ENDG 511
Industrial Internet of Things
Systems and Data Analytics

Final Project Report

2024-04-10

John Santos (30087188)
Kushal Gadhiya (30094278)
Irfan Khan (30095760)

Table of Contents

Abstract

Project Motivation

The motivation of this project is to continue the work behind Radar Object Detection Networks (RODNet: A Real-Time Radar Object Detection Network Cross-Supervised by Camera-Radar Fused Object 3D Localization, 2021) which sought to enhance conventional object detection networks which relied on camera-based sensing and at the same time explore concepts that was provided by the course such as *FastAI* and *TinyAI*.

The decision behind ***FastAI*** is to achieve real-time model inferences of trained neural networks that is crucial for fast-paced conditions such as road and traffic awareness in automated driving applications.

The decision behind ***TinyAI*** is to achieve model sizes that is suitable for low-power and limited processing/memory devices such as a Raspberry Pi which is crucial for applications that sought for low cost and light devices that can be easily integrated into vehicles for automated driving applications.

Project Evolution

The initial intention moving forward with the project was to focus only on radar-based object detection to apply pruning, quantization, and clustering for *TinyAI* and multibranch architectures for *FastAI*. However, after facing some challenges with pruning and quantization in PyTorch (*PyTorch*, n.d.) and as well as time constraints, the intention shifted towards areas that were more familiar such as camera-based object detection libraries such as TensorFlow (*TensorFlow*, n.d.) and YoloV5 (*YoloV5* n.d.) to apply pruning and quantization.

The project scope thus evolved to compare radar-based object detection implemented in PyTorch and camera-based object detection implemented in TensorFlow and YoloV5 to address the advantages of using radar-based object detection in obscured scenes where a camera-based object detection model could potentially be unreliable.

Project Execution

We implemented a custom multi-branch network for enabling early exiting to reduce the inference time for detecting the objects: pedestrian, cyclist, or car on the road using radar for our RODNet, radar-based detection model.

We trained a float16 TensorFlow saved model and TFLite model for 20 epochs using YoloV5 and implemented int8 quantization using TensorFlow and compared the model size, accuracy, and inference time of our camera-based detection model. This is the link to the repository.

Main Learnings

We learned about PyTorch framework and how it differs from TensorFlow in terms of method intentions and implications. We also learned about defining custom training schedule in PyTorch. Additionally, gaining insight into post processing tasks involved in object detection that is vital in removing redundant detections such as the non-max suppression technique which is different in radar-based object detection which employs the OLS[1] metric and camera-based object detection which employs the IoU[2] metric. Furthermore, we also learned about traditional object detection metrics such as mean average precision, recall, and accuracy that was adopted by radar-based object detection but using the OLS metric instead of the IoU for the thresholds. We also gained insight into 3D convolutional and transpose convolutional layers.

Problem/Use Case

There is a limitation of utilizing conventional camera-based sensing object detection networks under conditions of weak/strong lighting during night operations or very bright environments or perhaps poor weather conditions such as fog, smoke, dust, storms, and rain which can lead to little/high exposure or blur/occluded images.

The intention of utilizing radar-based detection on top of camera-based detections is to take advantage of radar's increase in reliability to sense objects during harsh weather conditions. Frequency modulated continuous wave (FMCW) radar operates in the millimeter-wave (MMW) band (30-300GHz) which is lower than visible light, but gives it properties of increase penetration through fog, smoke, and dust and increased range detection capabilities due to the large bandwidth and increased working frequency *(RODNet, 2021)*.

The increasing trend of autonomous vehicle operations provides the prospects of utilizing radar-based object detection technology to aid in smarter decisions in autonomous vehicles by providing more accurate detections under poor visibility conditions where camera-based sensing accuracy could decline.

Dataset and Description

ROD2021 (Radar-Based Detection)

The dataset used concerning radar-based detections in this project is a subset of CRUW dataset (*CRUW Dataset*, 2021). The dataset was captured using a pair of stereo cameras and two 77 GHz FMCW radar antenna array. The sensors are well calibrated and synchronized. The dataset samples include the following scenarios: Parking Lot (PL), Campus Road (CR), City Street (CS), Highway (HW).

The role of the stereo cameras amongst other things is to localize 3D positions of the objects and provide monocular depth[3] estimation. The role of radar is to provide range[4] estimation without

any systematic bias. A *Heuristic Fusion Algorithm* is deployed to fuse the location results from both the camera and the radar where the azimuth[5] estimation of the camera and the range estimation of the radar is trusted. The purpose of this algorithm is to remove redundant radar locations and generate the best distance matches between each radar and camera detection. This process produces the final CRF annotations.

There are 3 classes in the dataset: pedestrian, cycle, and car. There are 40 training video sequences. Each sequence has around 800-1700 frames sampled at 30 FPS. Each frame consists of an RGB image and 4 radar NumPy files for unique chirp IDs. These files are pre-processed range-azimuth heat maps obtained after several Fast Fourier Transform. They have dimensions of 2x128x128. It also contains 40 annotation text files for 40 different sequences, where the ground truth labels about object's center location in terms of range and angle index is mentioned.

Original dataset link. However, we are using reduced dataset containing only 10 sequences for training.

EDI TY (Camera-Based Object Detection)

The dataset used concerning camera-based detections in this project was taken from Roboflow (*EDI TY Object Detection Dataset (V14, 2023-11-26 11:31am) by Vishwakarma Institute of Technology,* 2023) formatted as a YOLOv5 PyTorch Darknet dataset meaning it has an image and annotation file pair corresponding to the following directory structure.

```
Root directory
      |-------images
                |-------train
                          |----*.jpg/png/jpeg/
                |-------validate
                          |----*.jpg/png/jpeg/
                |-------test
                          |----*.jpg/png/jpeg/
      |-------labels
                |-------train
                          |----*.txt
                |-------validate
                          |----*.txt
                |-------test
                          |----*.txt
      |-------labels.txt
```

Figure 1: Camera-Based Detection Dataset Directory Structure

Furthermore, the annotations are in the YOLO format "*xc, yc, width, height"* which is used to describe the bounding box coordinates around the objects. Lastly, these coordinates are also normalized to the image dimensions.

There are currently 5 classes in the dataset *bike, car, cycle, pedestrian, and signal*. However, *bike* in this case is representative of motorbike whereas *cycle* is representative of common bicycles. Lastly, *signal* refers to traffic signals in the road. The most common annotation in the dataset is

of the class *car* ~3000-4500 objects whereas the rest of the classes are under ~1000 objects, with *bike* being the least number of annotations ~200 objects. The dataset contains 2801 training images, 802 validation images, and 397 test images. All images in the dataset were resized to a square resolution of 640x640 in RGB format. Machine Learning Methods and Approaches

Machine Learning Methods

Base Model – Radar Based

The base model architecture of the radar-based model is RODNet shown in the diagram below.

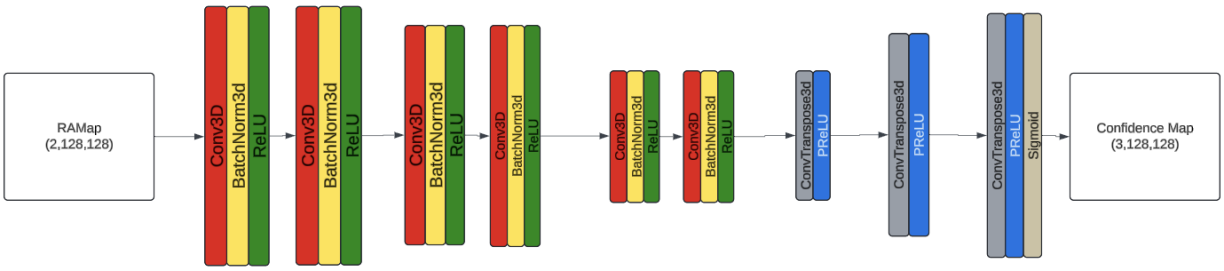

Figure 2: RODNet Base Model Architecture

The inputs are 4D NumPy arrays, where an additional dimension represents 16 consecutive frames in a sample to capture the temporal features. There are 28,601,088 parameters in the encoder section and 5,919,174 parameters in the decoder section. This model uses a custom metric called object location similarity (OLS) to remove redundant detections/peaks from the predicted confidence map.

$$[\text{Eq. 1}]\ OLS = e^{\frac{-d^2}{2(s*\kappa_{cls})^2}}$$

Here, d represents the distance between two points in the RF image, s represents the distance from the radar sensor, and κ_{cls} is a constant for the object class.

This location-based non-maximum suppression involves detecting the 8-neighbor peaks by sweeping a 2D window across the map and sorting them into ascending order. Then, the OLS distance between two elements is calculated. If it is greater than the threshold, the peak with the lower confidence score is suppressed.

Base Model – Camera Based

The base model architecture of the camera-based model is from YoloV5 shown in the diagram below.

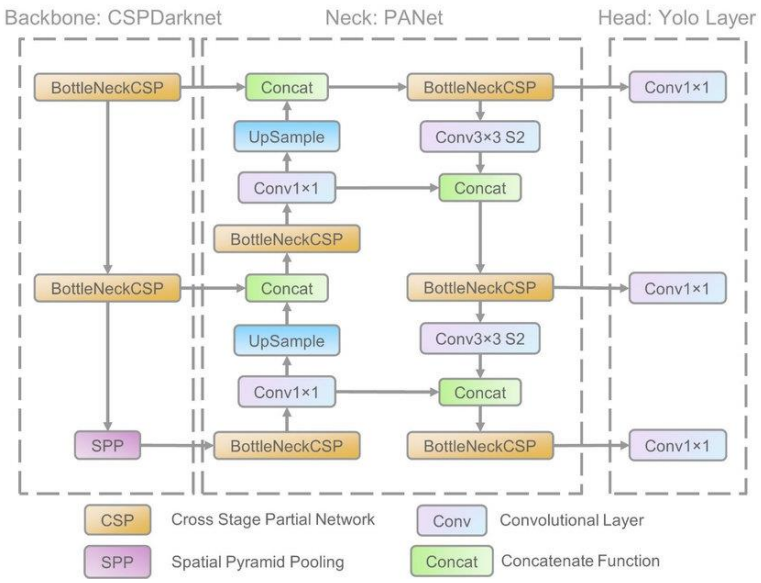

Figure 3: YoloV5 Architecture. Image was taken from: https://www.researchgate.net/figure/The-network-architecture-of-Yolov5-It-consists-of-three-parts-1-Backbone-CSPDarknet_fig1_349299852

The network architecture consists of 3 parts: *Backbone*, *Neck*, and the *Head*. The *Backbone* is responsible for feature extractions of the input data, the *Neck* is responsible for fusing features, and the *Head* is responsible for producing model detection bounding boxes, scores, and labels (*Figure 4. The Network Architecture of Yolov5*, 2021).

Fast AI – Radar Based

In order to reduce the inference time of the RODNet model used for object detection using radar, we have implemented a custom multibranch architecture using PyTorch. The figure below represents the block diagram of the architecture.

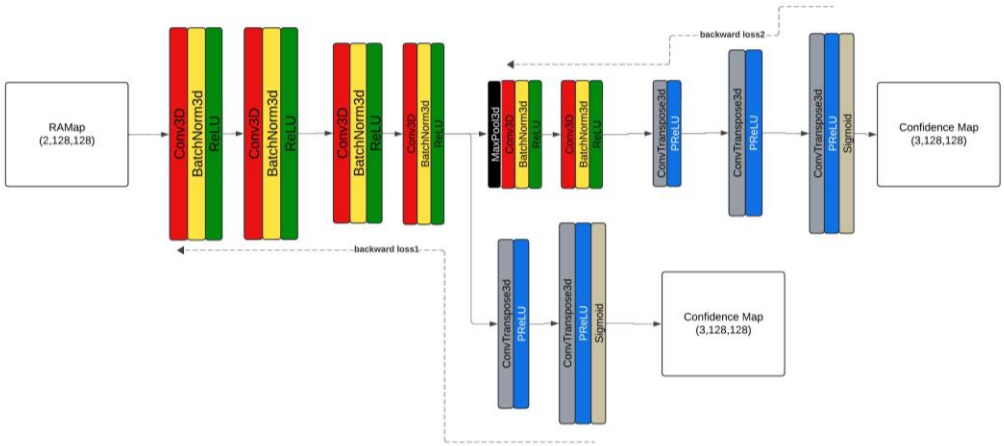

Figure 4: Early Exiting Architecture

The model is divided into two sections: the encoder and the decoder. The encoder uses 3D convolutional layers, whereas the decoder uses 3D transpose convolutional layers. After 4 convolutional layers in the encoder, an early branch is inserted. This branch has 2 transpose convolutional layers, followed by a PReLU activation layer, and the last layer is a sigmoid activation layer. The long branch has 2 convolutional layers and 3 transpose convolutional layers, followed by a sigmoid layer.

The weights of the short branch and the base are updated by backward propagating the binary cross-entropy loss between the predicted confidence map and the ground truth confidence map. Whereas the weights of the long branch are updated according to its own loss value obtained. **The base, short branch, and long branch contains 6481152, 1200453, and 28039110 parameters/weights.**

Tiny AI – Radar Based

The initial intention of the project regarding model compression was to utilize PyTorch methods for pruning and quantization on the existing RODNet model architecture that is implemented in PyTorch. However, there were some challenges encountered during the implementation of these methods which will be discussed in the next sections.

As noted above, due to the complex challenges and limited time allowances, the project intention shifted towards implementing TinyAI for our camera-based detection model using TensorFlow which will be described in the sections below.

PyTorch Pruning and Challenges

Pruning was the first method explored for compressing the model which aims to remove weights with values close to zero to reduce the overall size of the network. PyTorch performs both One-shot and Iterative pruning methods. Furthermore, PyTorch provides two more sub-types for pruning: *Local* and *Global* pruning. Local pruning is a technique to prune the model tensors one at a time to provide the users direct control over pruning of specific tensors and to compare statistics (weight magnitude, activation gradient, etc.) between pruned tensors. Global pruning is a technique to prune the model all at once, the lowest specified sparsity percentage across the whole model (*Pruning Tutorial — PyTorch Tutorials 2.2.1+cu121 Documentation*, n.d.).
This project explored one-shot pruning using both local and global pruning techniques in PyTorch. After implementing these techniques and applying them to the existing RODNet architecture using PyTorch guidelines available, the model size increased as shown below.

```
Original parameters = 34520260
Size (KB): 138101.86
Sparsity in conv1a.weight: 2.16%
Sparsity in conv1b.weight: 12.79%
Sparsity in conv2a.weight: 12.77%
Sparsity in conv2b.weight: 17.98%
Sparsity in conv3a.weight: 18.00%
Sparsity in conv3b.weight: 25.55%
Sparsity in convt1.weight: 14.25%
Sparsity in convt2.weight: 10.08%
Sparsity in convt3.weight: 1.99%
Global sparsity: 20.00%
Pruned parameters = 34520260
Size (KB): 141788.51
```

Figure 5: PyTorch One-shot, Local Pruning Results

The results of pruning in PyTorch show that the model size increased from 138,101.86 KB to 141,788.51 KB after pruning the model. Although the model size increased, the model sparsity was expected because it was specified to be at 20.00%. The model parameters also did not change, which was unexpected because the pruned weights were removed using `torch.nn.utils.prune.remove()` method.

In light of this, ongoing research is still being performed to understand the cause of this phenomenon. However, one case study from another source states that "there are methods that implement pruning in PyTorch, but they do not lead to faster inference time nor memory savings" (Polivin, 2021) because the sparse operations are not currently supported in PyTorch 1.7 and lower. This project builds on top of the RODNet repository which specifies PyTorch version to 1.4. Furthermore, other sources shown below state that saving the model after pruning retains the original size of the model, but the sparse operation reduces the model size.

```
import torch
import torch.nn.utils.prune as prune

t = torch.randn(100, 100)
torch.save(t, 'full.pth')

p = prune.L1Unstructured(amount=0.9)
pruned = p.prune(t)
torch.save(pruned, 'pruned.pth')

sparsified = pruned.to_sparse()
torch.save(sparsified, 'sparsified.pth')
```

```
21K sparsified.pth
40K pruned.pth
40K full.pth
```

Figure 6: PyTorch Proper Pruning Workflow
Image taken from: https://github.com/pytorch/tutorials/pull/605

Since the current PyTorch version used in this project does not support this functionality, it will be in our future work to implement pruning with PyTorch > 1.7.

PyTorch Quantization and Challenges

Quantization is another method explored for compressing the model which transforms the computation and tensors into smaller bit-widths leading into reduced precision (*Quantization — PyTorch 2.2 Documentation*, n.d.).

PyTorch offers three types of quantization:

1. Dynamic quantization (weights are quantized but the activations are read/stored in floating point but quantized during compute).

2. Static quantization (weights are quantized, activations are quantized, but calibration is required during post-training).
3. Static quantization aware training (weights quantized, activations quantized, quantization numeric modeled during training).

Since the RODNet model architecture is comprised of 3D convolutional layers, static quantization will be utilized as dynamic quantization currently does not have support for these layers. Furthermore, static quantization aware training will not be utilized as the primary motivation is to perform post-training quantization of the model. However, after implementing these techniques and applying them to the existing RODNet architecture to quantize the model to 8-bit integer representation using PyTorch guidelines available, the model size only slightly decreased.

```
Original Size (KB): 138101.86
Int8 Quantization Size (KB): 138101.54
```

Figure 7: PyTorch Int8 Quantization Model Size

Furthermore, when performing 16-bit float quantization of the model, the model size decreased drastically, but it was unexpected that the 8-bit integer model is larger than 16-bit float model.

```
Original Size (KB): 138101.86
cdc.encoder.conv1a.weight :  torch.float32
Decreased Precision Size (KB): 69057.7
cdc.encoder.conv1a.weight :  torch.float16
```

Figure 8: PyTorch Float16 Quantization Model Size

Furthermore, to quantize the model to 16-bit floating point, we invoked the property `` `.half()` `` of the model that was originally 32-bit floating point. However, we had to deviate from our original training parameters in order to fine tune the quantized model shown below.

- Wrap the training method with `` `torch.cuda.amp.autocast()` `` to automatically cast layers to their supported types (*ValueError : Attempting to Unscale Fp16 Gradients,* 2020).
- Utilize a specific loss function `` `torch.nn.BCEWithLogitsLoss` `` over the original `` `torch.nn.BCELoss` ``
- Set `` `eps=1e-4` `` in the `` `torch.optim.Adam` `` over the original `` `eps=1e-8` `` such that the loss returned will not result in NaN.

However, we decided to abandon this path for the following reasons:

- Unfamiliarity with proper usage of the API in PyTorch will lead to loopholes and more time consumption.
- The deviation of the original training parameters will make for an inconsistent experiment.
- The quantized and fine-tuned model fails to generate detections on the validation set.

Tiny AI – Camera Based

The camera-based architecture of the model produced by YoloV5 is 16-bit precision floating point. The compression technique used is *Post-Training Dynamic Range Quantization* in TensorFlow (*Post-training Quantization*, n.d.). This type of quantization technique only quantizes the weights from floating point to 8-bit precision integer at conversion time. However, the outputs are still stored as floating point. This type of quantization technique was chosen due to its simplicity of usage of the API.

Evaluation Metrics and Results

Radar-Based Metrics

Base Model

1. **Inference Time & Prediction Results**
   The average inference time between 1000 samples took approximately 1.0s for 16 frames, therefore, around 62.5ms for a single frame.

```
Average inference time : 0.9999359176158905 seconds
```

Figure 9: Base Model Average Inference Time

Total number of detections for objects of all classes present in the 1000 samples is listed.

```
Actual number of pedestrians: 789 and detected: 418
Actual number of cyclist: 502 and detected: 365
Actual number of car: 602 and detected: 2211
```

Figure 10: Base Model Detection to Ground Truth Comparisons

2. **Loss Probability Distribution Function**
   The loss PDF curve for all three kinds of objects is shown in the graph below:

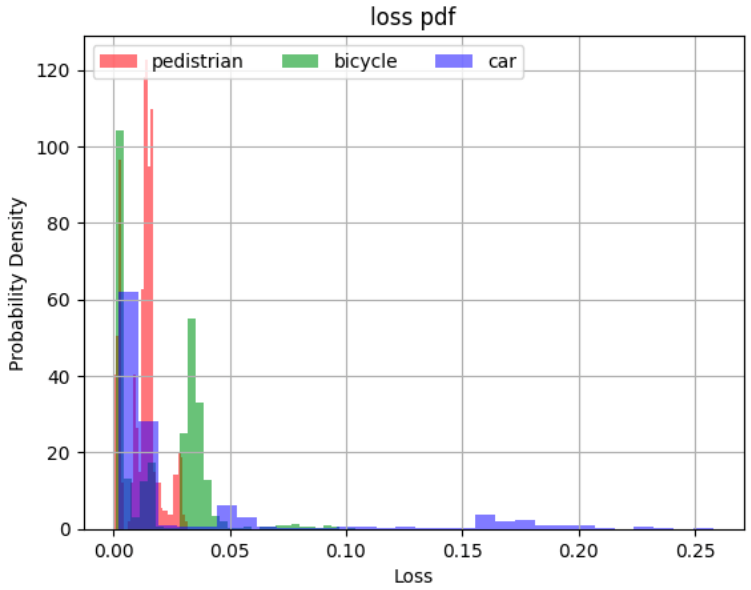

Figure 11: Base Model Loss PDF

Multibranched Model

1. **Number of Early Exits**

   Unlike classification, which uses entropy as a threshold to determine whether the output should exit early through a short branch or long branch, we have defined a new threshold which compares the number of detections per object class with the ground truth. Firstly, we set the threshold too high, where the output will exit early if all the object class detections match with the ground truth. However, only 2 frames exited early. After lowering the threshold, the output will exit early if any one of the object classes matches with the ground truth. Out of 1000 inputs, 731 samples exited early with most of the false positives being for cars.

```
Number of samples successfully exited early 731
Total number of pedestrians: 342
Total number of cyclist: 419
Total number of car: 1767
GT Total number of pedestrians: 533
GT Total number of cyclist: 351
GT Total number of car: 410
```

Figure 12: Early Exiting Results

Tradeoff

The effect of increasing the threshold reduces the number of early exit worthy detections. However, decreasing the threshold increases the number of early exit detections that may not be the right detections. A good performing model will need to have a threshold set

that is not too lenient as to allow poor detections, but not too strict that prevents the model from outputting detections in the early exit branch.

2. **Inference Time**

This metric measures the responsiveness of the model and is vital for real time analysis and justifies the use of multibranch architecture. The average time for the short branch measured across 1000 samples consisting of 16 consecutive frames is *0.93s* and for long branch it is *1.12s*. Hence, the early exit is *faster by 190ms*.

```
Average Short branch inference time : 0.9319982299804688 seconds
Average Long branch inference time  : 1.1247846949100495 seconds
```

Figure 13: Multibranch Timing Performance

These results are for 16 consecutive frames, hence for one frame it takes around *58ms* and *70ms* for short and long branch, respectively.

Output	Inference time of 16 frames [s]	Inference time of single frame [s]
Short branch	0.93	0.058
Long branch	1.12	0.070

Table 1: Multibranch Frame Timing Comparisons

3. **Loss Probability Distribution Function**

The loss probability distribution functions for short and long branch are also shown.

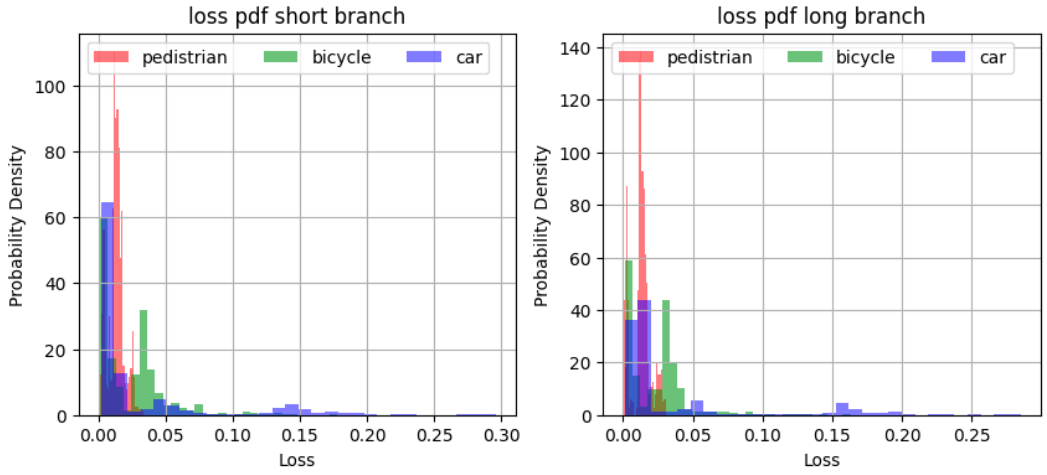

Figure 14: Loss PDF Multibranch Model

The loss curve for the pedestrian and cyclist has a much narrower distribution compared to the car and are also centered at low loss.

Tradeoff

The short branch loss curve for the car has higher probability density at around 65 compared to longer branch around 40. Thus, explaining the higher number of false negative for car when output was exited early.

Camera-Based Metrics

The metrics were produced using the library deepview-validator (*How to Evaluate Computer Vision Models with Deepview-Validator*, 2024). The metrics shown are precision, recall, and accuracy which follows standard equations based on the total number of true positives, false positives, and false negatives recorded.

$$\text{[Eq. 2]. } precision = \frac{True\ Positives}{True\ Positives + False\ Positives}$$

$$\text{[Eq. 3]. } Recall = \frac{True\ Positives}{True\ Positives + False\ Negatives}$$

$$\text{[Eq. 4] } Accuracy = \frac{True\ Positives}{True\ Positives + False\ Positives + False\ Negatives}$$

The mean average metrics such as mAP@0.50 is based on the IoU threshold set to 0.50 and it is calculated as the average of the average precision (area under the curve) of each class in the precision vs. recall curve. The rest of the metrics such as mAR@0.50 and mACC@0.50 are not based on the area under the curve, but it is based on the average recall (following the same equations above) but still considering the IoU threshold at 0.50.

The timings shown are only for the inference time which measures the model's time for producing detections on the image passed, where "max" suggests the maximum timing recorded, "min" is the minimum timing recorded, and "average" is the average timing across all the validation samples passed.

The plots provided below are *precision vs. recall*, *confusion matrix,* and *histogram of classes.*

1. The ***precision vs. recall curve*** shows the tradeoff between precision and recall, a common phenomenon in object detection. However, the ideal case would be a large area under the curve to represent a high performing object detection model.
2. The ***confusion matrix*** maps the number of detections falling in the right category of classes to match the ground truth. An ideal case would be for all the detections to fall in the diagonal representing perfect matches of true positives for the detections. However, a detection of background, but a non-background ground truth would indicate a false negative (the model missed to detect this annotation) and a ground truth of background, but a non-background detection would indicate a false positive (the model made a detection around an object of non-interest). The other case would be misclassifications of categories which are also labeled as false positives.

3. The ***histogram of classes*** shows the performance per class in terms of precision, recall, and accuracy representing the histogram on the left or the number of true positives, false positives, and false negatives representing the histogram on the right. An ideal case would be high precision, recall, and accuracy, and as well as true positives, but low number for the false negatives and false positives.

Base Model

The model evaluated is the 16-bit precision floating-point model that was trained using YoloV5 and then post converted to a TFLite model. The model size is 13.51MB.

1. Base Model Detection Metrics

Accuracy	Precision	Recall	mAP@0.50	mAR@0.50	mACC@0.50
57.18%	90.58%	60.76%	64.01%	44.88%	41.69%

Average Inference Time (ms)	Max Inference Time(ms)	Min Inference Time (ms)
388.67	1297.00	328.00

Table 2: Base Model Metrics

2. Base Model Detection Plots

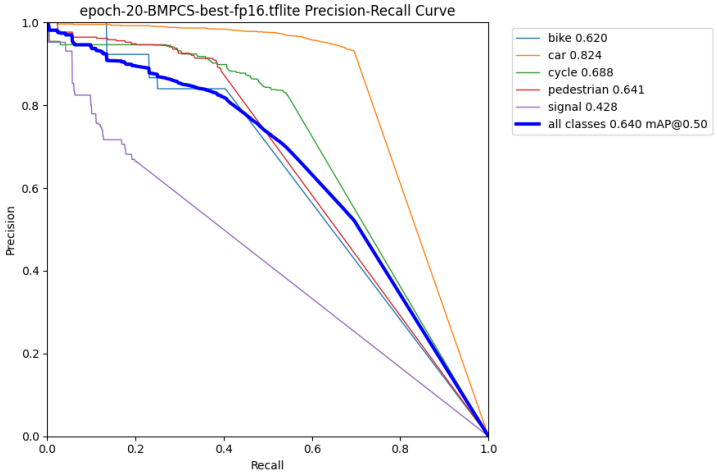

Figure 15: Base Model Precision vs. Recall

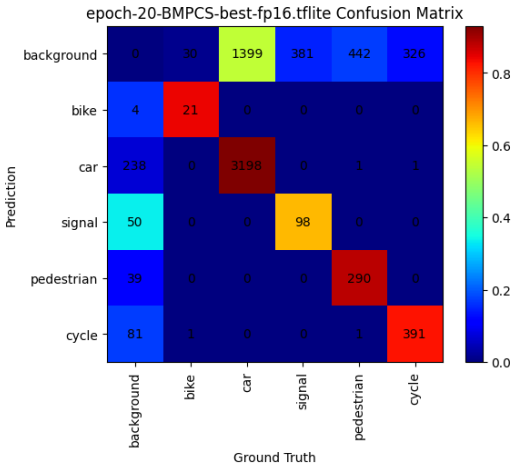

Figure 16: Base Model Confusion Matrix

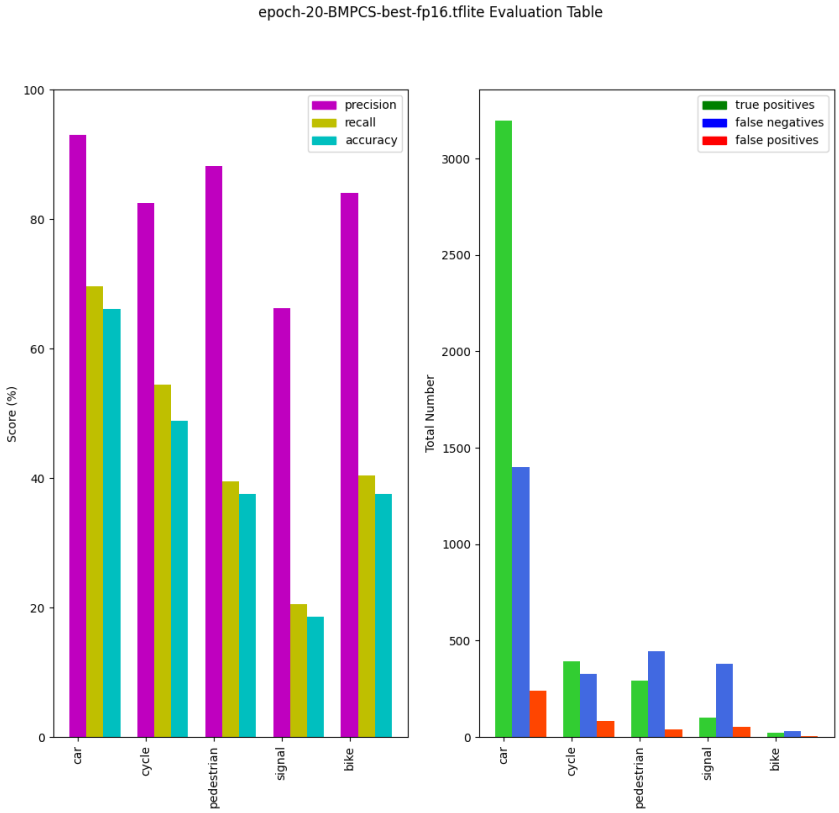

Figure 17: Base Model Histogram of Classes

Based on the plots shown above, it seems that the highest metric accounts for the class "car", the model produces the most true positives for this class, and yet it produces the most false positives and false negatives as well because the dataset contains the most annotations around cars. On the other hand, the model has the lowest performance at detecting the class "signal" despite the lowest number of samples in the dataset is from the class "bike". Lastly, the rest of the classes, "bike", "cycle", and "pedestrian" have similar performance when it comes to the model detecting these objects.

Quantized Model

The model evaluated is the 8-bit precision integer model that took the 16-bit float model and dynamically quantized at post training. The model size is 6.97MB. This was around 2x decrease in model size when compared to the base model.

1. Quantized Model Detection Metrics

2. Accuracy	Precision	Recall	mAP@0.50	mAR@0.50	mACC@0.50
57.00%	90.36%	60.65%	63.82%	44.65%	41.47%

Average Inference Time (ms)	Max Inference Time(ms)	Min Inference Time (ms)
2106.06	3422.0	1828.00

Table 3: Quantized Model Metrics

The accuracy and the rest of the metrics remains relatively the same as the base model, but there is a slight decrease in the metrics which is expected in quantized models due to the decrease in precision. However, it was unclear that the model inference time increase 2-5x after quantization. Due to time constraints, there was no chance of investigating this further, but it will be part of the future work.

Tradeoff

Quantization is a method of representing the weights with less precision to produce a smaller model. However, this comes at the price of reduction in accuracy because the weights are no longer as precise and becomes more of an approximation.

3. Quantized Model Detection Plots

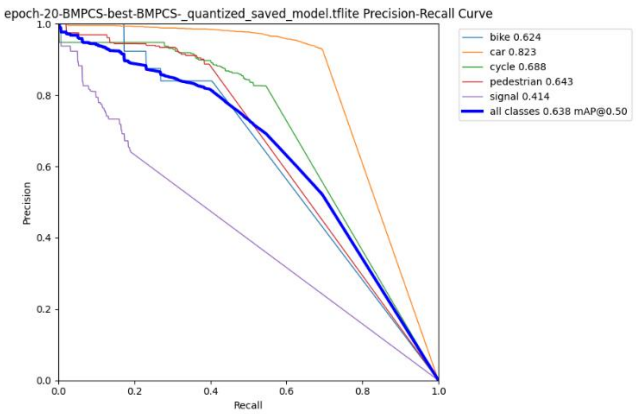

Figure 18: Quantized Model Precision vs. Recall

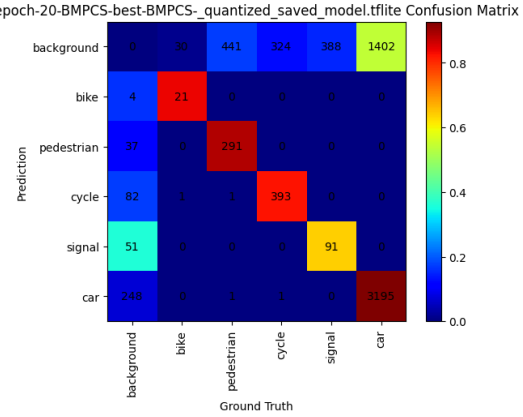

Figure 19: Quantized Model Confusion Matrix

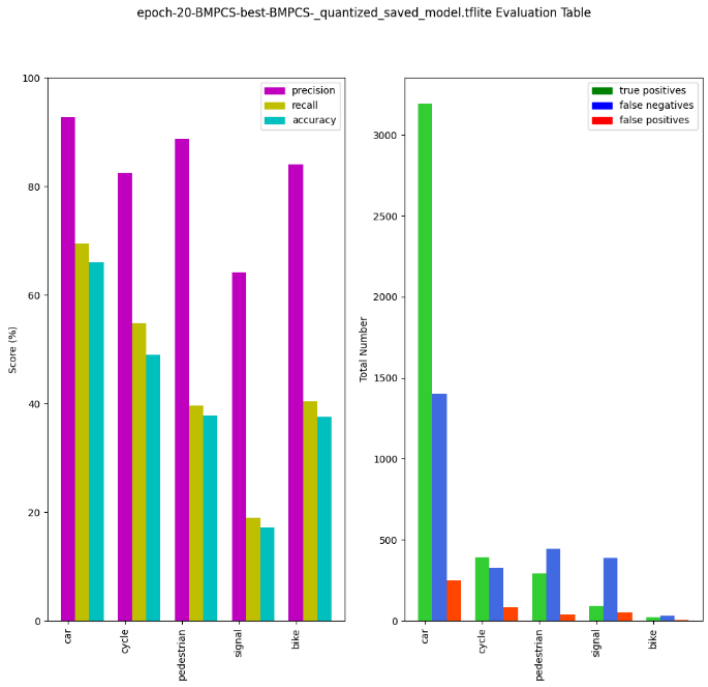

Figure 20: Quantized Model Histogram of Classes

A similar pattern is shown in the plots when compared to the base model where the model performs well for the class “car” and the worst for the class “signal”. However, when analyzing the numbers shown in the confusion matrix, it is evident that the quantized model has 7 more false negatives, 10 more false positives, and 7 less true positives when compared to the base model.

Radar-Based and Camera-Based Performance Comparisons

One of the primary challenges that this project report seeks to address is to identify cases where radar-based object detection could be reliable. One of the cases identified is during low visibility conditions that may impact the performance of conventional camera-based detections. The following results are from the camera-based detection model which shows its ability to detect in 3 levels of difficulty marking clear to obscured images due to lighting or other factors.

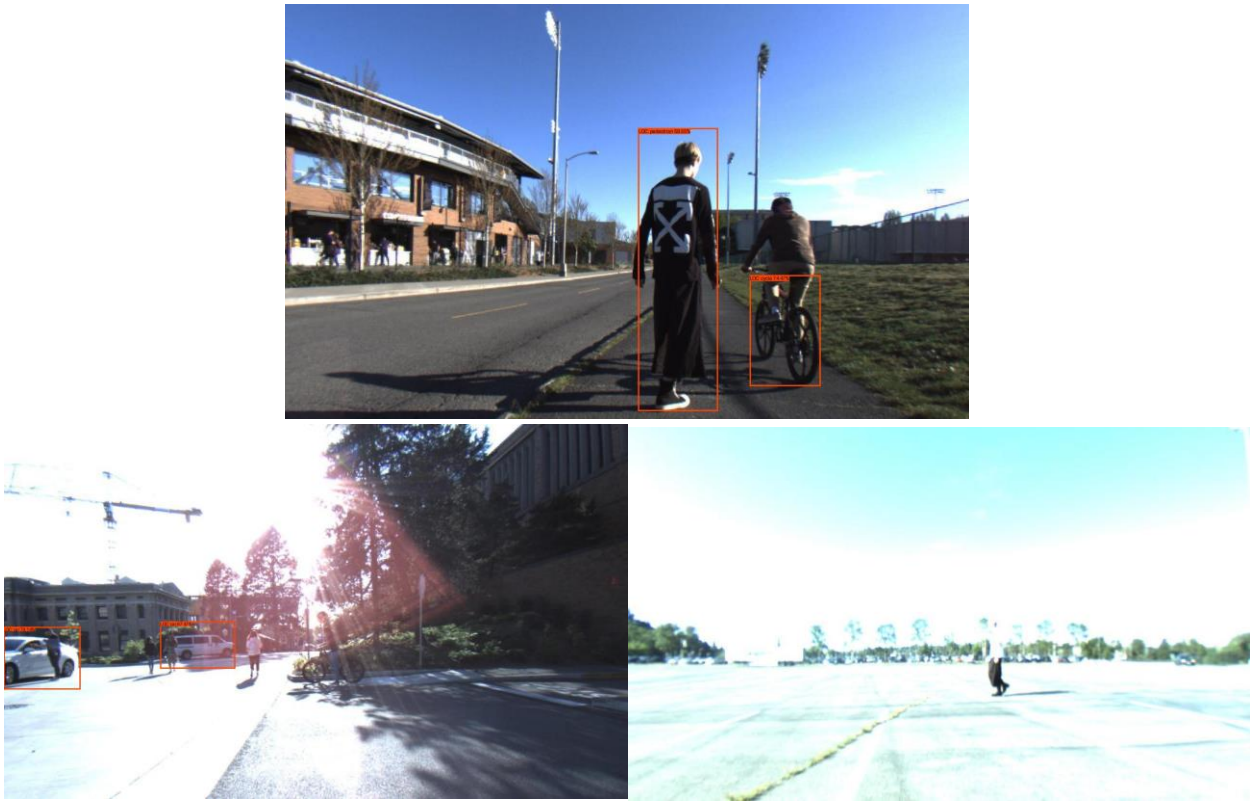

Figure 21: Camera-Based Detection Results 3 Difficulties

The top image shows a clear, easy image to perform detections for which the camera-based detection model was able to detect the pedestrian and the cycle. However, in medium difficulty images such as what is shown on the bottom left, the camera-based detection model only captured two of the objects, failing to detect important objects such as the pedestrians and the cycle. For the bottom right image, marks a difficult image with too much lighting that blends the pedestrian to the background for which the camera-based detection model fails to detect.

Now the following results will show the radar-based detection model using the same sequences the contains the same images but relying on the radar data from these sequences to produce detections.

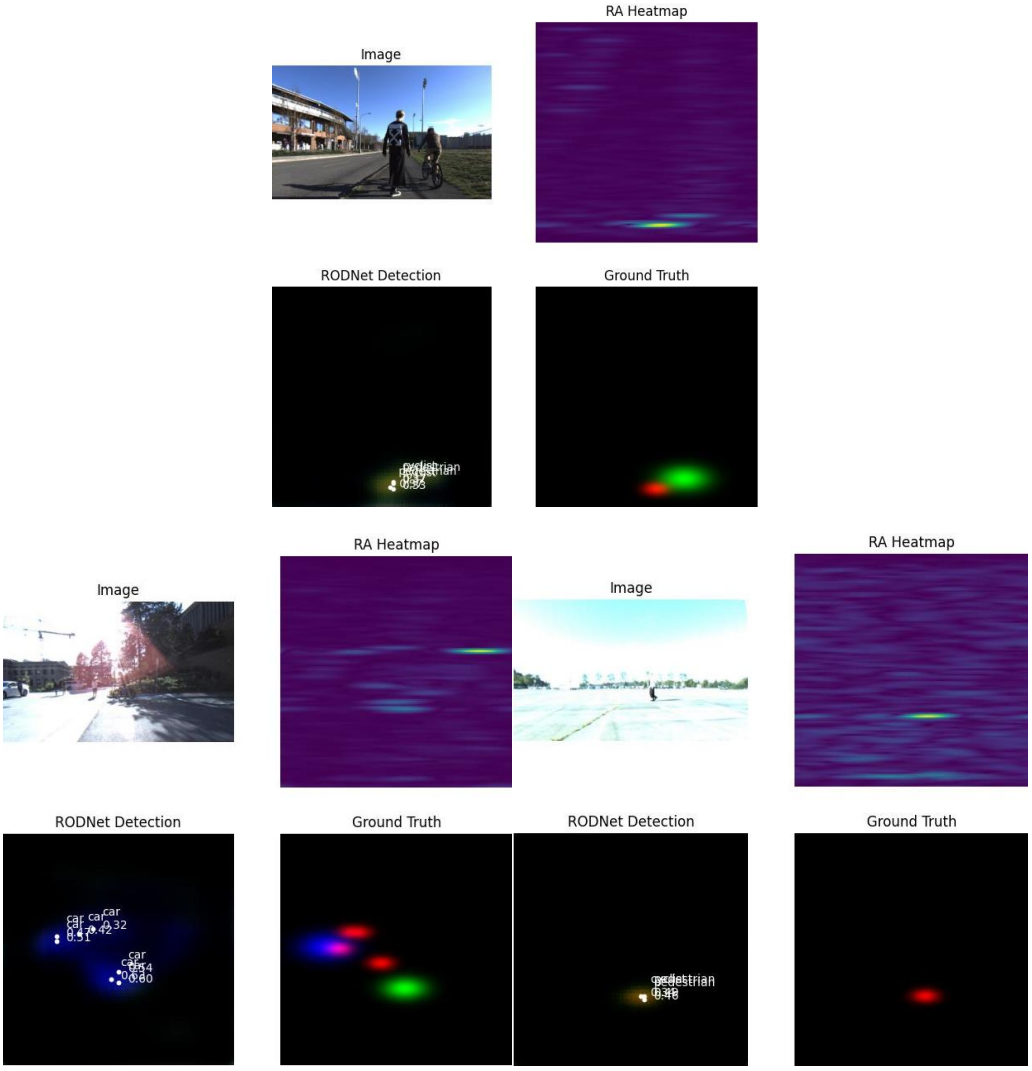

Figure 22: Radar-Based Detection Results 3 Difficulties

As shown in the results above, the radar-based object detection model was able to provide detections on all level of difficulty which provides promising ground for a reliable use of this field in machine learning on cases with low visibility conditions that could result in poor performance in conventional camera-based object detection models.

Challenges and Future Improvements

Challenges Faced

Throughout the project, we encountered numerous challenges during training and testing the RODNet model, which are listed below:

- Learning curve involved in using PyTorch framework compared to TensorFlow used throughout the coursework.
- The annotation text files for some of the sequences provided in the ROD2021 dataset were incomplete, with missing range and angle index for some frames. The included script for processing the dataset was unable to handle the testing sequences, resulting in a key-value error when attempting to enumerate through the PyTorch data loader.
- Due to issues with the testing data, we were unable to use the provided 'test.py' script and had to write our own script to obtain prediction results. This process was time-consuming as we had to define the OLS metric used for obtaining final detections without proper instructions or documentation.
- Significant training time on GPU was required. Since our project was an object detection task, the model has millions of parameters, and not having pretrained weights made it difficult to implement transfer learning.
- Complications arose with running 'eval.py' as it requires the testing sequences to be in a specific format. Due to difficulties in processing the test data, we had to perform a random split on the training dataset, making it impossible to use 'eval.py' to obtain the average precision and average recall.
- During our attempts to implement model compression techniques for the RODNet model, specifically pruning and quantization within PyTorch, we encountered unexpected results. Instead of the model size being reduced after pruning, it increased with uneven sparsity across the layers. For quantization, only 'float16' was able to reduce the model size, while 'int8' did not result in any reduction in the model size.

Future Work

For improving our model performance, we are looking to do the tasks listed below:

- The first thing we want to tackle is to create a new script that can calculate average precision and average recall without the data to be formatted in a certain manner like with 'eval.py'.
- Since ROD2021 is the only dataset available for radar detections with range-azimuth heat maps as inputs. We want to refine the dataset by removing the frames from the dataset with missing ground truth detection labels.
- To reduce the training time and increase the average precision and average recall, we want to implement transfer learning in our multibranch architecture.

- We also want to research into proper implementation of compression techniques like pruning and quantization in PyTorch. Then integrate an iterative pruning schedule with our multibranch architecture.
- Lastly, we want to investigate the increase in inference time post quantization with the camera-based object detection model (YOLOv5).

References

RODNet: A Real-Time Radar Object Detection Network Cross-Supervised by Camera-Radar Fused Object 3D Localization. (2021, June 1). IEEE Journals & Magazine | IEEE Xplore. https://ieeexplore.ieee.org/abstract/document/9353210

Y. W. (2022, March 15). *GitHub - yizhou-wang/RODNet: RODNet: Radar object detection network*. GitHub. https://github.com/yizhou-wang/RODNet/tree/master

PyTorch. (n.d.). PyTorch. https://pytorch.org/

TensorFlow. (n.d.). TensorFlow. https://www.tensorflow.org/

U. (2022, November 22). *GitHub - ultralytics/yolov5: YOLOv5 🚀 in PyTorch > ONNX > CoreML > TFLite*. GitHub. https://github.com/ultralytics/yolov5

CRUW Dataset. (2021, January 18). https://www.cruwdataset.org/

EDI TY Object Detection Dataset (v14, 2023-11-26 11:31am) by Vishwakarma Institute of Technology. (2023, November 26). Roboflow. https://universe.roboflow.com/vishwakarma-institute-of-technology-yqqb5/edi-ty/dataset/14

Pruning Tutorial — PyTorch Tutorials 2.2.1+cu121 documentation. (n.d.). https://pytorch.org/tutorials/intermediate/pruning_tutorial.html

Polivin, O. (2021, December 24). *Experiments in Neural Network Pruning (in PyTorch)*. Medium. https://olegpolivin.medium.com/experiments-in-neural-network-pruning-in-pytorch-c18d5b771d6d

P. (2020, February 19). *Add pruning tutorial by mickypaganini · Pull Request #605 · pytorch/tutorials*. GitHub. https://github.com/pytorch/tutorials/pull/605

Quantization — PyTorch 2.2 documentation. (n.d.). https://pytorch.org/docs/stable/quantization.html

ValueError : Attemting to unscale fp16 Gradients. (2020, May 17). PyTorch Forums. https://discuss.pytorch.org/t/valueerror-attemting-to-unscale-fp16-gradients/81372/3

Figure 4. The network architecture of Yolov5. (2021, February). ResearchGate. https://www.researchgate.net/figure/The-network-architecture-of-Yolov5-It-consists-of-three-parts-1-Backbone-CSPDarknet_fig1_349299852

Post-training quantization. (n.d.). TensorFlow. https://www.tensorflow.org/lite/performance/post_training_quantization

How to Evaluate Computer Vision Models with Deepview-Validator. (2024). Au-Zone Technologies. https://support.deepviewml.com/hc/en-us/articles/11511279358221-How-to-Evaluate-Computer-Vision-Models-with-Deepview-Validator

ElSayedMMostafa. (n.d.).Elsayedmmostafa/modulation_classification_using_early_exiting: Research Repo for applying early exiting in modulation classification use case. GitHub. https://github.com/ElSayedMMostafa/modulation_classification_using_early_exiting/tree/main

ENDG 511 Final Project Repository. (n.d.). *GitHub - ENELEngineering/ENDG511_Project_RODNet*. GitHub. https://github.com/ENELEngineering/ENDG511_Project_RODNet/tree/main

Appendix

[1] OLS: Object Location Similarity

[2] IoU: Intersection Over Union

[3] Depth: Measure of vertical distance below or above a system reference.

[4] Range: Measure of linear distance from the center of the system reference.

[5] Azimuth: An angular measurement representing the horizontal angle from a cardinal direction such as North to the object of interest.