rarely and will need a lot more effort in gathering the data required and having them annotated. The quality of the dataset is also a major concern as discussed earlier. One solution would be to collaborate with stakeholders with similar interests in gathering such datasets and allocating resources to do quality checks over them at least in the initial stages. The number of classes needs to be reassessed and classes with similar features could be clubbed together for better detection.

7.5 FEDn YOLOv5 Experiment

Running the FEDn YOLOv5 experiment, the requirements starting from the hardware are drastically different from the MNIST experiment. The wait time had to be fine-tuned to sync the client model training. Client 1 at the Edge lab was on the same network and had access to reasonably faster storage and it ran each round (20 epochs) in under 20 minutes, whereas client 2 housed at DTU had to communicate over a VPN connection and even with a powerful GPU it took about 45 minutes to complete one round of training. If the window to share the partial weights is missed by one of the clients then it waits until the next sync. The combiner waits until the time-out duration mentioned to see if the client qualifies for a weights sync during that round. Syncing both the clients proved challenging as the connection over the VPN was unstable and had the possibility of disconnecting. FEDn is equipped to handle a scenario where a client becomes offline, but in the current setup with just two clients it is crucial to have a reliable link between the clients and the server during syncs at least. The class imbalance in the datasets greatly impacts federated learning and needs to be addressed. The number of local epochs was 20 epochs and the number of rounds was 3, at the end of training the model has not yet converged and showed signs of overfitting at SLU's client. The chosen no of local epochs and the number of rounds decide the convergence of the model and the time taken. It is seen that the model needs more than 3 rounds to converge as the dataset is varied and considerably large for both the clients. It is important that the features of the classes they learn during the federated learning can be used on their local data for prediction. It was discussed that the quality and the quantity of the datasets vary drastically between both clients. This reflects the results that were observed when the weights learned over the federated network were tested on local test datasets.

8

Conclusion

In this study, the feasibility of applied machine learning in the field of marine life conservation was explored, in particular the use of object detection and classification models to help identify bycatches on gillnet fishing video feed. Two different datasets were explored, one built by us from the video data from SLU and one provided by DTU. It is observed that both the models were trained on imbalanced datasets and are biased towards only a few of the 13 classes. In the DTU model, it is observed that the Ray and Fish classes were confused by the model because of the visual similarities between them, while the same occurs with porpoises and sharks in the SLU model. It was interesting to notice that the class 'Banana Pinger' with only a few hundred images was easily detected with high precision and recall by the SLU model.

The primary goal of this study was to identify the five bycatch classes, which were porpoise, seal, shark, birds, and rays. Machine learning techniques such as federated learning were tried to see if the model's performances improved. While the models were trained as clients on a federated framework, the best-tuned model still underperformed. It is observed that the class imbalance affects the weights that are synced every round and takes longer than usual to converge. Keeping the clients synced between every round also proved challenging as one client was present on the same network and location as the server, while the other client, being hundreds of miles away, had to connect to the server over a VPN connection.

For further improvement in the performance of the models, it would be fair to rethink the hierarchical FedAvg aggregation function used in the FEDn framework with a function that better accounts for the imbalance in the data while calculating aggregated weights. The syncing of the clients also needs more time to find the right fit for the current setup. The model would also benefit from introducing data for underrepresented classes that have fewer data points. Out of the 13 classes in the SLU dataset, only fish and porpoise have more than 600 images to train. Even then, the fish class is comprised of different common fish species clubbed together into a single class with a majority of them being from one or two of the species that are caught(cod and flat fishes). It is worth the time and resources to collect data on the individual species so that the model learns to differentiate them.

Overall, the study indicates that applied machine learning can be used in the study of bycatches and, when done with a quality dataset, will lead to promising results. Adapting the federated framework is complex, but when tuned for the current setup, it could leverage a varied dataset from the clients while leaving room for collaborators to join in the future.

Bibliography

[1] Aloysius T.M. van Helmond et al. "Electronic monitoring in fisheries: Lessons from global experiences and future opportunities". In: *Fish and Fisheries* 21.1 (2020), pp. 162–189. DOI: https://doi.org/10.1111/faf.12425. eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1111/faf.12425. URL: https://onlinelibrary.wiley.com/doi/abs/10.1111/faf.12425.

[2] Aliki Panagopoulou et al. "Caught in the Same Net? Small-Scale Fishermen's Perceptions of Fisheries Interactions with Sea Turtles and Other Protected Species". In: *Frontiers in Marine Science* 4 (2017). ISSN: 2296-7745. DOI: 10.3389/fmars.2017.00180. URL: https://www.frontiersin.org/article/10.3389/fmars.2017.00180.

[3] Ratana Chuenpagdee et al. *Bottom-up, global estimates of small-scale marine fisheries catches*. 2006. DOI: http://dx.doi.org/10.14288/1.0074761.

[4] Gildas Glemarec, Sara Königson, and Lotte Kindt-Larsen. *Bycatch in Baltic Sea commercial fisheries: High-risk areas and evaluation of measures to reduce bycatch*. English. HELCOM, 2021.

[5] Aloysius T.M. van Helmond et al. "Electronic monitoring in fisheries: Lessons from global experiences and future opportunities". In: *Fish and Fisheries* 21.1 (2020), pp. 162–189. DOI: https://doi.org/10.1111/faf.12425. eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1111/faf.12425. URL: https://onlinelibrary.wiley.com/doi/abs/10.1111/faf.12425.

[6] Lotte Kindt-Larsen et al. "Observing incidental harbour porpoise Phocoena phocoena bycatch by remote electronic monitoring". In: *Endangered Species Research* 19 (Nov. 2012), pp. 75–83. DOI: 10.3354/esr00455.

[7] Gildas Glemarec et al. "Assessing seabird bycatch in gillnet fisheries using electronic monitoring". In: *Biological Conservation* 243 (2020), p. 108461. ISSN: 0006-3207. DOI: https://doi.org/10.1016/j.biocon.2020.108461. URL: https://www.sciencedirect.com/science/article/pii/S0006320719313916.

[8] *Meet the winners of the N+1 Fish, N+2 Fish challenge*. Dec. 19, 2017. URL: https://www.drivendata.co/blog/fish-winners/.

[9] Olaf Ronneberger. *U-Net: Convolutional Networks for Biomedical Image Segmentation*. May 18, 2015. URL: https://arxiv.org/abs/1505.04597.

[10] Jonathan Hui. *SSD object detection: Single Shot MultiBox Detector for real-time processing*. Dec. 15, 2020. URL: https://jonathan-hui.medium.com/ssd-object-detection-single-shot-multibox-detector-for-real-time-processing-9bd8deac0e06.

[11] M. Rizwan Khokher et al. “Early lessons in deploying cameras and artificial intelligence technology for fisheries catch monitoring: where machine learning meets commercial fishing”. In: *Canadian Journal of Fisheries and Aquatic Sciences* 0.0 (0), pp. 1–10. DOI: 10.1139/cjfas-2020-0446. eprint: https://doi.org/10.1139/cjfas-2020-0446. URL: https://doi.org/10.1139/cjfas-2020-0446.

[12] Laura Mannocci et al. “Machine learning to detect bycatch risk: Novel application to echosounder buoys data in tuna purse seine fisheries”. In: *Biological Conservation* 255 (Feb. 2021). DOI: 10.1016/j.biocon.2021.109004.

[13] Richard. *Deep learning methods applied to electronic monitoring data: automated catch event detection for longline fishing | Bycatch Management Information System (BMIS)*. 2021. URL: https://www.bmis-bycatch.org/references/dmye6wq8.

[14] Yann LeCun et al. “Object recognition with gradient-based learning”. In: *Shape, contour and grouping in computer vision*. Springer, 1999, pp. 319–345.

[15] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. “Imagenet classification with deep convolutional neural networks”. In: *Advances in neural information processing systems* 25 (2012).

[16] Olga Russakovsky et al. “Imagenet large scale visual recognition challenge”. In: *International journal of computer vision* 115.3 (2015), pp. 211–252.

[17] Kaiming He et al. “Deep residual learning for image recognition”. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016, pp. 770–778.

[18] Joseph Redmon et al. “You only look once: Unified, real-time object detection”. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016, pp. 779–788.

[19] Ross Girshick et al. “Rich feature hierarchies for accurate object detection and semantic segmentation”. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2014, pp. 580–587.

[20] Bandyopadhyay Hmrishav. *YOLO: Real-Time Object Detection Explained*. May 26, 2022. URL: https://www.v7labs.com/blog/yolo-object-detection.

[21] Richardson Santiago Teles de Menezes, Rafael Marrocos Magalhaes, and Helton Maia. “Object Recognition Using Convolutional Neural Networks”. In: *Recent Trends in Artificial Neural Networks*. Ed. by Ali Sadollah and Carlos M. Travieso-Gonzalez. Rijeka: IntechOpen, 2019. Chap. 5. DOI: 10.5772/intechopen.89726. URL: https://doi.org/10.5772/intechopen.89726.

[22] Gutta Surya. *Object Detection Algorithm—YOLO v5 Architecture*. Jan. 6, 2022. URL: https://medium.com/analytics-vidhya/object-detection-algorithm-yolo-v5-architecture-89e0a35472ef.

[23] Adrian Rosebrock. *Intersection over Union (IoU) for object detection*. Apr. 30, 2022. URL: https://pyimagesearch.com/2016/11/07/intersection-over-union-iou-for-object-detection/.

[24] *What is Classification Threshold*. Aug. 5, 2021. URL: https://deepchecks.com/glossary/classification-threshold/#:%5C%7E:text=The%5C%

20threshold%5C%20governs%5C%20the%5C%20choice,set%5C%20to%5C%200.5%5C%20by%5C%20default..

[25] Ahmed Fawzy Gad. *Mean Average Precision (mAP) Explained*. Apr. 9, 2021. URL: https://blog.paperspace.com/mean-average-precision/.

[26] Brendan McMahan et al. "Communication-efficient learning of deep networks from decentralized data". In: *Artificial intelligence and statistics*. PMLR. 2017, pp. 1273–1282.

[27] Jakub Konečn et al. "Federated optimization: Distributed machine learning for on-device intelligence". In: *arXiv preprint arXiv:1610.02527* (2016).

[28] Peter Kairouz et al. "Advances and open problems in federated learning". In: *Foundations and Trends® in Machine Learning* 14.1–2 (2021), pp. 1–210.

[29] Morgan Ekmefjord et al. "Scalable federated machine learning with FEDn". In: *arXiv preprint arXiv:2103.00148* (2021).

[30] Piotr Skalski. *Make Sense*. https://github.com/SkalskiP/make-sense/. 2019.

[31] A. Dutta, A. Gupta, and A. Zissermann. *VGG Image Annotator (VIA)*. Version: X.Y.Z, Accessed: INSERT$_DATE_HERE$. 2016. URL: http://www.robots.ox.ac.uk/~vgg/software/via/.

[32] Alexander Buslaev et al. "Albumentations: fast and flexible image augmentations". In: *Information* 11.2 (2020), p. 125.

[33] Anchor Labs. *BlackBox Analyzer*. URL: http://www.anchorlab.dk/Analyzer.aspx#: (visited on 05/12/2022).

[34] *NVIDIA QUADRO RTX 5000 16GB*. 2019. URL: https://www.nvidia.com/content/dam/en-zz/Solutions/design-visualization/quadro-product-literature/quadro-rtx-5000-data-sheet-us-nvidia-704120-r4-web.pdf.

[35] *YOLO Darknet TXT Annotation Format*. URL: https://roboflow.com/formats/yolo-darknet-txt#:%5C%7E:text=YOLO%5C%20Darknet%5C%20TXT%5C%20Annotation%5C%20Format&text=This%5C%20format%5C%20contains%5C%20one%5C%20text,IDs%5C%20to%5C%20human%5C%20readable%5C%20strings. (visited on 05/12/2022).

[36] *NVIDIA TESLA T4 TENSOR CORE 16GB*. 2019. URL: https://www.nvidia.com/content/dam/en-zz/Solutions/Data-Center/tesla-t4/t4-tensor-core-product-brief.pdf.

Department Of Computer Science and Engineering
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden
www.chalmers.se

CHALMERS
UNIVERSITY OF TECHNOLOGY