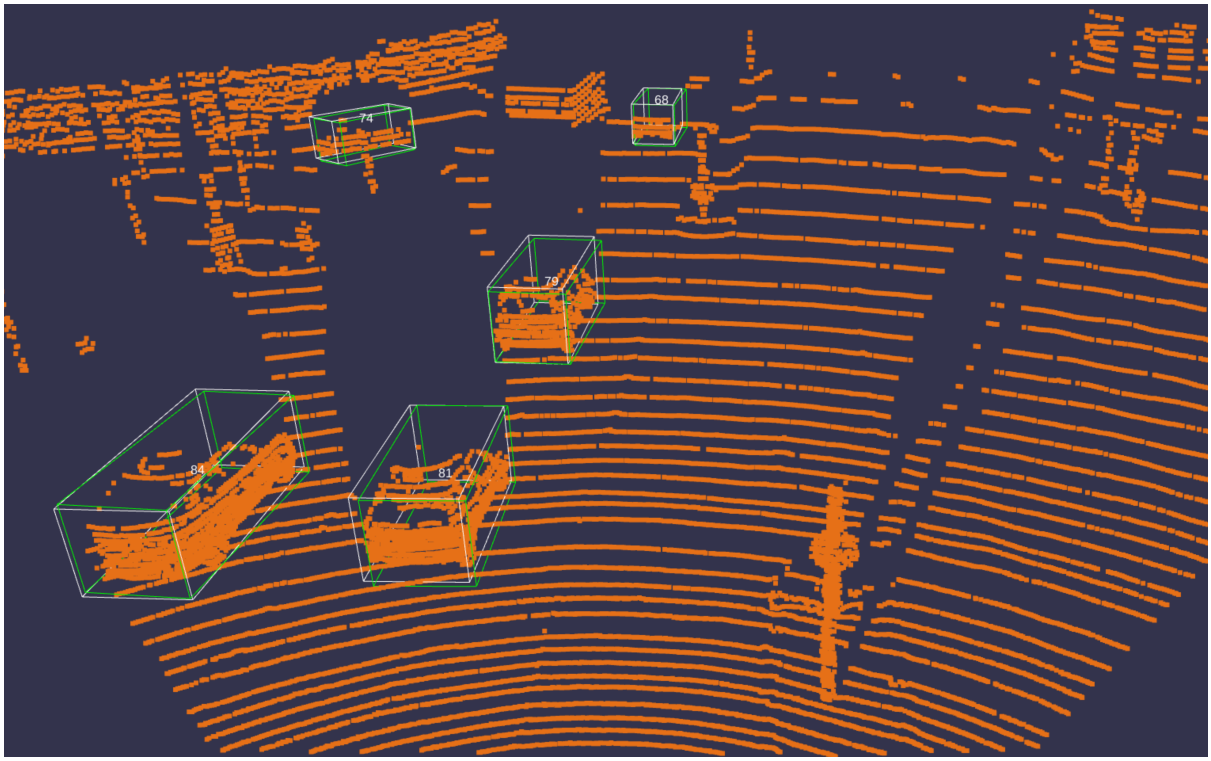
CVAIAC 2024

Project 2

Task 2

Group 05

Carl Brander
Tanguy Dieudonné
Han Xi



December 20, 2024

Problem 2: Building a two Stage 3D Object Detector

Task 2.1: IoU and Recall Calculation

- **What is the average recall for the validation-set?**
  After modifying the testing script used for Task 1 to evaluate all 1'712 validation set datapoints, the average recall measured is 0.815154 or 81.5154%.
- **Why is recall a good metric to assess the quality of the first stage proposals (as opposed to e.g. average precision)?**
  Precision is defined as the number of true positives (TP) over the sum of TP and the number of false positives (FP). However, since we want the network to output sufficient amount of proposals that we can use for refinement, the precision will be low as there might be many false positives. Hence the precision will be not indicative at all. The recall is defined as TP over the sum of TP and the number of false negatives (FN). This metric aligns more properly with our goal, ignoring all the false positives when we have many proposals, but punishing when the bounding box proposals are missing for an object.

Task 2.2: ROI Pooling

As seen in Figure 4 the code correctly contains points to their Region of interest. For this visualization, three random frames were chosen (14, 553, 1512) and for each frame, only the first ROI and first bounding box proposal is shown.

The bounding box parameters with the layout [[x,y,z,h,w,l,ry]] are:

- Frame 14, first ROI bounding box: [[-0.643 1.7499 25.7174 1.5079 1.5986 3.568 -1.5714]]
- Frame 553, first ROI bounding box: [[-3.2182 1.8576 16.9819 1.4912 1.59 3.798 1.44]]
- Frame 1512, first ROI bounding box: [[-1.6687 1.6847 12.3009 1.533 1.6058 3.8017 1.6275]]

The code to recreate these visualizations can be found in the Appendix 2.

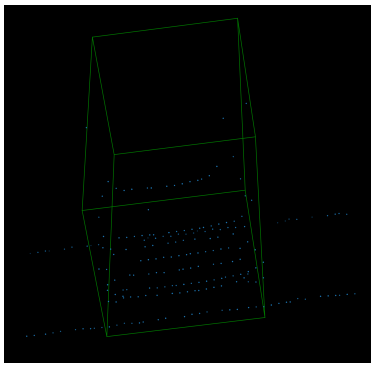


Figure 1: Frame 14, First ROI Element

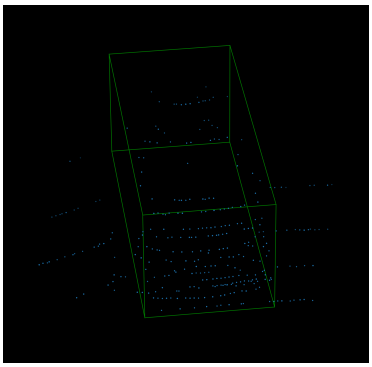


Figure 2: Frame 553, First ROI Element

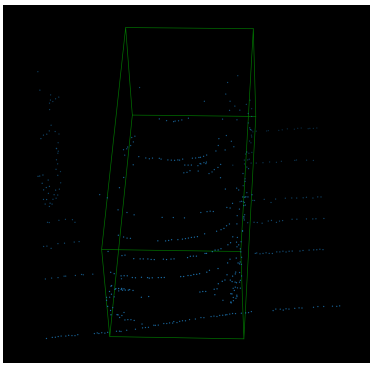


Figure 3: Frame 1512, First ROI Element

Figure 4: ROI pointclouds including their first element 3D bounding box of the three randomly chosen frames.

Task 2.3: Proposal Sampling

This visualization in Figure 5 shows the point-cloud ROI's and the filtered 3D bounding box proposals. Specifically, this image shows frame Nr. 299.

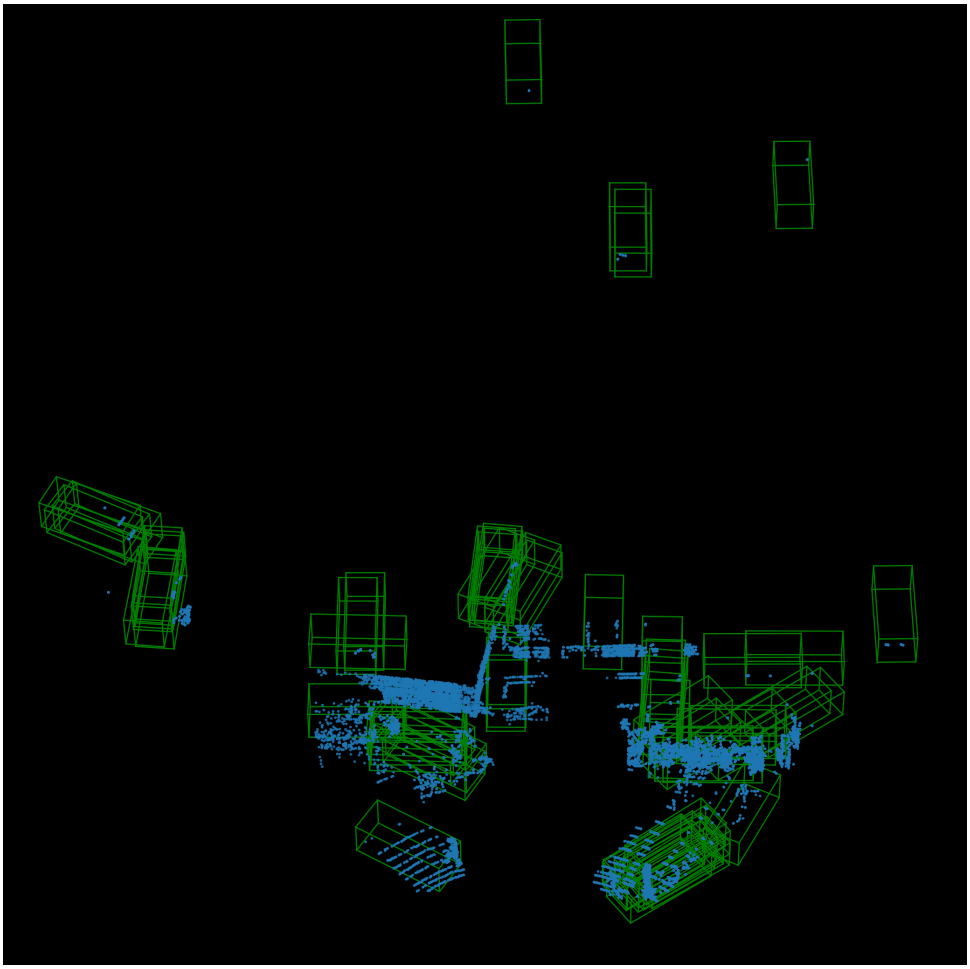


Figure 5: This figure shows the filtered pointcloud ROIs combined including the 3D bounding box proposals of frame 299.

- **Why is such a sampling scheme required?** This sampling scheme is required for the following reasons: first, the input data size must match the fixed input size of the second-stage network. Otherwise, we would have to handle a network with variable input size, which is not trivial in general. Second, as the foreground / background classes are imbalanced, this sampling scheme allows the model to not overfit on a specific class.
- **What would happen if we randomly sample all the proposals for each scene?** Random sampling of all the proposals biases background samples, as they are more abundant in autonomous driving scenes. The final data would include more background objects in expectation. Thus, after training on the imbalanced data, the network will not be able to generalize well on both the classification and regression tasks, as it would tend to predict the majority class, and fail to predict meaningful bounding boxes for the minority class of the data.
- **What if we don't sample easy background proposal at all?** If we don't sample easy background proposals, the hard background proposals that remain would contain many points that are included in the foreground proposals as well. This intersection makes it more difficult for the second stage network to learn the differences between the foreground and background samples, as it would have to predict different labels for similar input points.
- **What if we consider a sample to be a foreground if it's above 0.5 IoU and background otherwise?** Without a clear distinction between the foreground and background, two similar proposals (e.g. with an IoU difference of 0.05) are hard to categorize during training. The training procedure would thus become unstable.
- **Why do we need to match the ground truth with its highest IoU proposal?** If all predicted bounding boxes for an object have a low IoU, that object will be discarded by the

sampling scheme. However, we lose critical information by leaving it away. This means that the second-stage model trained on this dataset might not be able to identify some objects that are hard to detect, which is quite hazardous in autonomous driving scenarios.

Task 2.4: Loss Definition

The implementation of the loss function can be found in Appendix 2.

Task 2.5: Non Maximum Supression

- **Must the IoU be calculated on a BEV or can it also be calculated on 3D?** In our case, computing the 2D BEV IoU is enough to remove redundant predictions, as normally the cars are not vertically stacked on the plane. In this regard, it is not different from the 3D IoU. However, in general, the BEV IoU only considers the vehicle's length and width, ignoring its height. This limitation can make it difficult for the model to accurately differentiate between vehicles with similar footprints but different heights, such as long cars (e.g., limousines) and trucks, or regular cars and vans. Given that cars are much more prevalent than larger vehicles like trucks and vans, relying on 2D BEV IoU can lead to false negatives for these larger vehicles, as the model may misclassify them due to their height not being considered. In contrast, computing IoU in 3D can account for all dimensions—length, width, and height—providing a more accurate representation of object overlap and reducing the risk of such misclassifications.
- **What advantages does the 2D BEV IoU have over 3D (or vice versa) for NMS?** The 2D BEV IoU is computationally faster than the 3D IoU, as it involves fewer dimensions and less complexity in calculation. However, as mentioned previously, the 2D BEV IoU does not account for height variations, which can be a significant limitation in certain scenarios. For instance, in a traffic network that traverses areas with varying elevation (e.g., bridges, hills), the apparent length of vehicles in a 2D BEV might appear shorter, as the view is constrained to a flat, top-down perspective. This distortion increases the risk of misclassifying larger vehicles, such as trucks, as smaller ones, like cars. Similarly, in a highway scenario where many trucks are present, the absence of height information in the 2D BEV could lead to increased misclassification, and such misclassification risks are more dangerous. This makes 2D BEV IoU less reliable in such contexts compared to 3D IoU, which can capture both the horizontal and vertical dimensions of objects.

Task 2.6: Training the Network

Training the model took ≈ 22.5 hours. In Figure 6 the training loss over the 35 epochs can be seen.

Figure 6: Training loss of the model.

In Figure 7 the mAP performance of easy, moderate, and hard targets can be seen over the 35 epochs of training.

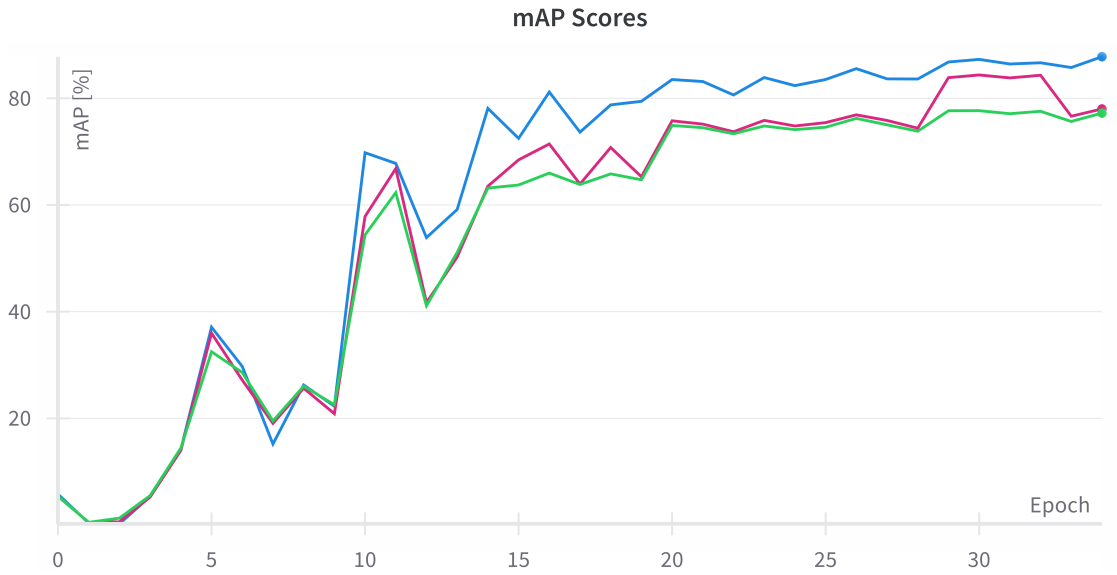


Figure 7: mAP performance for easy (blue), moderate (red) and hard (green) targets evolution during training.

Figure 11 shows the same scene at three different stages during training. In the beginning (left) halfway through (middle) and at the end of training (right).

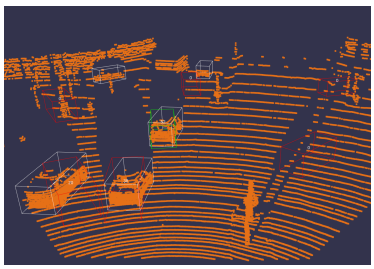


Figure 8: Example scene at epoch 0 at the beginning of the training.

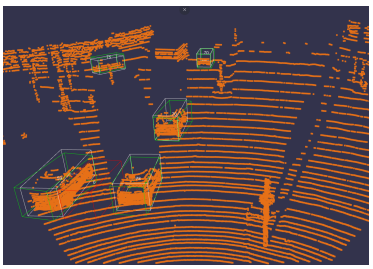


Figure 9: Example scene at epoch 16 halfway through the training.

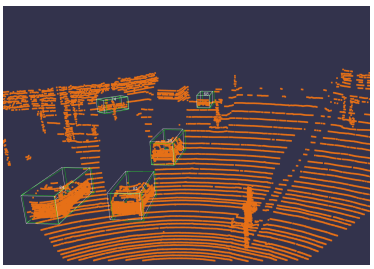


Figure 10: Example scene at epoch 34 at the end of the training.

Figure 11: Example Scene at three points during the model's training run showing the improvement of the 3D bounding boxes.

The final performance results of the trained model can be seen in Table 1.

Dataset	mAP easy	mAP moderate	mAP hard
Test (CodaLab)	85.25	75.04	72.66
Validation (Wandb)	87.80	78.00	77.23

Table 1: Comparison of mAP performance on easy, moderate and hard detections in percent. Comparison of the results between the validation set on Wandb and the test set on CodaLab.

Appendix 2: Code

Code used for Task 2.1

Only passes the IoU test with an epsilon of 0.05 (5cm) of box size enlargement.

```
import numpy as np

def label2corners(label):
    '''
    Task 1
    input
        label (N,7) 3D bounding box with (x,y,z,h,w,l,ry)
    output
        corners (N,8,3) corner coordinates in the rectified reference frame
    '''
    #make sure label is two dimensional
    label = np.array(label)
    if label.ndim == 1:
        label = label[np.newaxis, :]
    #make sure label is (N,7)
    if label.shape[1] != 7:
        label = label.T

    #reshape to make sure it is two dimensional with N,7
    label = label.reshape(-1, 7)

    N = label.shape[0]

    # Extract parameters
    x, y, z, h, w, l, ry = [label[:, i] for i in range(7)]  # Each has shape (N,)


    epsilon = 0.05
    l, w, h = l + epsilon, w + epsilon, h + epsilon

    # Define corners in the local coordinate system before rotation (shape: [N,8,3])
    x_corners = np.stack([w/2,  w/2, -w/2, -w/2,  w/2,  w/2, -w/2, -w/2], axis=1)  #
    ↪  Shape: [N,8]
    y_corners = np.stack([np.zeros(N), np.zeros(N), np.zeros(N), np.zeros(N), -h, -h,
    ↪  -h, -h], axis=1)  # Shape: [N,8]
    z_corners = np.stack([l/2, -l/2, -l/2,  l/2,  l/2, -l/2, -l/2,  l/2], axis=1)  #
    ↪  Shape: [N,8]

    corners = np.stack((x_corners, y_corners, z_corners), axis=2)  # Shape: [N,8,3]

    # Compute rotation matrices (shape: [N,3,3])
    cos_ry = np.cos(ry)
    sin_ry = np.sin(ry)

    R = np.zeros((N, 3, 3))
    R[:, 0, 0] = cos_ry
    R[:, 0, 2] = sin_ry
    R[:, 1, 1] = 1
    R[:, 2, 0] = -sin_ry
    R[:, 2, 2] = cos_ry
```

```
    # Rotate corners (shape: [N,8,3])
    corners_rotated = np.einsum('nij,nkj->nki', R, corners)

    # Translate corners
    corners_rotated += np.stack((x, y, z), axis=1)[:, np.newaxis, :]



    return corners_rotated

def get_iou(pred, target):
    '''
    Task 1
    input
        pred (N,7) 3D bounding box corners
        target (N,7) 3D bounding box corners (shouldn't this be M,7?)
    output
        iou (N,M) pairwise 3D intersection-over-union
    '''
    #get the corners
    pred_corners = label2corners(pred)
    target_corners = label2corners(target)

    #initialize iou to zeros
    iou = np.zeros((pred.shape[0], target.shape[0]))

    for i in range(pred.shape[0]):
        for j in range(target.shape[0]):
            #get the intersection
            min_corner = np.maximum(np.min(pred_corners[i], axis=0),
            ↪  np.min(target_corners[j], axis=0))
            max_corner = np.minimum(np.max(pred_corners[i], axis=0),
            ↪  np.max(target_corners[j], axis=0))

            #compute the volume of the intersection

            dx = max(0, min(max_corner[0], max_corner[0]) - max(min_corner[0],
            ↪  min_corner[0]))
            dy = max(0, min(max_corner[1], max_corner[1]) - max(min_corner[1],
            ↪  min_corner[1]))
            dz = max(0, min(max_corner[2], max_corner[2]) - max(min_corner[2],
            ↪  min_corner[2]))
            intersection = dx * dy * dz

            #compute the volume of the union
            union = np.prod(np.max(pred_corners[i], axis=0) - np.min(pred_corners[i],
            ↪  axis=0)) + np.prod(np.max(target_corners[j], axis=0) -
            ↪  np.min(target_corners[j], axis=0)) - intersection

            #compute the iou
            iou[i,j] = intersection / (union + 1e-8)

    return iou



def compute_recall(pred, target, threshold):
    '''
```

```
    Task 1
    input
        pred (N,7) proposed 3D bounding box labels
        target (M,7) ground truth 3D bounding box labels
        threshold (float) threshold for positive samples
    output
        recall (float) recall for the scene
    '''
    #get the iou
    iou = get_iou(pred, target)

    #get the maximum iou for each target
    max_iou = np.max(iou, axis=0)

    #compute the recall
    recall = np.sum(max_iou > threshold) / target.shape[0]

    return recall
```

Modified Testing Script to evaluate the recall over all 1712 datapoints:

```
# Course: Computer Vision and Artificial Intelligence for Autonomous Cars, ETH Zurich
# Material for Project 2

import os, sys, argparse
import pathlib
import psutil
import yaml
from tqdm import tqdm
sys.path.append(os.path.dirname(os.path.dirname(os.path.abspath(__file__))))
from timeit import default_timer as timer
import numpy as np
import torch

from dataset import DatasetLoader

parser = argparse.ArgumentParser(description='')
parser.add_argument('--config_path', default='config.yaml')
parser.add_argument('--recordings_dir', default='tests/recordings')
parser.add_argument('--task', type=int, default=1)
args = parser.parse_args()

ERROR_TRESHOLD = 0.002
DURATION_THRESHOLD = 1.3
LOSS_THRESHOLD = 1e-7
task2_idx = 266

class CheckTest():
        def __init__(self, config, recordings_dir):
                self.config, self.recordings_dir = config, recordings_dir
                self.ds = DatasetLoader(config['data'], 'val')

        def display_test_result(self, result, duration=False):
                result_message = 'Test passed.' if result else 'Test failed.'
```

```
                print(result_message)
                if duration:
                        print('Consider improving your implementation as this will
                        ↪  cause a bottleneck when training.')
                else:
                        print('Speed test passed.')

        def task1(self, warmup=False):
                from utils.task1 import compute_recall

                # Compute recall and duration for the same frame count
                # demo length is full length of the dataset
                demo_length = len(self.ds)
                print("number of frames", demo_length)
                recall = np.empty((demo_length,))
                duration = np.empty((demo_length,))
                pbar = tqdm(range(demo_length), disable=warmup)
                for i in pbar:
                        pred, target = self.ds.get_data(i, 'detections'),
                        ↪  self.ds.get_data(i, 'target')
                        start = timer()
                        recall[i] = compute_recall(pred, target,
                        ↪  threshold=self.config['eval']['t_rpn_recall'])
                        duration[i] = timer() - start
                        pbar.set_description('recall [%]: {:.1f}  | average duration
                        ↪  [ms]: {:.1f}'.format(
                                                                recall[:(i+1)].mean()*100,
                                                                duration[:(i+1)].mean()*1000
                # Check results
                if not warmup:
                        print("Recall mean", np.abs(recall.mean()),"Duration mean",
                        ↪  duration.mean())


def replace_username(config):
        user = os.getenv('USER') or os.getenv('USERNAME')  # This works for Linux and
        ↪  Windows
        config['data']['root_dir'] = config['data']['root_dir'].replace('$USER', user)
        config['eval']['output_dir'] = config['eval']['output_dir'].replace('$USER',
        ↪  user)
        config['trainer']['default_root_dir'] =
        ↪  config['trainer']['default_root_dir'].replace('$USER', user)
        return config

if __name__=='__main__':
        config = yaml.safe_load(open(args.config_path, 'r'))
        config = replace_username(config)
        assert(os.path.exists(args.recordings_dir))
        check = CheckTest(config, args.recordings_dir)

        getattr(check, f'task{args.task}')(warmup=True)

        getattr(check, f'task{args.task}')()
```

Code used for Task 2.2

```
import numpy as np
from utils.task1 import label2corners
from utils.vis import point_scene
import wandb
import os


def roi_pool(pred, xyz, feat, config):
    '''
    Task 2
    a. Enlarge predicted 3D bounding boxes by delta=1.0 meters in all directions.
       As our inputs consist of coarse detection results from the stage-1 network,
       the second stage will benefit from the knowledge of surrounding points to
       better refine the initial prediction.
    b. Form ROI's by finding all points and their corresponding features that lie
       in each enlarged bounding box. Each ROI should contain exactly 512 points.
       If there are more points within a bounding box, randomly sample until 512.
       If there are less points within a bounding box, randomly repeat points until
       512. If there are no points within a bounding box, the box should be
       ↪  discarded.
    input
        pred (N,7) bounding box labels
        xyz (N,3) point cloud
        feat (N,C) features
        config (dict) data config
    output
        valid_pred (K',7)
        pooled_xyz (K',M,3)
        pooled_feat (K',M,C)
            with K' indicating the number of valid bounding boxes that contain at
            ↪  least
            one point
    useful config hyperparameters
        config['delta'] extend the bounding box by delta on all sides (in meters)
        config['max_points'] number of points in the final sampled ROI
    '''
    delta = config['delta']
    max_points = config['max_points']

    # Enlarge bounding boxes
    enlarged_pred = pred.copy()
    enlarged_pred[:, 3:6] += delta * 2  # Assuming pred columns: x, y, z, h, w, l, ry

    # Precompute rotation matrices for all bounding boxes
    ry = enlarged_pred[:, 6]
    cos_ry = np.cos(ry)
    sin_ry = np.sin(ry)

    valid_pred = []
    pooled_xyz = []
    pooled_feat = []

    # Vectorized rotation matrix generation
    rotation_matrices = np.array([
        [cos_ry, np.zeros_like(cos_ry), sin_ry],
        [np.zeros_like(cos_ry), np.ones_like(cos_ry), np.zeros_like(cos_ry)],
```

```
        [-sin_ry, np.zeros_like(cos_ry), cos_ry]
    ]).transpose(2, 0, 1)


    for i in range(enlarged_pred.shape[0]):
        box = enlarged_pred[i]
        x, y, z, h, w, l, ry_i = box

        # Translate and rotate points in one step
        translated_xyz = xyz - np.array([x, y, z])
        local_xyz = np.dot(translated_xyz, rotation_matrices[i])

        # Create mask for points inside the box
        mask = (
            (local_xyz[:, 0] > (-l/2) & (local_xyz[:, 0] < (l/2) &
            (local_xyz[:, 1] > (-h/2) & (local_xyz[:, 1] < (h/2) &
            (local_xyz[:, 2] > (-w/2) & (local_xyz[:, 2] < (w/2)
        )

        in_box_indices = np.nonzero(mask)[0]

        if len(in_box_indices) < 1:
            continue

        box_xyz = xyz[in_box_indices]
        box_feat = feat[in_box_indices]

        # Sample points
        if len(box_xyz) > max_points:
            sampled = np.random.choice(len(box_xyz), max_points, replace=False)
        else:
            sampled = np.random.choice(len(box_xyz), max_points, replace=True)

        valid_pred.append(pred[i])
        pooled_xyz.append(box_xyz[sampled])
        pooled_feat.append(box_feat[sampled])

    valid_pred_arr = np.array(valid_pred)
    pooled_xyz_arr = np.array(pooled_xyz)
    pooled_feat_arr = np.array(pooled_feat)

    visualize = False
    if visualize:

        #initialize wandb
        wandb.init(project='CVAIAC-Ex2', entity='Project_1_CVAIAC')


        #reshape pooled_xyz_arr to (N,3)
        pooled_xyz_arr_reshaped = pooled_xyz_arr.reshape(-1, 3)

        #change order of x,y,z to witch to a top view
        pooled_xyz_arr_reshaped = np.array([pooled_xyz_arr_reshaped[:, 1],
        ↪  pooled_xyz_arr_reshaped[:, 0], pooled_xyz_arr_reshaped[:, 2]]).T

        scene = point_scene(pooled_xyz_arr_reshaped, valid_pred_arr, valid_pred_arr,
        ↪  name='roi_pool')
        wandb.log(scene)
```

```
    return valid_pred_arr, pooled_xyz_arr, pooled_feat_arr
```

Code used to extract single ROI visualizations:

```
# Course: Computer Vision and Artificial Intelligence for Autonomous Cars, ETH Zurich
# Material for Project 2

import os, sys, argparse
import pathlib
import psutil
import yaml
from tqdm import tqdm
sys.path.append(os.path.dirname(os.path.dirname(os.path.abspath(__file__))))
from timeit import default_timer as timer
import numpy as np
import torch


from dataset import DatasetLoader

parser = argparse.ArgumentParser(description='')
parser.add_argument('--config_path', default='config.yaml')
parser.add_argument('--recordings_dir', default='tests/recordings')
parser.add_argument('--task', type=int, default=2)
args = parser.parse_args()

ERROR_TRESHOLD = 0.002
DURATION_THRESHOLD = 1.3
LOSS_THRESHOLD = 1e-7
#three random indices on the val dataset
task2_idx = [553, 1512,14]

class CheckTest():
        def __init__(self, config, recordings_dir):
                self.config, self.recordings_dir = config, recordings_dir
                self.ds = DatasetLoader(config['data'], 'val')

        def task2(self, warmup=False):
                from utils.task2 import roi_pool
                # ROI pool for first data point
                for index in task2_idx:
                        valid_pred, pooled_xyz_array, pooled_feature_array =
                        ↪  roi_pool(pred=self.ds.get_data(index, 'detections'),
                        ↪  xyz=self.ds.get_data(index, 'xyz'),
                        ↪  feat=self.ds.get_data(index, 'features'),
                        ↪  config=self.config['data'])
                        # save the result into numpy files
                        np.save(os.path.join(self.recordings_dir,
                        ↪  f'task2_valid_pred_{index}.npy'), valid_pred)
                        np.save(os.path.join(self.recordings_dir,
                        ↪  f'task2_pooled_xyz_{index}.npy'), pooled_xyz_array)
                        np.save(os.path.join(self.recordings_dir,
                        ↪  f'task2_pooled_feature_{index}.npy'),
                        ↪  pooled_feature_array)
```

```
def replace_username(config):
        user = os.getenv('USER') or os.getenv('USERNAME')  # This works for Linux and
        ↪  Windows
        config['data']['root_dir'] = config['data']['root_dir'].replace('$USER', user)
        config['eval']['output_dir'] = config['eval']['output_dir'].replace('$USER',
        ↪  user)
        config['trainer']['default_root_dir'] =
        ↪  config['trainer']['default_root_dir'].replace('$USER', user)
        return config

if __name__=='__main__':
        config = yaml.safe_load(open(args.config_path, 'r'))
        config = replace_username(config)
        assert(os.path.exists(args.recordings_dir))
        check = CheckTest(config, args.recordings_dir)

        getattr(check, f'task{args.task}')()
```

Code to visualize single ROI with bounding boxes:

```
import numpy as np
import matplotlib.pyplot as plt
from load_data import load_data
from vis import project_point_cloud

def render_point_cloud(data, only_fov,only_car,frame_nr,output_filename='output.png'):
    # get correct colors for each point
    colors = np.array([data["color_map"][label][::-1] for label in
    ↪  data["sem_label"].squeeze()], dtype=np.float32)
    colors /= 255.

    # get points from frame
    #load numpy array file
    points = np.load(f'./data/task2_pooled_xyz_{frame_nr}.npy')
    points = points[0]

    points = points.reshape(-1, 3)

    #switch axes
    #points = np.array([points[:, 0], points[:, 2], points[:, 1]]).T



    #load bounding box parameters
    bounding_boxes = np.load(f'./data/task2_valid_pred_{frame_nr}.npy')


    bounding_boxes = bounding_boxes[0]

    bounding_boxes = bounding_boxes.reshape(-1, 7)
```

```
print("Bounding Box data: ", bounding_boxes)


#used if only showing points that are in cam_2's FoV
if only_fov:
    # add 1s for homogeneous coordinates
    points[:, 3] = 1

    # Project points to image plane
    projected_points = project_point_cloud(data, points)

    # Get image dimensions
    image = data["image_2"]
    image_shape = image.shape

    # Filter points within image boundaries
    keep = (projected_points[:, 0] >= 0) & (projected_points[:, 0] <
    ↪  image_shape[1]) & \
           (projected_points[:, 1] >= 0) & (projected_points[:, 1] <
           ↪  image_shape[0]) & \
           (points[:, 0] > 0)
    points = points[keep]
    colors = colors[keep]


# Render 3D point cloud
#fullscreen
fig = plt.figure(figsize=(12, 8))
ax = fig.add_subplot(111, projection='3d')

# Plot point cloud
ax.scatter(points[:, 0], points[:, 1], points[:, 2],
           s=10)

# Render bounding boxes
#objects = data["objects"]
T_inv = np.linalg.inv(data["T_cam0_velo"])

for obj in bounding_boxes:
    dimensions = obj[3:6] #height, width, length
    location = obj[0:3] # x, y, z in cam0 coordinates
    rotation = obj[6] #rotation angle of box


    #(x,y,z,h,w,l,ry) is the bounding_boxes

    h, w, l = dimensions

    # Define the eight corners of the bounding box in the object's coordinate
    ↪  frame
    x_corners = [ l/2,  l/2, -l/2, -l/2,  l/2,  l/2, -l/2, -l/2]
    y_corners = [ 0,    0,    0,    0,   -h,   -h,   -h,   -h]
    z_corners = [ w/2, -w/2, -w/2,  w/2,  w/2, -w/2, -w/2,  w/2]

    # Combine into a numpy array
    corners = np.array([x_corners, y_corners, z_corners])

    # Define the rotation matrix around the Y-axis
```

```
        R = np.array([
            [np.cos(rotation), 0, np.sin(rotation)],
            [0, 1, 0],
            [-np.sin(rotation), 0, np.cos(rotation)]
        ])

        # Rotate the corners
        corners_rotated = np.dot(R, corners)

        # Translate the corners to the object's location in cam0
        corners_rotated[0, :] += location[0]
        corners_rotated[1, :] += location[1]
        corners_rotated[2, :] += location[2]


        # Convert to homogeneous coordinates
        corners_hom = np.vstack((corners_rotated, np.ones((1,
        ↪  corners_rotated.shape[1]))))

        # Transform from cam0 to velo coordinates using T_inv
        #corners_velo_hom = np.dot(T_inv, corners_hom)
        corners_velo_hom = corners_hom
        corners_velo = corners_velo_hom[:3, :].T  # Shape: (8, 3)

        # Define the edges of the bounding box
        edges = [
            [0, 1], [1, 2], [2, 3], [3, 0],  # Top face
            [4, 5], [5, 6], [6, 7], [7, 4],  # Bottom face
            [0, 4], [1, 5], [2, 6], [3, 7]   # Vertical edges
        ]

        # Plot each edge of the bounding box
        for edge in edges:
            ax.plot(*corners_velo[edge, :].T, color='green', linewidth=2)

    ax.set_xlabel('X')
    ax.set_ylabel('Y')
    ax.set_zlabel('Z')
    ax.set_title('Point Cloud Visualization')

    #make background black
    ax.set_facecolor('black')
    #remove walls of the box for all sides
    ax.xaxis.pane.fill = False
    ax.yaxis.pane.fill = False
    ax.zaxis.pane.fill = False
    #remove grid
    ax.grid(False)
    #remove axis
    ax.axis('off')
    #keep axis scales equal to values
    ax.set_box_aspect([np.ptp(points[:, 0]), np.ptp(points[:, 1]), np.ptp(points[:,
    ↪  2])])

    #center plot on the middle of the point cloud
    ax.auto_scale_xyz(points[:, 0], points[:, 1], points[:, 2])
```

```
    plt.tight_layout()
    plt.show()
    #plt.savefig(output_filename, dpi=300, bbox_inches='tight')
    #plt.close()

if __name__ == '__main__':
    data = load_data('data/data.p')
    # Defines if only points within the camera's field of view should be rendered
    only_fov = False
    # Filter out all points except those that belong to the "car" class
    only_car = False


    frame_nr = 553
    render_point_cloud(data,only_fov, only_car, frame_nr)
```

Code used for Task 2.3

```python
import numpy as np

from .task1 import get_iou

def sample_proposals(pred, target, xyz, feat, config, train=False):
    '''
    Task 3
    a. Using the highest IoU, assign each proposal a ground truth annotation. For each
    ↪  assignment also
       return the IoU as this will be required later on.
    b. Sample 64 proposals per scene. If the scene contains at least one foreground
    ↪  and one background
       proposal, of the 64 samples, at most 32 should be foreground proposals.
       ↪  Otherwise, all 64 samples
       can be either foreground or background. If there are less background proposals
       ↪  than 32, existing
       ones can be repeated.
       Furthermore, of the sampled background proposals, 50% should be easy samples
       ↪  and 50% should be
       hard samples when both exist within the scene (again, can be repeated to pad up
       ↪  to equal samples
       each). If only one difficulty class exists, all samples should be of that
       ↪  class.
    input
        pred (N,7) predicted bounding box labels
        target (M,7) ground truth bounding box labels
        xyz (N,512,3) pooled point cloud
        feat (N,512,C) pooled features
        config (dict) data config containing thresholds
        train (string) True if training
    output
        assigned_targets (64,7) target box for each prediction based on highest iou
        xyz (64,512,3) indices
        feat (64,512,C) indices
        iou (64,) iou of each prediction and its assigned target box
    useful config hyperparameters
        config['t_bg_hard_lb'] threshold background lower bound for hard difficulty
        config['t_bg_up'] threshold background upper bound
        config['t_fg_lb'] threshold foreground lower bound
        config['num_fg_sample'] maximum allowed number of foreground samples
        config['bg_hard_ratio'] background hard difficulty ratio (#hard samples/
        ↪  #background samples)
    '''

    t_bg_hard_lb = config['t_bg_hard_lb']
    t_bg_up = config['t_bg_up']
    t_fg_lb = config['t_fg_lb']
    num_fg_sample = config['num_fg_sample']
    bg_hard_ratio = config['bg_hard_ratio']

    # Get the IoU for each combination of pred and target
    iou = get_iou(pred, target)  # iou = (N, M)

    # Get the highest IoU and corresponding target index for each prediction
    max_iou = np.max(iou, axis=1)  # max_iou = (N,)
    max_iou_idx = np.argmax(iou, axis=1)  # max_iou_idx = (N,)
```

```
    assigned_targets = target[max_iou_idx]  # assigned_targets = (N, 7)

    # Classify proposals
    fg_mask = max_iou >= t_fg_lb
    hard_bg_mask = (max_iou >= t_bg_hard_lb) & (max_iou < t_fg_lb)
    easy_bg_mask = max_iou < t_bg_hard_lb

    fg_indices = np.where(fg_mask)[0]
    hard_bg_indices = np.where(hard_bg_mask)[0]
    easy_bg_indices = np.where(easy_bg_mask)[0]

    N_fg = len(fg_indices)
    N_bg_hard = len(hard_bg_indices)
    N_bg_easy = len(easy_bg_indices)

    has_fg = N_fg > 0
    has_bg = (N_bg_hard + N_bg_easy) > 0

    selected_indices = []

    if has_fg and has_bg:
        num_fg_samples = min(N_fg, num_fg_sample, 32)
        total_bg_samples = 64 - num_fg_samples

        # Distribute background samples
        if N_bg_hard > 0 and N_bg_easy > 0:
            num_bg_hard = total_bg_samples // 2
            num_bg_easy = total_bg_samples - num_bg_hard
        elif N_bg_hard > 0:
            num_bg_hard = total_bg_samples
            num_bg_easy = 0
        elif N_bg_easy > 0:
            num_bg_hard = 0
            num_bg_easy = total_bg_samples
        else:
            num_bg_hard = 0
            num_bg_easy = 0

        # Sample foreground proposals
        fg_sample_indices = np.random.choice(
            fg_indices, size=num_fg_samples, replace=(num_fg_samples > N_fg))
        selected_indices.extend(fg_sample_indices)

        # Sample hard background proposals
        if num_bg_hard > 0:
            bg_hard_sample_indices = np.random.choice(
                hard_bg_indices, size=num_bg_hard, replace=(num_bg_hard > N_bg_hard))
            selected_indices.extend(bg_hard_sample_indices)

        # Sample easy background proposals
        if num_bg_easy > 0:
            bg_easy_sample_indices = np.random.choice(
                easy_bg_indices, size=num_bg_easy, replace=(num_bg_easy > N_bg_easy))
            selected_indices.extend(bg_easy_sample_indices)

    elif has_fg and not has_bg:
        num_fg_samples = min(N_fg, 64)
        selected_indices = np.random.choice(
```

```
            fg_indices, size=64, replace=(64 > N_fg)).tolist()

    elif not has_fg and has_bg:
        total_bg_samples = 64

        if N_bg_hard > 0 and N_bg_easy > 0:
            num_bg_hard = total_bg_samples // 2
            num_bg_easy = total_bg_samples - num_bg_hard
        elif N_bg_hard > 0:
            num_bg_hard = total_bg_samples
            num_bg_easy = 0
        elif N_bg_easy > 0:
            num_bg_hard = 0
            num_bg_easy = total_bg_samples
        else:
            num_bg_hard = 0
            num_bg_easy = 0

        # Sample hard background proposals
        if num_bg_hard > 0:
            bg_hard_sample_indices = np.random.choice(
                hard_bg_indices, size=num_bg_hard, replace=(num_bg_hard > N_bg_hard))
            selected_indices.extend(bg_hard_sample_indices)

        # Sample easy background proposals
        if num_bg_easy > 0:
            bg_easy_sample_indices = np.random.choice(
                easy_bg_indices, size=num_bg_easy, replace=(num_bg_easy > N_bg_easy))
            selected_indices.extend(bg_easy_sample_indices)
    else:
        # If neither foreground nor background proposals exist, sample randomly
        selected_indices = np.random.choice(
            len(pred), size=64, replace=(64 > len(pred))).tolist()

    # Ensure we have exactly 64 samples
    selected_indices = np.array(selected_indices)

    if len(selected_indices) < 64:
        num_missing = 64 - len(selected_indices)
        extra_indices = np.random.choice(
            selected_indices, size=num_missing, replace=True)
        selected_indices = np.concatenate([selected_indices, extra_indices])
    elif len(selected_indices) > 64:
        selected_indices = np.random.choice(
            selected_indices, size=64, replace=False)

    # Shuffle the selected indices
    np.random.shuffle(selected_indices)

    # Gather the outputs
    assigned_targets = assigned_targets[selected_indices]
    xyz = xyz[selected_indices]
    feat = feat[selected_indices]
    iou = max_iou[selected_indices]

    return assigned_targets, xyz, feat, iou
```

Code used for Task 2.4

```python
import torch
import torch.nn as nn
import os

class RegressionLoss(nn.Module):
    def __init__(self, config):
        super().__init__()
        self.config = config
        self.loss = nn.SmoothL1Loss(reduction='none')

    def forward(self, pred, target, iou):
        '''
        Task 4.a
        We do not want to define the regression loss over the entire input space.
        While negative samples are necessary for the classification network, we
        only want to train our regression head using positive samples. Use 3D
        IoU >= 0.55 to determine positive samples and alter the RegressionLoss
        module such that only positive samples contribute to the loss.
        input
            pred (N,7) predicted bounding boxes
            target (N,7) target bounding boxes
            iou (N,) initial IoU of all paired proposal-targets
        useful config hyperparameters
            self.config['positive_reg_lb'] lower bound for positive samples
        '''
        positive_reg_lb = self.config['positive_reg_lb']

        positive_mask = iou >= positive_reg_lb
        pred = pred[positive_mask]
        target = target[positive_mask]
        # Calculate the loss for each dimension
        loss = self.loss(pred, target)

        location_loss = loss[:, :3].mean(dim=0).mean()
        dimension_loss = loss[:, 3:6].mean(dim=0).mean()
        orientation_loss = loss[:, 6].mean(dim=0).mean()
        loss = location_loss + 3 * dimension_loss + orientation_loss


        #AFTER CHECKING THE DIFFERENCE BETWEEN RECORDED LOSS AND MY LOSS i CONCLUDE
        ↪  THEY ARE THE SAME
        #AND USING == TO COMPARE THEM IS NOT A GOOD IDEA
        #recorded_loss = torch.load(os.path.join("./tests/recordings",
        ↪  'task4_reg_loss.pt'))

        #print("losses are identical: ", loss == recorded_loss)
        #print("losses difference: ", abs(loss - recorded_loss))
        #print("recorded_loss", recorded_loss)
        #print("loss", loss)

        return loss

class ClassificationLoss(nn.Module):
    def __init__(self, config):
        super().__init__()
        self.config = config
```

```
        self.loss = nn.BCELoss()

    def forward(self, pred, iou):
        '''
        Task 4.b
        Extract the target scores depending on the IoU. For the training
        of the classification head we want to be more strict as we want to
        avoid incorrect training signals to supervise our network.  A proposal
        is considered as positive (class 1) if its maximum IoU with ground
        truth boxes is >= 0.6, and negative (class 0) if its maximum IoU <= 0.45.
            pred (N,7) predicted bounding boxes
            iou (N,) initial IoU of all paired proposal-targets
        useful config hyperparameters
            self.config['positive_cls_lb'] lower bound for positive samples
            self.config['negative_cls_ub'] upper bound for negative samples
        '''

        positive_cls_lb = self.config['positive_cls_lb']
        negative_cls_ub = self.config['negative_cls_ub']

        positive_mask = iou >= positive_cls_lb
        negative_mask = iou <= negative_cls_ub

        selected_mask = positive_mask | negative_mask

        pred_filtered = pred[selected_mask]
        iou_filtered = iou[selected_mask]
        target = (iou_filtered >= positive_cls_lb).float()

        # Ensure target and pred_filtered have the same shape
        if pred_filtered.dim() > target.dim():
            target = target.unsqueeze(1)

        loss = self.loss(pred_filtered, target)
        return loss
```

Code used for Task 2.5

```python
import numpy as np

import os
from utils.task1 import get_iou

def nms(pred, score, threshold):
    '''
    Task 5
    Implement NMS to reduce the number of predictions per frame with a threshold
    of 0.1. The IoU should be calculated only on the 2D BEV.
    input
        pred (N,7) 3D bounding box with (x,y,z,h,w,l,ry)
        score (N,) confidence scores
        threshold (float) upper bound threshold for NMS
    output
        s_f (M,7) 3D bounding boxes after NMS
        c_f (M,1) corresopnding confidence scores
    '''
    # set z to 0 and h to 1
    bev_boxes = pred.copy()
    bev_boxes[:, 1] = 0
    bev_boxes[:, 3] = 1

    # Sort the scores in descending order
    sorted_indices = np.argsort(score)[::-1]

    # Initialize the list of selected indices
    selected_indices = []

    while len(sorted_indices) > 0:
        selected_index = sorted_indices[0]
        selected_indices.append(selected_index)

        sorted_indices = sorted_indices[1:]  # Remove the selected index
        if len(sorted_indices) == 0:
            break

        #modify the input of get_iou to be (1,7) and (M,7)
        chosen_box = bev_boxes[selected_index].T.reshape(1,7)
        compared_boxes = bev_boxes[sorted_indices]


        # Calculate the IoU with the rest of the boxes
        iou = get_iou(chosen_box, compared_boxes)

        # Keep boxes with IoU less than or equal to the threshold
        indices_to_keep = np.where(iou < threshold)[1]
        sorted_indices = sorted_indices[indices_to_keep]

    s_f = pred[selected_indices]
    c_f = np.expand_dims(score[selected_indices], axis=1)

    return s_f, c_f
```