ME 292B Final Project

Hao Ouyang, Shuang Shao, Yuqing Zhang, Carol Guo

Department of Mechanical Engineering, UC Berkeley.

1 Introduction

1.1 Problem Statement

Recently, autonomous driving technology has been rapidly developing, with the goal of improving road safety and reducing accidents caused by human error. A crucial component of the autonomous driving system is object detection, which enables vehicles to detect and respond to their surroundings. In this project, the team aimed to develop a data-science-based solution for object detection using a combination of camera and LiDAR sensors.

The camera captures the visual image of the environment and provides information about the appearance and texture of the object. On the other hand, The LiDAR uses lasers to measure the distance to an object and create a 3D map of its environment. By combining the data from the two sensors, the location and the type of the object can be determined, which is essential for autonomous vehicles.

1.2 Approach

The main objective of the project is to design a data-processing pipeline that can efficiently and accurately identify objects in real-time. Machine learning algorithms, including deep neural networks, are used to classify the object based on the image captured by the camera. The data obtained by the LiDAR is used to obtain the precise coordinates of the object, which can be used for localization and path planning. Finally, a matching algorithm is designed to fuse the data of the two sensors to obtain a comprehensive cognition of the environment. The proposed solution has the potential to significantly improve the safety and reliability of autonomous driving systems, which can help the vehicle to detect and avoid obstacles, such as other vehicles and pedestrians.

In the following sections, the technical details of the approach, including data acquisition, data preprocessing, detector design, and sensor fusion, will be discussed.

2 Methods

2.1 Data Acquisition

In this project, The camera detector captures a video dataset in `.avi` format, showing the movement of two humans, the truck, the building, and the pier. The `.avi` format is a widely used video file format that supports high-quality video with relatively small file size, making it convenient for storage and analysis.

Fig. 1: Video Overview

The LiDAR detector measures the distance to objects in the scene using lasers and generates a `.pcd` file containing the 3D coordinates of the objects detected. The `.pcd` file is a common file format used for storing point cloud data and can be used for various applications such as 3D reconstruction, object detection, and localization. In this project, the point cloud data captured by the LiDAR detector provides information for understanding the scene and objects in it from a 3D perspective. Each point in the point cloud represents a point in 3D space and contains information such as its x, y, and z coordinates, as well as the intensity and other optional attributes.

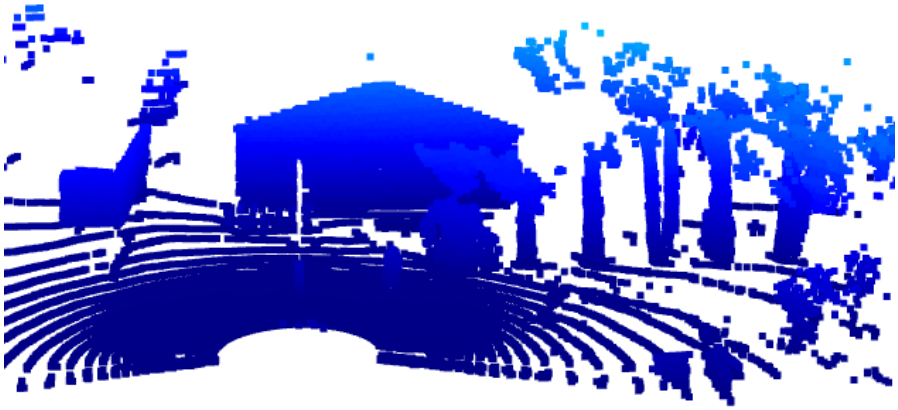


Fig. 2: PCD File Overview

2.2 Data Preprocessing

The team performed data preprocessing on the data obtained from the LiDAR, while the data obtained from the camera did not require any preprocessing. This is because the input for target detection using the camera is generally in the form of videos or images, which do not require any preprocessing procedures. However, the `.pcd` data from the LiDAR contains numerous point clouds, and the clustering process demands substantial computing resources, which can also be time-consuming. Therefore, the team filtered out irrelevant areas, such as the ground, to limit the processing area and enhance the processing speed.

The first step is to down-sample the point cloud data using voxel sampling, which divides the point cloud space into cubic voxels. This process samples the center point of each voxel as the representative point for that voxel. If multiple points exist within a voxel, they can be merged into one point or randomly selected. Adjusting the size of the voxel controls the resolution of the point cloud, with smaller voxels resulting in higher resolution. In this project, a voxel grid size of 0.05 meters was chosen to balance information retention and computational efficiency.

Next, the region of interest (ROI) is defined as a 3D box with a maximum bounding point of (-10, -10, -2) and a minimum bounding point of (10, 10, 2), encompassing all target elements.

The ground segmentation step involved removing the ground points from the data. To achieve this, the team utilized the existing `plane segmentation` function in the Python open3D library, which uses the Random Sample Consensus (RANSAC) method to process raw data and acquire a plane. The RANSAC algorithm selects a subset of points to fit a plane and then tests all the remaining points to determine

whether they are inliers or outliers based on a threshold distance. Points that are within a certain distance threshold from the fitted plane are considered as inliers, while those outside the threshold are considered outliers. It then selects a new subset of inliers and repeats the process until the desired number of inliers is obtained or the maximum iteration limit is reached. The fitted plane is then considered the ground. To use the `plane segmentation` function, the team inputs the maximum distance between an inlier and the fitted plane, the minimum number of points of the plane, and the number of iterations. The function provides four numbers that define the plane function in the 3D coordinate system and the set of inliers. However, the team needed the outlier points and therefore took the inverse of the inliers. The advantage of using the RANSAC method is that it considers the influence of noise to a large extent. In this case, the team set the distance threshold to 0.1 m, the iteration limit to 3, and the minimum number of points to 1000.

2.3 Detector Design

In the following sections, concise overviews of the camera detector and LiDAR detector will be discussed.

2.3.1 Camera Detector

The team utilized Convolutional Neural Networks (CNN) to detect objects in the video data. CNNs are a type of deep learning algorithm that has revolutionized the field of computer vision by allowing machines to recognize patterns and features in images. Unlike traditional neural networks, CNNs have convolutional layers that can automatically learn spatial hierarchies of features from the raw pixel values of the input images [1]. The main goal of the team's object detection task using CNN was to identify whether any frame in the video data contained a person. If a person is detected, the CNN would output the position of the rectangle containing the person. This position information was then used for sensor fusion in subsequent steps of the analysis.

The team used YOLO Nano for object detection, based on the YOLO algorithm that achieves real-time object detection with a single neural network pass. YOLO Nano shares a similar neural network structure with other YOLO versions, but it differs by using fewer convolutional layers and parameters to reduce computational complexity, making it a suitable choice for resource-constrained devices such as mobile devices and embedded systems. [2].

The YOLO Nano network consists of EP, PEP, and FCA modules, as shown in Figure 3. These modules are designed to extract features from the input image and classify objects in the image. Here is a brief overview of the modules and what they do:

- EP module (Expand and Project module): The EP module consists of two convolutional layers that are used to expand the number of channels in the feature maps

and then project them back to the original number of channels. The purpose of this module is to capture more complex and abstract features in the input image.

- PEP module (Project, Expand, and Pool module): The PEP module combines three operations: projection, expansion, and pooling. The module first projects the feature maps to a lower dimension, then expands them to a higher dimension, and finally applies max pooling to the expanded feature maps. This module is designed to capture more detailed features in the input image.

- FCA module (Fully Connected Attention module): The FCA module uses a fully connected layer to perform global pooling of the feature maps and then uses the resulting vector to compute attention weights for each channel in the feature maps. The attention weights are used to selectively amplify or suppress different channels in the feature maps, which helps the network focus on the most relevant features for object detection.

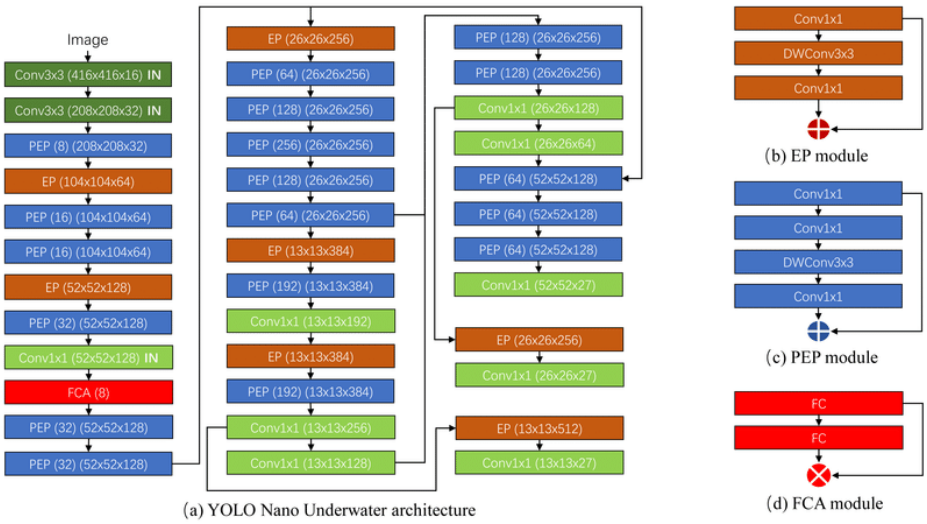


Fig. 3: YOLO Nano Network Architecture[2]

In terms of classification, the YOLO Nano network uses a final set of convolutional and fully connected layers to generate object detection predictions. The output of the network is a set of bounding boxes and class probabilities for each object in the input image.

Based on the description of the YOLO Nano network structure, the team was able to summarize the steps that CNN takes to generate an output containing a set of bounding boxes and class probabilities for each object in an input image as follows:

1. Feature Extraction: The input image is fed into the network and processed through a series of convolutional layers, such as the EP and PEP modules, which extract different levels of features from the image. The extracted features are then passed through a fully connected attention (FCA) module, which selects and weights the most relevant features for object detection.

2. Object Proposal: The feature maps are then used to generate a set of object proposals. These proposals are potential bounding boxes that may contain objects in the image. This is done using a set of anchor boxes, which are predefined boxes of different sizes and aspect ratios.

3. Object Classification: The proposals are then classified as either containing an object or not. This is done using a classification network that takes the feature maps and the proposed bounding boxes as input. The classification network outputs a probability score for each proposed bounding box, indicating the likelihood that it contains an object.

In addition to network architecture, the choice of activation functions, loss functions, and optimization methods also has a significant impact on the effectiveness of CNN algorithms. In the case of YOLO Nano, it employs specific activation functions, loss functions, and optimization methods, which will be discussed in detail below.

Leaky ReLU is used as the activation function in YOLO Nano. It is similar to the traditional ReLU function but allows a small, non-zero gradient when the input is negative.[3] This helps prevent the problem of "dying ReLU", where neurons can become inactive and stop learning during training. The relevant formulas are shown below, and the corresponding image is shown in Figure 4.

$$ReLU(x) = \max(0, x) = \begin{cases} x, & \text{if } x > 0 \\ 0, & \text{otherwise} \end{cases} \tag{1}$$

$$LeakyReLU(x) = \max(\alpha x, x) = \begin{cases} x, & \text{if } x > 0 \\ \alpha x, & \text{otherwise} \end{cases} \tag{2}$$

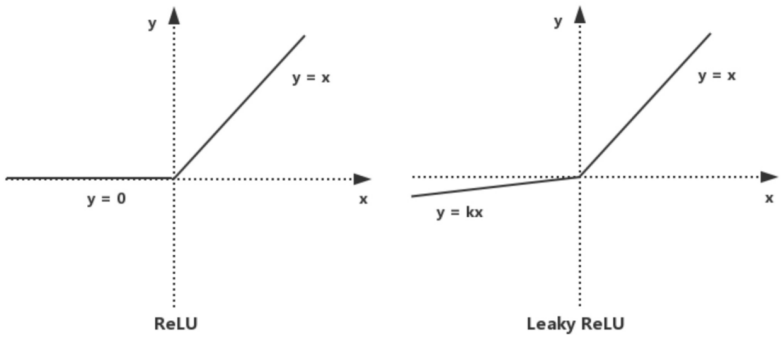


Fig. 4: Comparison: ReLU and Leaky ReLU [3]

Focal Loss is a specialized loss function that is used in YOLO Nano for object detection.[4] It is designed to address the problem of class imbalance, where the number of negative examples far outweighs the number of positive examples. Focal Loss puts more emphasis on hard examples that are misclassified, which helps the network focus on learning to classify objects better. It is defined as:

$$L_{FL}(p_t) = -\alpha_t(1-p_t)^{\gamma}\log(p_t) \tag{3}$$

where p_t is the predicted probability of the true class. α_t is the balancing factor that assigns a weight to each class based on its frequency. γ is the focusing parameter that controls the degree of emphasis on examples. The α_t term is defined as:

$$\alpha_t = \begin{cases} 1-\alpha & \text{if } y_t = 1 \\ \alpha & \text{otherwise} \end{cases} \tag{4}$$

where y_t is the ground truth label for the true class, α is a hyperparameter that controls the weighting of the positive and negative examples

Regarding the optimization process, YOLO Nano utilized the Adam optimization algorithm. Adam is a widely-used optimization algorithm that dynamically adapts the learning rate for each parameter based on the gradient and the previous update history.[5] This optimization algorithm is well-suited for training deep neural networks like YOLO Nano, as it enables the network to converge more rapidly and avoid getting trapped in local minima.

For the training and inference process, the training process of YOLO Nano is similar to other neural networks, using forward and backward propagation to calculate loss and update weights. In the inference stage, YOLO Nano predicts the object positions and categories in the input image by performing forward propagation. Since the team only detected people in the video, which is one of the most important classes in the object detection field, pre-trained weight files on the COCO dataset typically include

detection for the "person" class. Therefore, the team did not collect new data for training, but used pre-trained weights on the COCO dataset for inference.

2.3.2 LiDAR Detector

The team developed a clustering function called `cluster` to group together points in the point cloud. The function takes three parameters: `cluster_threshold`, `min_size`, and `max_size`. The `cluster_threshold` is a floating-point number that sets the distance threshold for clustering points in the point cloud. The `min_size` is an integer that sets the minimum number of points required to form a cluster, while the `max_size` is an integer that sets the maximum number of points allowed in a cluster.

To cluster the points in the point cloud, the function uses the DBSCAN (Density-Based Spatial Clustering of Applications with Noise) algorithm. This algorithm automatically finds the number of clusters based on the data and parameters given. It works by defining a distance threshold called the epsilon parameter (`eps`) and a minimum number of points required to form a cluster called the minimum points parameter (`min_size`). The algorithm starts by selecting a random point in the dataset and checks if it has at least `min_size` neighbors within `eps` distance. If it does, it forms a cluster with those neighbors. The algorithm repeats this process for each neighbor point in the cluster until all points in the cluster are identified. If a point does not have enough neighbors to form a cluster, it is labeled as noise and is not part of any cluster. [6]

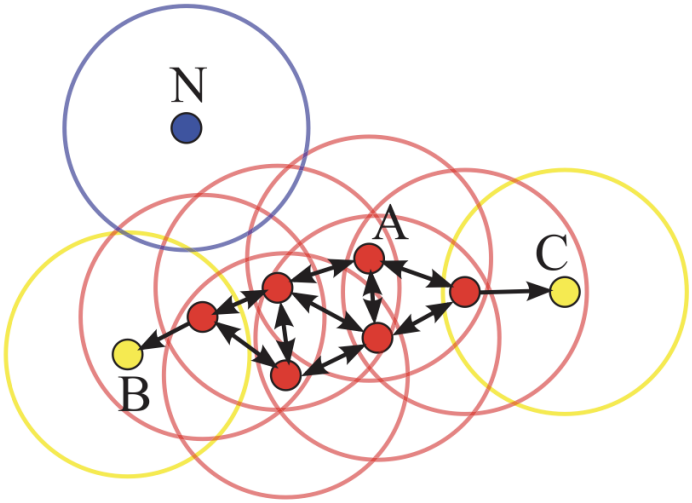


Fig. 5: DBSCAN Cluster Model [6]

Figure 5 serves as an illustrative example of the DBSCAN (Density-Based Spatial Clustering of Applications with Noise) cluster model. The circles depicted in the figure

represent the neighborhoods of each data point, wherein the radius of each circle is determined by the parameter `eps`. In addition, the parameter `min_size` is set to 4, such that any circle containing at least four points will classify the centered point as a core point. Upon selection of a starting point at random, DBSCAN proceeds to identify whether it is a core point. If not, such as point N, that point is categorized as noise, and another point is selected randomly. However, if it is a core point, such as point A, the algorithm expands the cluster by identifying its neighboring points as either core or edge points. Edge points, such as points B and C, fall within a circle formed by a core point; on the other hand, the circles formed by these points do not contain enough points to classify them as core points. In the case of edge point neighbors, the algorithm does not expand the cluster further. However, if the neighbor is a core point, all of its neighbors must also be identified. The clustering procedure continues until all the points in the `.pcd` file are processed.

In this project, the DBSCAN clustering algorithm was employed with specific parameter settings. The parameter `eps` was assigned a value of 0.5, while `min_size` was set to 30. Moreover, the team introduced the concept of `max_size`, which was set to 1000. This parameter was utilized to exclude large objects from the analysis, such as houses and trucks, which were deemed irrelevant. If the number of points in a cluster exceeded the `max_size` value, the entire cluster was labeled as noise and excluded from the final output. The implementation of the clustering algorithm was conducted using the `cluster_dbscan` function from the Open3D library, which returns a list of clusters.

The team also implemented a function named `create_bounding_box`, which takes a list of point cloud clusters as input. The function iterates through each cluster in the list, converts the point cloud to a NumPy array of points, creates an axis-aligned bounding box around the point cloud cluster using the `create_from_points` method from the Open3D library, obtains the 8 corner points of the bounding box using the `get_box_points` method, and finally appends the corner points to a list called `self.results`. These corner points can be used to visualize the bounding boxes around the objects in the point cloud clusters.

2.4 Sensor Fusion

After obtaining the detection results from the camera and the LiDAR, the team needed to fuse these data and match the corresponding bounding boxes. The detailed algorithm for this process is demonstrated below:

1. Transform every vertex of the LiDAR bounding box from the LiDAR coordinate system to the camera coordinate system using the transformation matrix T.

$$T = \begin{bmatrix} R & t \\ 0 & 1 \end{bmatrix}, \tag{5}$$

   where R represents a 3×3 rotation matrix, and t represents a 3×1 translation vector. Then, use the camera intrinsic parameters K to project the vertices from

camera coordinates to pixel coordinates.

$$K = \begin{bmatrix} f_x & 0 & c_x & 0 \\ 0 & f_y & c_y & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix}. \tag{6}$$

f_x and f_y represent the focal length of the camera, i.e. the distance between the camera and the image plane. c_x and c_y represent the position of the origin of the image plane in the pixel coordinate system.

2. For each LiDAR bounding box, calculate its outer rectangle based on the pixel coordinates of its eight vertices. The output is a list of coordinates of the upper-left and lower-right corner points of these bounding boxes.

3. Compute the Intersection over Union (IoU) metric for each pair of the camera bounding box and the LiDAR bounding box to measure the overlap between these two bounding boxes.

$$IoU = \frac{A \cap B}{A \cup B}, \tag{7}$$

where A and B are two bounding boxes from different sensors, respectively. Then use these IoU values to construct a cost matrix.

$$cost = 1 - IoU \tag{8}$$

4. Based on the cost matrix, perform linear assignment to assign each camera bounding box to the most likely LiDAR bounding box. Here, the team chose to use `scipy.optimize.linear_sum_assignment` function based on the Hungarian algorithm. The principle of the Hungarian algorithm is to iteratively find the minimum cost assignment of objects from one set to another, until all objects have been assigned. By utilizing this algorithm, the team was able to obtain the best possible result for matching camera and LiDAR detection results.

Eventually, this sensor fusion algorithm returned the result of matching camera and LiDAR detection results. By combining the results from the camera and LiDAR, the team was able to determine the spatial coordinates of each bounding box detected by the camera.

3 Results

3.1 Camera Result

The results showed that YOLO Nano provided a powerful balance between accuracy, size, and computational complexity. The model that the team implemented did not

require GPU and CUDA support and can run directly on a CPU with a running speed of 11 FPS (based on an 8th generation Core i7 CPU), making it very convenient to deploy on any computer or even embedded platform.

The detection results of the dataset are shown in Figure 6. In the absence of ground truth, the team cannot reflect the performance of the detector by calculating mAP. However, in the detection of the entire video stream, the team found that YOLO Nano can successfully detect every person in each frame of the image, achieving a confidence level of around 80% in single-person detection and around 70% in multi-person detection. This demonstrates the excellent performance of the camera detector and provides a good foundation for subsequent sensor fusion.

(a) Single-person detection (b) Multi-person detection

Fig. 6: Camera Result: Single-person and multi-person detection

3.2 LiDAR Result

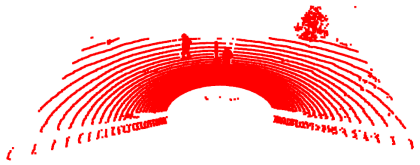
Figure 7 presents the LiDAR results before and after the ground segmentation and clustering process, for two point cloud files, where a person is away from and near the pier.

After performing down-sampling and ROI filtering, the team successfully captured the desired scene, which included two people, a tree, and a pier. The accuracy achieved was sufficient to recognize the outlines of these objects with the naked eye, indicating that the parameters used in the down-sampling process were appropriate.

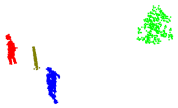
The ground segmentation process was successful in removing the ground from the dataset and retaining only the two persons, the tree, and the pier.

The clustering process divided the point cloud into distinct parts, as shown in Figure 7b and Figure 7d, with each element presented in a unique color. However, in Figure 7d, when the person and the pier were in close proximity, the clustering algorithm

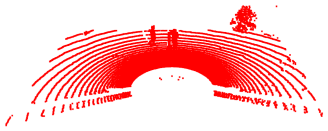
was unable to distinguish them as separate entities and instead recognized them as a single element.



(a) Original data, person far from the pier



(b) Ground removed, person far from pier



(c) Original data, person near the pier

(d) Ground removed, person near the pier

Fig. 7: LiDAR Result: Comparison Between Original Data and Processed Data

3.3 Fusion Result

The team fused the detection results from the first frame of the video and the first file of the LiDAR, as shown in Figure 8. The final result of sensor fusion is presented in Figure 9.

Fig. 8: Camera Result and LiDAR Result Before Fusion

Fig. 9: Fusion Result

The blue box in Figure 9 represents the camera detection result for the first frame of the video. The detection result for the first LiDAR file is that there exist three clusters in this figure, i.e. three different elements. After computing the IoU values between the camera bounding box and three LiDAR bounding boxes, the team derived the cost matrix as shown below:

$$cost = \begin{bmatrix} 1 & 0.8515 & 1 \end{bmatrix}. \tag{9}$$

By implementing the linear assignment, the team successfully matched the camera bounding box with the second LiDAR bounding box, as shown as the red box in Figure 9.

The results in Figure 9 show that using the fusion algorithm above, the detection results match relatively well and the two bounding boxes are overlapping to a great extent. However, it is also noted that there are significant differences between two bounding boxes in terms of their positions and widths. One of the possible reasons for this might be the inconsistent timing between the first video frame and the first LiDAR file. As a result, LiDAR may not be able to provide precise spatial coordinates for the camera bounding box at this specific moment.

4 Discussion

Achieving accurate alignment between the video and LiDAR data is crucial for various applications such as object detection, localization, and tracking. Ideally, each frame of the video should correspond to each file of the LiDAR, as this would provide precise spatial and temporal synchronization between the two modalities. However, the dataset that the team used in this project lacked a time stamp, which made it challenging to establish a direct correspondence between the video frames and LiDAR files.

Due to this limitation, the team was unable to perform real-time processing. Nevertheless, the team was able to match the first frame of the video with the first file of the LiDAR, which allowed them to explore the potential of the algorithm despite the limitation.

To overcome the limitation, the team could acquire a new dataset that includes a time stamp or other temporal reference for both the video and LiDAR data. This would enable the team to establish a direct correspondence between each video frame and its corresponding LiDAR file, allowing for precise spatial and temporal alignment.

References

[1] O'Shea, K., Nash, R.: An introduction to convolutional neural networks. arXiv preprint arXiv:1511.08458 (2015)

[2] Wong, A., Famuori, M., Shafiee, M.J., Li, F., Chwyl, B., Chung, J.: Yolo nano: a highly compact you only look once convolutional neural network for object detection. arXiv preprint arXiv:2010.16162 (2020)

[3] Xu, J., Li, Z., Du, B., Zhang, M., Liu, J.: Reluplex made more practical: Leaky relu. In: 2020 IEEE Symposium on Computers and Communications (ISCC), pp. 1–7 (2020). https://doi.org/10.1109/ISCC50000.2020.9219587

[4] Kingma, D.P., Ba, J.L.: Adam: A method for stochastic optimization. arXiv preprint arXiv:1412.6980 (2014)

[5] Lin, T.-Y., Goyal, P., Girshick, R., He, K., Dollar, P.: Focal loss for dense object detection. In: Proceedings of the IEEE International Conference on Computer Vision (ICCV), pp. 2980–2988 (2017)

[6] Schubert, E., Sander, J., Ester, M., Kriegel, H.P., Xu, X.: Dbscan revisited, revisited: why and how you should (still) use dbscan. ACM Transactions on Database Systems (TODS) **42**(3), 1–21 (2017)