National College of Ireland

Configuration Manual

MSc Research Project
Artificial Intelligence

Hugh Plunkett
Student ID: 23312173

School of Computing
National College of Ireland

Supervisor: Abdul Shahid

National College of Ireland
Project Submission Sheet
School of Computing

Student Name:	Hugh Plunkett
Student ID:	23312173
Programme:	Artificial Intelligence
Year:	2025
Module:	MSc Research Project
Supervisor:	Abdul Shahid
Submission Due Date:	20/12/2018
Project Title:	Configuration Manual
Word Count:	XXX
Page Count:	13

I hereby certify that the information contained in this (my submission) is information pertaining to research I conducted for this project. All information other than my own contribution will be fully referenced and listed in the relevant bibliography section at the rear of the project.

ALL internet material must be referenced in the bibliography section. Students are required to use the Referencing Standard specified in the report template. To use other author's written or electronic work is illegal (plagiarism) and may result in disciplinary action.

Signature:	
Date:	15th September 2025

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST:

Attach a completed copy of this sheet to each project (including multiple copies).	☐
Attach a Moodle submission receipt of the online project submission, to each project (including multiple copies).	☐
You must ensure that you retain a HARD COPY of the project, both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer.	☐

Assignments that are submitted to the Programme Coordinator office must be placed into the assignment box located outside the office.

Office Use Only	
Signature:	
Date:	
Penalty Applied (if applicable):	

Configuration Manual

Hugh Plunkett
23312173

1 Introduction

This configuration handbook contains supplementary information on the research project's implementation phase, as well as the environment setup that was used.

1.1 Hardware

The following hardware was used throughout this project:

- **CPU**: AMD Ryzen 9 5900XT 3.3 GHz 16-Core Processor
  - 16 Cores
  - 32 Threads
  - Base Clock: 3.3GHz
- **GPU**: Gigabyte GAMING OC GeForce RTX 4060 Ti 16GB
  - 16GB GDDR6 VRAM
  - Boost Clock: 2.310 GHz
- **Memory**: Corsair Vengeance LPX 64GB (2 × 32GB) DDR4
  - 3200MHz Speed
  - CL16 Latency
  - Dual Channel Configuration
- **Storage**: Kingston NV3 NVMe SSD 2TB
  - PCIe 4.0 ×4 Interface
  - M.2 2280 Form Factor
  - Sequential Read: 6000 MB/s

1.2 Software

Due to different code architetcure some models are built using TensorFlow [1] while others use PyTorch [2] and as a result have slightly different Software architectures. Unless stated otherwise all models use Nvidia CUDA toolkit version 11.6 for GPU acceleration[3].

2 Dataset

This project uses the KITTI Dataset [4] specifically the 3D Object Detection Evaluation 2017 data that is available on their website: Kitti Website Link

> *The 3D object detection benchmark consists of 7481 training images and 7518 test images as well as the corresponding point clouds, comprising a total of 80.256 labeled objects.*[4]

3 Models

3.1 Point RCNN

This work would not be possible if not fr the code provided by Shi, Shaoshuai and Wang, Xiaogang and Li, Hongsheng [5].

Requirements

All the codes are tested in the following environment:

- Linux (tested on Ubuntu 22.04)
- Python 3.6+
- PyTorch 1.13

Install PointRCNN

a. Clone the PointRCNN repository.

```
git clone --recursive https://github.com/sshaoshuai/PointRCNN.git
```

If you forget to add the `--recursive` parameter, just run the following command to clone the Pointnet2.PyTorch submodule.

```
git submodule update --init --recursive
```

b. Install the dependent python libraries like `easydict`, `tqdm`, `tensorboardX` etc.

c. Build and install the `pointnet2_lib`, `iou3d`, `roipool3d` libraries by executing the following command:

```
sh build_and_install.sh
```

Install KITTI Dataset

Please set up your directory as follows:

```
PointRCNN
├── data
│   └── KITTI
│       ├── ImageSets
│       └── object
│           ├── training
│           │   └── calib & velodyne & label_2 & image_2 & (optional:  planes)
│           └── testing
│               └── calib & velodyne & image_2
├── lib
├── pointnet2_lib
└── tools
```

Training

Currently, the two stages of PointRCNN are trained separately. Firstly, to use the ground truth sampling data augmentation for training, we should generate the ground truth database as follows:

```
python generate_gt_database.py --class_name 'Car' --split train
```

Training of RPN stage

To train the first proposal generation stage of PointRCNN with a single GPU, run the following command:

```
python train_rcnn.py --cfg_file cfgs/default.yaml
--batch_size 16 --train_mode rpn --epochs 200
```

After training, the checkpoints and training logs will be saved to the corresponding directory according to the name of your configuration file. Such as for the default.yaml, you could find the checkpoints and logs in the following directory:

```
PointRCNN/output/rpn/default/
```

which will be used for the training of RCNN stage.

Training of RCNN stage

Generate the augmented offline scenes by running the following command:

```
python generate_aug_scene.py --class_name Car
--split train --aug_times 4
```

Save the RPN features and proposals by adding `--save_rpn_feature`:

To save features and proposals for the training, we set `TEST.RPN_POST_NMS_TOP_N=300` and `TEST.RPN_NMS_THRESH=0.85` as follows:

```
python eval_rcnn.py --cfg_file cfgs/default.yaml --batch_size 4 --eval_mode rpn
--ckpt ../output/rpn/default/ckpt/checkpoint_epoch_200.pth --save_rpn_feature
--set TEST.SPLIT train_aug TEST.RPN_POST_NMS_TOP_N 300 TEST.RPN_NMS_THRESH 0.85
```

To save features and proposals for the evaluation, we keep `TEST.RPN_POST_NMS_TOP_N=100` and `TEST.RPN_NMS_THRESH=0.8` as default:

```
python eval_rcnn.py --cfg_file cfgs/default.yaml --batch_size 4 --eval_mode rpn
--ckpt ../output/rpn/default/ckpt/checkpoint_epoch_200.pth --save_rpn_feature
```

Now we could train our RCNN network. Note that you should modify `TRAIN.SPLIT=train_aug` to use the augmented scenes for the training, and use `--rcnn_training_roi_dir` and `--rcnn_training_feature_dir` to specify the saved features and proposals in the above step:

```
python train_rcnn.py --cfg_file cfgs/default.yaml --batch_size 4
--train_mode rcnn_offline --epochs 30 --ckpt_save_interval 1
--rcnn_training_roi_dir
../output/rpn/default/eval/epoch_200/train_aug/detections/data
--rcnn_training_feature_dir ../output/rpn/default/eval/epoch_200/train_aug/features
```

For the offline GT sampling augmentation, the default setting to train the RCNN network is `RCNN.ROI_SAMPLE_JIT=True`, which means that we sample the RoIs and calculate their GTs in the GPU. I also provide the CPU version proposal sampling, which is implemented in the dataloader, and you could enable this feature by setting `RCNN.ROI_SAMPLE_JIT=False`. Typically the CPU version is faster but costs more CPU resources since they use multiple workers.

3.2 Fast Point RCNN

This code was provided upon request by Yilun Chen on behalf of their reseacrh team [6]. All the codes are tested in the following environment:

- Linux (tested on Ubuntu 22.04)
- Python 3.10
- TensorFlow 2.10

Install
Install dependencies including Mayavi for visualization (optional) by running:

```
sh install.sh
```

Install KITTI Dataset
After installing the code and setting up the directory as follows:

```
data/KITTI/data2/
├── training   ← training data
│   ├── image_2
│   ├── label_2
│   └── velodyne
└── testing   ← testing data
    ├── image_2
    ├── label_2
    └── velodyne
```

Traiing and Evaluations
To run this code on one GPU (GPU 0 according to our system we run the following code:

```
python tf_train.py -d 0
```

Then to run the evaluation script on epoch 40 we run the following code:

```
python tf_val.py --test -d 0 40
```

3.3 Complex Yolo V4

This project uses the Complex YOLOv4 implementation developed by Nguyen Mau Dzung, publicaly available on their GitHub repositry[1].
This project utilises the following system structure:

- Linux (tested on Ubuntu 22.04)
- Python 3.6+
- PyTorch 1.13

After cloning the GitHub Repo and installing all requirements using

```
pip install -U -r requirements.txt
```

we install the kitti dataset files, primarily the Velodyne point clouds (29 GB), Training labels of object data set (5 MB), Camera calibration matrices of object data set (16 MB), Left color images of object data set (12 GB). The files should be set up in the following configuration:

```
ROOT
├── checkpoints/
│   ├── complex_yolov3/
│   └── complex_yolov4/
├── dataset/
│   └── kitti/
│       ├── ImageSets/
│       │   ├── train.txt
│       │   └── val.txt
│       ├── training/
│       │   ├── image_2/ <-- for visualization
│       │   ├── calib/
│       │   ├── label_2/
│       │   └── velodyne/
│       ├── testing/
│       │   ├── image_2/ <-- for visualization
│       │   ├── calib/
│       │   └── velodyne/
│       └── classes_names.txt
└── src/
    ├── config/
    ├── cfg/
    │   ├── complex_yolov3.cfg
    │   ├── complex_yolov3_tiny.cfg
    │   ├── complex_yolov4.cfg
    │   ├── complex_yolov4_tiny.cfg
    │   ├── train_config.py
    │   └── kitti_config.py
    └── data_process/
```

[1] https://github.com/maudzung/Complex-YOLOv4-Pytorch

```
    ├── kitti_bev_utils.py
    ├── kitti_dataloader.py
    ├── kitti_dataset.py
    ├── kitti_data_utils.py
    ├── train_val_split.py
    └── transformation.py
├── models/
    ├── darknet2pytorch.py
    ├── darknet_utils.py
    ├── model_utils.py
    └── yolo_layer.py
├── utils/
    ├── evaluation_utils.py
    ├── iou_utils.py
    ├── logger.py
    ├── misc.py
    ├── torch_utils.py
    ├── train_utils.py
    └── visualization_utils.py
├── evaluate.py
├── test.py
├── test.sh
├── train.py
└── train.sh
├── README.md
└── requirements.txt
```

To train the model on our one gpu we run the file train.sh which gives us the following code:

```
#!/usr/bin/env bash
python train.py \
  --saved_fn 'complex_yolov4' \
  --arch 'darknet' \
  --cfgfile ./config/cfg/complex_yolov4.cfg \
  --batch_size 4 \
  --num_workers 4 \
  --no-val \
  --gpu_idx 0
```

Then we just run the test.py file

```
python test.py --gpu_idx 0 --peak_thresh 0.2
```

Which gives us our evaluation results.

3.4 AVOD

This part uses the project code provided by Jason Ku and team [7], without their publicly available github repositry this would not be possible. This project utilises the following system structure:

- Linux (tested on Ubuntu 22.04)
- Python 3.6+
- Tensorflow 2.10

We install the code:
Clone this repo:

```
git clone git@github.com:kujason/avod.git --recurse-submodules

If you forget to clone the wavedata submodule:
git submodule update --init --recursive
```

And we install dependencies:

```
Install Python dependencies:
cd avod
pip3 install -r requirements.txt
```

Make sure to Add *avod* (top level) and *wavedata* to your PYTHONPATH
Compile integral image library in wavedata

```
sh scripts/install/build_integral_image_lib.bash
```

Avod uses Protobufs to configure model and training parameters. Before the framework can be used, the protos must be compiled (from top level avod folder):

```
sh avod/protos/run_protoc.sh
```

Dataset Download
Download the Kitti Dataset and have the file directory look as follows:

```
Kitti/
└── object/
    ├── testing/
    ├── training/
    │   ├── calib/
    │   ├── image_2/
    │   ├── label_2/
    │   ├── planes/
    │   └── velodyne/
    ├── train.txt
    └── val.txt
```

Training
To modify the training config. There are sample configuration files for training inside avod/configs. You can train on the example configs, or modify an existing configuration. To train the model on the GPU we will need to specify GPU 0 so we run the following command to start training:

```
python avod/experiments/run_training.py
--pipeline_config=avod/configs/pyramid_cars_with_aug_example.config
--device='0'
--data_split='train'
```

Testing
For evaluation we run the following command

```
python avod/experiments/run_evaluation.py
--pipeline_config=avod/configs/pyramid_cars_with_aug_example.config
--device='0' --data_split='val'
```

All results should be saved in avod/data/outputs. Here you should see proposals_and_scores and final_predictions_and_scores results.

3.5 VoxelNET

This is built on the following enviroment:

- Linux (tested on Ubuntu 22.04)
- Python 3.10
- TensorFlow 2.10

and uses code from the unofficial VoxelNet github repo[2] maintained by qianguih.
Downloading Model
Using the following steps:

1. Clone the repository.
2. Compile the Cython module:

```
$ python3 setup.py build_ext --inplace
```

3. Compile the evaluation code:

```
$ cd kitti_eval
$ g++ -o evaluate_object_3d_offline evaluate_object_3d_offline.cpp
```

4. Grant execution permission to the evaluation script:

```
$ cd kitti_eval
$ chmod +x launch_test.sh
```

Downloading Database

1. Download the 3D KITTI detection dataset. Required data includes:
   - Velodyne point clouds (29 GB): input data to VoxelNet
   - Training labels of object data set (5 MB): input label to VoxelNet

[2] `https://github.com/qianguih/voxelnet`

- Camera calibration matrices (16 MB): for visualization
- Left color images (12 GB): for visualization

2. Crop the point cloud data:
   - Update directories in `data/crop.py`
   - Run the cropping script:

     ```
     $ python data/crop.py
     ```

   - Note: This will overwrite raw point cloud data with cropped versions
3. Split the training set and arrange folders with this structure:

```
DATA_DIR
├── training
│   ├── image_2
│   ├── label_2
│   └── velodyne
└── validation
    ├── image_2
    ├── label_2
    └── velodyne
```

4. Update paths in configuration files:
   - `config.py`
   - `kitti_eval/launch_test.sh`

Training

1. Specify the GPUs to use in `config.py`: (In our case we want to use GPU_0)
   - Open the configuration file and set the desired GPU devices
   - Example (for using GPUs 0 and 1):

     ```
     GPU_IDS = [0, 1]
     ```

2. Run `train.py` with your desired hyper-parameters:

   ```
   $ python3 train.py --alpha 1 --beta 10
   ```

Testing

1. To produce final predictions on the validation set after training:

   ```
   $ python3 test.py -n default
   ```

3

3.6 VoxelRCNN

For Voxel RCNN we use the official implementation provided on github [3] by Jiajun Deng [8].
We VoxelRCNN ran on the following enviroment:

- Linux (tested on Ubuntu 22.04)
- Python 3.6+
- PyTorch 1.13

Downloading Model
Prepare for the running environment.

```
docker pull djiajun1206/pcdet-pytorch1.5
```

Downloading Database
Download the official KITTI 3D object detection dataset and organize as follows:

```
Voxel-R-CNN
├── data
│   └── kitti
│       ├── ImageSets
│       ├── training
│       │   ├── calib
│       │   ├── velodyne
│       │   ├── label_2
│       │   ├── image_2
│       │   └── planes (optional)
│       └── testing
│           ├── calib
│           ├── velodyne
│           └── image_2
├── pcdet
└── tools
```

Generate the Data Info:

```
python -m pcdet.datasets.kitti.kitti_dataset \
    create_kitti_infos tools/cfgs/dataset_configs/kitti_dataset.yaml
```

And run the setup code:

```
python setup.py develop
```

Training
The configuration file is in tools/cfgs/voxelrcnn, and the training scripts is in tools/scripts. We want to make sure we are using GPU_0 and are training on all 3 classes.

```
cd tools
sh scripts/train_voxel_rcnn.sh
```

Testing
The configuration file is in tools/cfgs/voxelrcnn, and the training scripts is in tools/scripts.

```
cd tools
sh scripts/eval_voxel_rcnn.sh
```

[3] https://github.com/djiajunustc/Voxel-R-CNN

3.7 PVRCNN and CaDNN

Both of these models have official releases and maintained code github repositries provided by the OpenPCDet Development Team [9]. OpenPCDet is an open-source toolbox for 3D object detection, specifically from point clouds.
Both models are tested in the following enviroment:

- Linux (tested on Ubuntu 22.04)
- Python 3.6+
- PyTorch 1.13

Downloading Model
Clone the OpenPCDet repository.

```
git clone https://github.com/open-mmlab/OpenPCDet.git
```

Install the dependent libraries as follows:

```
Install the SparseConv library, we use the implementation from [spconv].
If you use PyTorch 1.1, then make sure you install the spconv v1.0 with (commit 8
```

Install this pcdet library and its dependent libraries by running the following command:

```
python setup.py develop
```

Downloading Database
Download the Kitti Dataset and have it structured as follows:

```
OpenPCDet
├── data
│   └── kitti
│       ├── ImageSets
│       ├── training
│       │   ├── calib & velodyne & label_2 & image_2 & (optional:  planes) & (optional:
│       │   │   depth_2)
│       └── testing
│           ├── calib & velodyne & image_2
├── pcdet
└── tools
```

Generate the data infos by running the following command:

```
python -m pcdet.datasets.kitti.kitti_dataset create_kitti_infos tools/cfgs/dataset_co
```

Training
To train the PVRCNN we run the following command:

```
python tools/train.py  --cfg_file tools/cfgs/kitti_models/pv_rcnn.yaml
```

And to train the CaDNN file we run the following command:

```
python tools/train.py --cfg_file tools/cfgs/kitti_models/CaDDN.yaml
```

Testing
Training
To test the PVRCNN model we just trained we run the following command:

```
python tools/test.py  --cfg_file tools/cfgs/kitti_models/pv_rcnn.yaml
```

And to test the CaDNN model we run the following command:

```
python tools/test.py --cfg_file tools/cfgs/kitti_models/CaDDN.yaml
```

4 FPS Measuring

In order to measure the Frames-Per-Second (FPS) of each model we use linuxs built in time command to find the length of each testing run.
For example:

```
time python tools/test.py --cfg_file ... --ckpt ...
```

Has an output looking like:

```
real    1m45.2s  # Actual wall-clock time (what we want to measure)
user    2m30.1s  (ignore)
sys     0m5.3s   (ignore)
```

As we have 7518 test samples we can easily calculate the time per sample (or FPS) using the following formula:

$$\text{FPS} = \frac{N}{time(s)} = \frac{7518}{time(s)} \tag{1}$$

5 Graphing

All graphing was done in Jupyter Notebook (python) in the following enviroment:

- Windows 11 Home Edition Version 23H2
- jupyter notebooks
- NumPy version: 1.26.4
- Pandas version: 2.3.0
- Matplotlib version: 3.8.4
- Seaborn version: 0.13.2

References

[1] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng, "TensorFlow: Large-scale machine learning on heterogeneous systems," 2015, software available from tensorflow.org. [Online]. Available: https://www.tensorflow.org/

[2] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Köpf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala, "Pytorch: An imperative style, high-performance deep learning library," 2019. [Online]. Available: https://arxiv.org/abs/1912.01703

[3] NVIDIA, P. Vingelmann, and F. H. Fitzek, "Cuda, release: 10.2.89," 2020. [Online]. Available: https://developer.nvidia.com/cuda-toolkit

[4] A. Geiger, P. Lenz, C. Stiller, and R. Urtasun, "Vision meets robotics: The kitti dataset," *International Journal of Robotics Research (IJRR)*, 2013.

[5] S. Shi, X. Wang, and H. Li, "Pointrcnn: 3d object proposal generation and detection from point cloud," in *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2019.

[6] Y. Chen, S. Liu, X. Shen, and J. Jia, "Fast point r-cnn," 2019. [Online]. Available: https://arxiv.org/abs/1908.02990

[7] J. Ku, M. Mozifian, J. Lee, A. Harakeh, and S. L. Waslander, "Joint 3d proposal generation and object detection from view aggregation," in *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE Press, 2018, p. 1–8. [Online]. Available: https://doi.org/10.1109/IROS.2018.8594049

[8] J. Deng, S. Shi, P. Li, W. Zhou, Y. Zhang, and H. Li, "Voxel r-cnn: Towards high performance voxel-based 3d object detection," 2021. [Online]. Available: https://arxiv.org/abs/2012.15712

[9] O. D. Team, "Openpcdet: An open-source toolbox for 3d object detection from point clouds," https://github.com/open-mmlab/OpenPCDet, 2020.